Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών,
Τηλεπικοινωνιών και Δικτύων

Διπλωματική Εργασία

«Ανάλυση ημιτονοειδούς μόνιμης κατάστασης για μεγάλης κλίμακας γραμμικά κυκλώματα»

«Sinusoidal steady state analysis for large scale linear circuits»

Γάκη Στυλιανή

Επιβλέποντες: Ευμορφόπουλος Νέστωρ
Λέκτορας

Σταμούλης Γεώργιος
Καθηγητής

Βόλος, 2012

Διπλωματική Εργασία για την απόκτηση Διπλώματος Μηχανικού Ηλεκτρονικών Υπολογιστών Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας,
στα Πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας.

. . . . . .

Γάκη Στυλιανή
Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων

Στην οικογένειά μου

Ευχαριστίες

Με την περάτωση της παρούσης εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Νέστορα Ευμορφόπουλο, για την εμπιστοσύνη που μου έδειξε, την άριστη συνεργασία και τις διαρκείς και εύστοχες υποδείξεις του που βοήθησαν στην έγκαιρη ολοκλήρωση αυτής της μελέτης. Επιπρόσθετα, θα ήθελα να ευχαριστήσω τον έτερο επιβλέποντα καθηγητή μου, κ. Γεώργιο Σταμούλη, για τις συμβουλές του.

Επίσης, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα του τμήματος Κωνσταντή Νταλούκα καθώς και τον συμφοιτητή μου Ιωαννίδη Σταύρο για τις πολύτιμες συμβουλές τους και την στήριξη τους, στοιχεία που έπαιξαν σημαντικό ρόλο στην ολοκλήρωση της παρούσης εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη υποστήριξη και κατανόηση που έδειξαν όσο κατά την διάρκεια εκπόνησης της παρούσης εργασίας τόσο και κατά την διάρκεια των σπουδών μου.

ΓΑΚΗ ΣΤΥΛΙΑΝΗ
Βόλος, 2012

Περιεχόμενα

Κατάλογος Αλγορίθμων

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

Κατάλογος Συντομογραφιών

AC	Alternating Current
ARMS	Algebraic Recursive Multilevel Solver
ARMS-ddPQ	ARMS with ideally diagonally dominant B block and P,Q permutations
Bi-CG	BiConjugate Gradient
CG	Conjugate Gradient
CSC	Compressed Column
CSR	Compressed Row
DC	Direct Current
IBM	International Business Machines
ILU	Incomplete LU
ILUK	ILU with level of fill K
ILUT	ILU with Threshold dropping
ILUTP	ILUT with column Pivoting
KCL	Kirchhoff Current Law
KVL	Kirchhoff Voltage Law
MNA	Modified Nodal Analysis
NZP	Non-Zero Pattern
PILU	Partial ILU
SoC	System on Chip

Περίληψη

Σε αυτήν την διπλωματική εργασία αρχικά παρουσιάζεται μία υλοποίηση δύο επαναληπτικών μεθόδων Συζυγών Κλίσεων, γνωστές ως CG και Bi-CG, για την επίλυση μεγάλων μιγαδικών αραιών γραμμικών συστημάτων. Σε αυτές τις επαναληπτικές μεθόδους εφαρμόζονται τεχνικές προρρύθμισης για μιγαδικούς πίνακες οι οποίες βασίζονται στην ατελή LU παραγοντοποίηση. Ακολούθως, παρουσιάζεται μία συγκριτική μελέτη των προρρυθμιστών ILUK, ILUT, ILUTP και ARMS όπου συγκρίνουμε την απόδοση τους ως προς τον χρόνο υπολογισμού τους, καθώς και τον συνολικό χρόνο εκτέλεσης και το πλήθος επαναλήψεων που απαιτούνται για την σύγκλιση των επαναληπτικών μεθόδων. Τα αριθμητικά αποτελέσματα αναλύονται για μία συγκεκριμένη περιοχή εφαρμογής που αφορά τα μεγάλα μιγαδικά αραιά γραμμικά συστήματα που προκύπτουν κατά την ανάλυση δικτύων διανομής ισχύος ολοκληρωμένων κυκλωμάτων. Επιπρόσθετα, γίνεται μία σύντομη σύγκριση των παραπάνω επαναληπτικών μεθόδων με την άμεση LU μέθοδο επίλυσης για την προαναφερθείσα περιοχή εφαρμογής.

Λέξεις Κλειδιά:
Προσομοίωση Κυκλωμάτων, Μιγαδικά Αραιά Συστήματα, CG, Bi-CG, Προρρυθμιστές Κατάστασης, ILU, ILUK, ILUT, ILUTP, ARMS, Δίκτυα Τροφοδοσίας.

Summary

This thesis primarily presents an implementation of two iterative solvers, namely CG and Bi-CG for the solution of large complex sparse linear systems. In these iterative solvers, preconditioning techniques for complex-valued matrices are applied which are based on incomplete LU decomposition. Then, a comparative study of ILUK, ILUT, ILUTP and ARMS is presented where we compare their performance in terms of time for computing each of them needed, as well as in terms of the total runtime and the number of iterations needed for the convergence of the iterative solvers. The numerical results are analysed on a specific application area regarding the large complex sparse liner systems obtained from the analysis of power grids of integrated circuits. A brief comparison between the above iterative solvers and the direct LU solver is also addressed for the aforementioned application area.

Keywords:
Circuit Simulation, Complex Linear Systems, CG, BiCG, Preconditioning, ILU, ILUK, ILUT, ILUTP, ARMS, Power Grid.

Εισαγωγή

Σκοπός της παρούσης διπλωματικής εργασίας είναι αρχικά η ανάπτυξη ενός εργαλείου για την επίλυση του γραμμικού συστήματος της μορφής:

$$A\,x = b$$

όπου πίνακας συντελεστών $A \in \mathbb{C}^{N\,x\,N}$ και είναι αραιός και εν δυνάμει μεγάλος. Για το σκοπό αυτό, η γνωστή επαναληπτική μέθοδος των συζυγών κλίσεων, γνωστή ως CG, καθώς και μία ευρετική επέκταση αυτής, γνωστή ως Bi-CG, υλοποιούνται σε συνδυασμό με χρήση κάποιου προρρυθμιστή κατάστασης. Γενικότερα, οι προρρυθμιστές έχουν ευρέως μελετηθεί για την περίπτωση της πραγματικής αριθμητικής, δηλαδή στην περίπτωση που ο $A \in \mathbb{R}^{Nx\,N}$, ενώ η συμπεριφορά τους στην μιγαδική περίπτωση είναι λιγότερο γνωστή. Στην παρούσα εργασία θα μελετήσουμε μία συγκεκριμένη κατηγορία προρρυθμιστών που βασίζονται στην ατελή LU παραγοντοποίηση. Συγκεκριμένα, θα μελετήσουμε τους προρρυθμιστές ILUK, ILUT, ILUTP και ARMS και θα δούμε την συμπεριφορά τους στην μιγαδική περίπτωση.

Οι προρρυθμιστές κατάστασης αποτελούν ένα πολύ βασικό στοιχείο για την βελτίωση της ταχύτητας των επαναληπτικών μεθόδων, αφού η ταχύτητα σύγκλισης των επαναληπτικών μεθόδων εξαρτάται από τις ιδιότητες του φάσματος και των ιδιοτιμών του πίνακα των συντελεστών A του γραμμικού συστήματος προς επίλυση. Η ανάγκη για αυτή την βελτίωση έγκειται στο γεγονός ότι ένας μεγάλος αριθμός σύγχρονων εφαρμογών αποτελούν πρόκληση για τις επαναληπτικές μεθόδους, αφού η επίλυση του γραμμικού συστήματος που παράγουν είναι ιδιαίτερα χρονοβόρα λόγω του τεράστιου όγκου υπολογισμών που απαιτούνται. Συχνά δε τα προκύπτοντα γραμμικά συστήματα είναι μιγαδικά. Μία από αυτές τις εφαρμογές αποτελούν τα δίκτυα διανομής ισχύος ή πιο απλά τα δίκτυα τροφοδοσίας, τα οποία είναι γίνονται όλο και μεγαλύτερα καθώς οι απαιτήσεις ισχύος στα ολοκληρωμένα κυκλώματα αυξάνονται καθώς η τεχνολογία αναπτύσσετε. Για αυτήν την περίπτωση, θα εξηγήσουμε αναλυτικά την σημαντικότητα αποδοτικής ανάλυσής της και θα παρουσιάσουμε μία συγκριτική μελέτη των προρρυθμιστών ILUK, ILUT, ILUTP και ARMS για την επίλυση των προκυπτόντων μεγάλων αραιών μιγαδικών γραμμικών συστημάτων. Η απόδοση των προρρυθμιστών αυτών θα συγκριθεί από άποψη του χρόνου υπολογισμού τους, καθώς και από άποψη του συνολικού χρόνου εκτέλεσης και πλήθος επαναλήψεων που απαιτούνται για την σύγκλιση των επαναληπτικών μεθόδων. Για την ίδια περιοχή εφαρμογής, θα δώσουμε επιπρόσθετα μία σύντομη συγκριτική αναφορά για την απόδοση της άμεσης LU μεθόδου επίλυσης σε σχέση με τις προαναφερθείσες επαναληπτικές μεθόδους.

<u>Διάρθρωση της Διπλωματικής Εργασίας</u>

Η παρουσίαση της παρούσης εργασίας κινείται σε τρεις βασικούς άξονες. Αρχικά, στο 1ο μέρος και συγκεκριμένα στα Κεφάλαια 1, 2 και 3, γίνεται η ανάλυση θεωρητικών. Στην συνέχεια, στο 2ο μέρος και στα Κεφάλαια 4 και 5 παρουσιάζονται τα βασικά στοιχεία των έτοιμων βιβλιοθηκών που θα χρησιμοποιήσουμε και δίνονται όλες οι υλοποιήσεις αλγορίθμων και τεχνικών που παρουσιάζονται κατά την ανάλυση των θεωρητικών. Τονίζουμε πως όλες οι υλοποιήσεις έγιναν με ανάπτυξη κώδικα σε γλώσσα C. Τέλος, στο 3ο μέρος και στα Κεφάλαια 6 και 7 εξηγούμε αναλυτικά την σημαντικότητα αποδοτικής ανάλυσης για την περιοχή εφαρμογής των δικτύων τροφοδοσίας και παρουσιάζονται και αναλύονται τα προκύπτοντα αριθμητικά αποτελέσματα της παρούσης μελέτης. Στην συνέχεια παρουσιάζουμε περιγραφικά τα βασικά στοιχεία κάθε κεφαλαίου της παρούσης εργασίας.

Στο Κεφάλαιο 1 περιγράφεται ο τρόπος με τον οποίο εφαρμόζουμε την μέθοδο της Τροποποιημένης Ανάλυσης Κόμβων για την διαμόρφωση του γραμμικού συστήματος προς επίλυση που περιγράφει την συμπεριφορά ενός ηλεκτρικού κυκλώματος και ειδικότερα πως αυτή διαμορφώνεται στην μιγαδική περίπτωση της ανάλυσης εναλλασσόμενου ρεύματος την οποία μελετάμε. Επισημαίνουμε πως για την υλοποίηση αυτού του εργαλείου έγινε χρήση συναρτήσεων του CXSparce πακέτου που περιέχει συναρτήσεις διαχείρισης αραιών μιγαδικών πινάκων και συστημάτων. Λεπτομέρειες για την βιβλιοθήκη αυτή, καθώς και ο αντίστοιχος κώδικας της υλοποίησης, δίνονται στο 2ο μέρος της παρούσης εργασίας.

Στο Κεφάλαιο 2 αρχικά δίνονται τα βασικά στοιχεία της άμεσης LU μεθόδου και των επαναληπτικών μεθόδων επίλυσης. Ακολούθως, παρουσιάζονται και περιγράφονται τα βασικά στοιχεία που αφορούν τόσο την χρησιμότητα, όσο και τον τρόπο εφαρμογής των προρρυθμιστών κατάστασης στις επαναληπτικές μεθόδους. Στην συνέχεια, παρουσιάζεται η προρρυθμισμένη επαναληπτική μέθοδος Συζυγών Κλήσεων καθώς και μία ευρετική επέκταση αυτής.

Στο Κεφάλαιο 3 παρουσιάζονται και περιγράφονται αναλυτικά οι προρρυθμιστές κατάστασης που είναι γνωστοί στην βιβλιογραφία ως ILUK, ILUT, ILUTP, ARMS και ARMS-ddpQ. Οι προρρυθμιστές αυτοί βασίζονται στην ατελή LU παραγοντοποίηση (ILU). Επισημαίνουμε πως ο ILUTP αποτελεί επέκταση του ILUT, καθώς και ο ARMS-ddpQ αποτελεί επέκταση του ARMS. Ιδιαίτερη περίπτωση συνιστούν οι δύο τελευταίοι, ARMS και ARMS-ddpQ, οι όποιοι αποτελούν πολυεπίπεδους ILU προρρυθμιστές οι οποίοι εφαρμόζουν μία τεχνική αναδρομικής επίλυσης για την εύρεση του προρρυθμιστή. Επισημαίνουμε πως για την εφαρμογή των προρρυθμιστών αυτών έγινε χρήση της βιβλιοθήκης ZITSOL η οποία παρέχει συναρτήσεις που υλοποιούν τους προαναφερθέντες προρρυθμιστές για την επίλυση αραιών μιγαδικών συστημάτων. Οι συναρτήσεις αυτές είναι υλοποιημένες σε γλώσσα C. Λεπτομέρειες για την βιβλιοθήκη αυτή , καθώς και ο αντίστοιχος κώδικας της ενσωμάτωσης της και χρήσης των συναρτήσεων της στο εργαλείο μας, δίνονται στο 2ο μέρος της παρούσης εργασίας.

Στο Κεφάλαιο 4 δίνονται τα βασικά στοιχεία υλοποίησης των πακέτων CXSparce και ZITSOL, καθώς και η περιγραφή των συναρτήσεων τους τις οποίες χρησιμοποιούμε στο εργαλείο μας.

Στο Κεφάλαιο 5 αρχικά περιγράφεται η απαιτούμενη μορφή του αρχείου εισόδου το οποίο θα περιέχει την περιγραφή του κυκλώματος και στην συνέχεια δίνεται η πρακτική υλοποίηση του εργαλείου προσομοίωσης κυκλωμάτων που αναπτύξαμε εδικά για την μιγαδική περίπτωση που προκύπτει για την εφαρμογή ανάλυσης εναλλασσόμενου ρεύματος.

Στο Κεφάλαιο 6 εξηγούμε αναλυτικά την περιοχή εφαρμογής των δικτύων τροφοδοσίας και την σημαντικότητα αποδοτικής ανάλυσης των προκυπτόντων γραμμικών συστημάτων στην περίπτωση αυτή. Επίσης, δίνουμε αναλυτική περιγραφή των στοιχείων που αφορούν τα προβλήματα αναφοράς(benchmarks) που χρησιμοποιήσαμε για τα αποτελέσματα της αριθμητικής ανάλυσης.

Στο Κεφάλαιο 7 επικεντρωνόμαστε ώστε να συγκρίνουμε την βελτίωση της απόδοσης και της αξιοπιστίας των επαναληπτικών μεθόδων CG και Bi-CG αντίστοιχα για την χρήση των διάφορων προρρυθμιστών κατάστασης για την επίλυση των προκυπτόντων γραμμικών συστημάτων στην περίπτωση των δικτύων τροφοδοσίας. Επιπρόσθετα, συγκρίνουμε την απόδοση αυτών σε σχέση με την απόδοση της άμεσης LU μεθόδου επίλυσης.

1ο μέρος : Ανάλυση Θεωρητικών

Κεφάλαιο 1

Ανάλυση Κυκλώματος

1.1. Τροποποιημένη Ανάλυση Κόμβων

Η συμπεριφορά ενός ηλεκτρικού κυκλώματος περιγράφεται από ένα σύνολο εξισώσεων που προκύπτουν συνδυάζοντας τις εξισώσεις των ηλεκτρικών στοιχείων και τους νόμους του Kirchhoff. Αυτό προκύπτει από το γεγονός ότι η αριθμητική προσομοίωση ενός ηλεκτρικού κυκλώματος είναι παρόμοια με την μοντελοποίηση των δικτύων, δηλαδή περιγράφουμε το κύκλωμα σαν ένα γράφημα από κλάδους και κόμβους. Κάθε κλάδος αντιπροσωπεύει ένα ηλεκτρικό στοιχείο του οποίου τα άκρα συνδέονται στους κόμβους. Τα απλά γραμμικά στοιχεία δύο ακροδεκτών, δηλαδή οι αντιστάσεις, οι πυκνωτές, τα πηνία, οι πηγές τάσης και οι πηγές ρεύματος, μπορούν να περιγραφούν πλήρως με μία σχέση μεταξύ του ρεύματος ενός κλάδου το οποίο θα συμβολίζουμε με i και της αντίστοιχης τάσης του κλάδου την οποία θα συμβολίζουμε με υ. Αυτές είναι οι γνωστές μας χαρακτηριστικές εξισώσεις. Έτσι, ένα κύκλωμα το οποίο αποτελείται από αυτά τα απλά γραμμικά στοιχεία, μπορεί να περιγραφεί πλήρως με τις τάσεις και τα ρεύματα των κλάδων και τα δυναμικά των κόμβων. Συνεπώς, η τοπολογία μπορεί να περιγραφεί με χρήση των νόμων του Kirchhoff.

Αρχικά, υπενθυμίζουμε τις χαρακτηριστικές εξισώσεις των γραμμικών στοιχείων δύο ακροδεκτών:

- Αντίσταση: Αντιστέκεται στην ροή του ηλεκτρικού ρεύματος ακολουθώντας τον νόμο $\upsilon = R\, i$, όπου R είναι η τιμή της αντίστασης.
- Πυκνωτής: Αποθηκεύει ενέργεια σε ένα ηλεκτροστατικό πεδίο ακολουθώντας τον νόμο $q = C\, \upsilon$, όπου q είναι το ηλεκτρικό φορτίο και C είναι η τιμή της χωρητικότητας. Η χαρακτηριστική εξίσωση ρεύματος-τάσης δίνεται από την σχέση $i = \frac{d}{dt}\, q = C\, \frac{d\upsilon}{dt}$ (για C σταθερό).
- Πηνίο: Αποθηκεύει ενέργεια σε ένα ηλεκτρομαγνητικό πεδίο ακολουθώντας τον νόμο $\Phi = L\, i$, όπου Φ είναι η μαγνητική ροή και L είναι η τιμή της αυτεπαγωγής. Η χαρακτηριστική εξίσωση ρεύματος-τάσης δίνεται από την σχέση $\upsilon = \frac{d}{dt}\, \Phi = L\, \frac{di}{dt}$ (για L σταθερό).
- Πηγή Τάσης: Η χαρακτηριστική εξίσωση ρεύματος-τάσης δίνεται από την σχέση $\upsilon = E$, όπου E είναι η ένταση της πηγής. Σημειώνουμε ότι το υ δεν εξαρτάται από το ρεύμα του κλάδου i.
- Πηγή Ρεύματος: Η χαρακτηριστική εξίσωση ρεύματος-τάσης δίνεται από την σχέση $i = I$, όπου I είναι η ένταση της πηγής. Σημειώνουμε ότι το i δεν εξαρτάται από την τάση του κλάδου υ.

Στην συνέχεια θα δείξουμε πως με την χρήση των νόμων του Kirchhoff θα καταλήξουμε σε ένα γραμμικό σύστημα εξισώσεων που θα περιγράφει την τοπολογία του ενός κυκλώματος το οποίο αποτελείται από αυτά τα απλά γραμμικά στοιχεία.

Από τον νόμο ρευμάτων του Kirchhoff (Kirchhoff Current Law, KCL) γνωρίζουμε ότι το αλγεβρικό άθροισμα των ρευμάτων σε έναν κόμβο του κυκλώματος, σε οποιαδήποτε

χρονική στιγμή, είναι ίσο με μηδέν. Επιπλέον, από τον νόμο τάσεων του Kirchhoff (Kirchhoff Volt Law, KVL) γνωρίζουμε ότι κατά μήκος οποιασδήποτε κλειστής διαδρομής πάνω σε ένα συγκεντρωμένο κύκλωμα, το αλγεβρικό άθροισμα των τάσεων κατά μήκος της διαδρομής αυτής είναι μηδέν σε κάθε χρονική στιγμή. Σε ένα πρακτικό κύκλωμα υπάρχουν πάρα πολλοί κόμβοι και κλειστές διαδρομές. Είναι προφανές ότι δεν πρακτικό να γράφουμε για κάθε έναν κόμβο τον KCL και για κάθε κλειστή διαδρομή τον KVL αντίστοιχα. Ευτυχώς υπάρχει ένας πιο συστηματικός τρόπος για να το κάνουμε αυτό. Θεωρώντας ότι ένα κύκλωμα έχει n κόμβους και k ανοιχτές διαδρομές, αριθμούμε αντίστοιχα τους κόμβους και τις ανοιχτές διαδρομές, παραλείποντας τον κόμβο της γείωσης. Επιπλέον για κάθε κλάδο ορίζουμε μία φορά λαμβάνοντας τον έναν κόμβο του κλάδου ως αρχικό και τον άλλο ως τελικό. Έτσι στον ελαττωμένο πίνακα πρόσπτωσης θα έχουμε:

$$a_{kl} = \begin{cases} 1, & \textit{αν ο κλάδος } kl \textit{ έχει αρχικό κόμβο τον } k \\ -1, & \textit{αν ο κλάδος } kl \textit{ έχει τελικό κόμβο τον } k \\ 0, & \textit{αλλιώς} \end{cases} \tag{1.1}$$

Οπότε τώρα η συμπαγής μορφή του KCL θα είναι:

$$A\, i = 0 \tag{1.2}$$

όπου $i = \left[i_1, i_2, \dots, i_k\right]$ είναι το διάνυσμα των ρευμάτων των κλάδων.
Επίσης, με χρήση του ελαττωμένου πίνακα πρόσπτωσης μπορούμε να έχουμε και μία απλή περιγραφή για την σχέση μεταξύ των δυναμικών των κόμβων και τις τάσεις των κλάδων, που θα είναι:

$$\upsilon = A^T\, e \tag{1.3}$$

όπου $\upsilon = \left[\upsilon_1, \upsilon_2, \dots, \upsilon_k\right]$ είναι το διάνυσμα των τάσεων των κλάδων και $e = \left[e_1, e_2, \dots, e_{n-1}\right]$ είναι το διάνυσμα των δυναμικών κόμβων.

Αν επιχειρήσουμε τώρα να γράψουμε όλες τις εξισώσεις του κυκλώματος που προκύπτουν από τις (1.2) και (1.3) και τις χαρακτηριστικές εξισώσεις όλων των στοιχείων του κυκλώματος θα έχουμε ένα αραιό σύστημα με $2k + n - 1$ αγνώστους. Όμως μπορούμε να μειώσουμε το πλήθος των αγνώστων χρησιμοποιώντας την *Τροποποιημένη ανάλυση κόμβων (Modified Nodal Analysis, MNA)* [1,2]. Σε αυτή την περίπτωση, αντικαθιστούμε τα ρεύματα κλάδων των στοιχείων που το ρεύμα που τα διαρρέει δεν απαλείφεται από την χαρακτηριστική τους εξίσωση, δηλαδή αντιστάσεις, πυκνωτές και πηγές ρεύματος, με την χαρακτηριστική τους εξίσωση και τις τάσεις των κλάδων με τις τάσεις των κόμβων. Για ευκολία στους υπολογισμούς αναλύουμε τον ελαττωμένο πίνακα πρόσπτωσης ως εξής:

$$A = \left[A_R, A_C, A_L, A_V, A_I\right] \tag{1.4}$$

όπου οι δείκτες αντιστοιχούν στα στοιχεία: αντίσταση, πυκνωτής, πηνίο, πηγή τάσης και πηγή ρεύματος αντίστοιχα. Έτσι, χρησιμοποιώντας τις χαρακτηριστικές εξισώσεις των στοιχείων, προκύπτει το ακόλουθο γενικό σύστημα MNA:

$$\begin{bmatrix} A_C C\, A_C^T & 0 & 0 \\ 0 & L & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{de}{dt} \\ \frac{di_L}{dt} \\ \frac{di_V}{dt} \end{bmatrix} + \begin{bmatrix} A_R\, G\, A_R^T & A_L & A_V \\ -A_L^T & 0 & 0 \\ A_V^T & 0 & 0 \end{bmatrix} \begin{bmatrix} e \\ i_L \\ i_V \end{bmatrix} = \begin{bmatrix} -A_I\, I \\ 0 \\ E \end{bmatrix} \tag{1.5}$$

όπου i_V είναι το διάνυσμα των ρευμάτων των κλάδων που "περιέχουν" πηγή τάσης, i_L είναι το διάνυσμα των ρευμάτων των κλάδων που "περιέχουν" πηνίο, I είναι το διάνυσμα των τιμών όλων των πηγών ρεύματος, E είναι το διάνυσμα των τιμών όλων των πηγών τάσης, C είναι ένας διαγώνιος πίνακας των χωρητικοτήτων κλάδων, G είναι ο διαγώνιος πίνακας των αγωγιμοτήτων κλάδων και L είναι ο διαγώνιος πίνακας των αυτεπαγωγών κλάδων.

Σημειώνουμε πως στην πράξη το MNA σύστημα διαμορφώνεται διαισθητικά, σε χρόνο γραμμικό, καθώς διαβάζεται το αρχείο περιγραφής του κυκλώματος. Αυτό βασίζεται στην ιδέα ότι για κάθε κυκλωματικό στοιχείο υπάρχει μια "συνεισφορά" στον πίνακα εξισώσεων η οποία ονομάζεται "σφραγίδα" του στοιχείου. Με βάση αυτήν την συνεισφορά διαμορφώνεται το MNA σύστημα.

Στην συνέχεια, θα δούμε πως από το γενικό σύστημα MNA λαμβάνουμε το τελικό σύστημα MNA για την περίπτωση που θέλουμε να εκτελέσουμε ανάλυση εναλλασσόμενου ρεύματος (Alternating Current, AC analysis) και θα δώσουμε αναλυτικότερα τις συνεισφορές του κάθε στοιχείου.

1.1.1. AC ανάλυση

Στα πλαίσια της εργασίας αυτής, θα θεωρήσουμε πως όλες οι πηγές είναι αρμονικές και έχουν κοινή συχνότητα f, δηλαδή για όλες τις πηγές ισχύει:

$$\upsilon(t) = \hat{V}\, e^{\,j(\omega t+\varphi)} = \hat{V}\,[\cos(\omega t+\varphi) + j\sin(\omega t+\varphi)] \tag{1.6}$$

και

$$i(t) = \hat{I}\, e^{j(\omega t+\varphi)} = \hat{I}\,[\cos(\omega t+\varphi) + j\sin(\omega t+\varphi)] \tag{1.7}$$

όπου $\omega = 2\pi f$ είναι η κυκλική συχνότητα, φ είναι η αρχική φάση και $\hat{V} = \sqrt{2}V$ και $\hat{I} = \sqrt{2}I$ είναι οι ενεργές τιμές της τάσης και του ρεύματος αντίστοιχα. Κάνοντας αυτές τις θεωρήσεις, είναι βολικό να δουλέψουμε στο πεδίο των συχνοτήτων και να χρησιμοποιήσουμε μιγαδικές ποσότητες. Στο πεδίο της συχνότητας, για τα στοιχεία του κυκλώματος έχουμε:

- Για όλες τις πηγές ισχύει:

$$V = \hat{V}\, e^{\,j\varphi} = \hat{V}\,[\cos\varphi + j\sin\varphi] \tag{1.8}$$

και

$$I = \hat{I}\, e^{\,j\varphi} = \hat{I}\,[\cos\varphi + j\sin\varphi] \tag{1.9}$$

με το πραγματικό μέρος να είναι το $V_{Re} = \hat{V}\cos\varphi$ και $I_{Re} = \hat{I}\cos\varphi$ αντίστοιχα και το φανταστικό μέρος να είναι το $V_{Im} = \hat{V}\sin\varphi$ και $I_{Im} = \hat{I}\sin\varphi$ αντίστοιχα. Οπότε, σε τριγωνομετρική μορφή θα είναι:

$$V = V_{Re} + j\,V_{Im} \quad \text{και} \quad I = I_{Re} + j\,I_{Im} \tag{1.10}$$

Αντίστοιχα σε πολική μορφή θα είναι:

$$\hat{V}\angle\varphi \quad \text{και} \quad \hat{I}\angle\varphi \tag{1.11}$$

όπου $\hat{V} = \sqrt{(_{V_{Re}})^2 + (_{V_{Im}})^2}$ και $\varphi = \tan^{-1}(V_{Im}/V_{Re})$. Ομοίως για το ρεύμα.

- Οι χωρητικότητες C αντικαθίστανται με χωρητικές αντιδράσεις $1/j\omega C$ ή χωρητικές επιδεκτικότητες $j\omega C$.

- Οι αυτεπαγωγές L αντικαθίστανται με επαγωγικές αντιδράσεις $j\omega L$ ή επαγωγικές επιδεκτικότητες $1/j\omega L$.

- Οι αντιστάσεις R και οι αγωγιμότητες G μένουν ως έχουν.

Αποδεικνύεται επίσης ότι στο πεδίο της συχνότητας ισχύουν οι νόμοι των τάσεων και των ρευμάτων του Kirchhoff, πάντα με βάση τις υποθέσεις που κάναμε. Επίσης, οι άγνωστοι τώρα θα είναι μιγαδικοί και θα αναζητούμε το μέτρο και την φάση τους συναρτήσει της κυκλικής συχνότητας ω.

Για να διαμορφώσουμε το MNA σύστημα κάνουμε την εξής σημαντική παρατήρηση: οι κλάδοι με χωρητικότητες και οι κλάδοι με αυτεπαγωγές μπορούνε να αντιμετωπίζονται με τον ίδιο τρόπο όπως οι κλάδοι με αντιστάσεις. Με βάση αυτό, το προκύπτον MNA σύστημα θα είναι:

$$\underbrace{\begin{bmatrix} A_R Y_R A_R^T + A_C Y_C A_C^T + A_L Y_L A_L^T & A_V \\ A_V^T & 0 \end{bmatrix}}_{\text{πίνακας συντελεστών } A} \begin{bmatrix} e \\ i_V \end{bmatrix} = \begin{bmatrix} -A_I I \\ E \end{bmatrix} \qquad (1.12)$$

όπου $Y_R = G, Y_C$ και Y_L ο διαγώνιος πίνακας των αγωγιμοτήτων, των χωρητικών επιδεκτικοτήτων και των επαγωγικών επιδεκτικοτήτων αντίστοιχα. Επισημαίνουμε ότι το διάνυσμα των ρευμάτων και των τάσεων θα περιέχει τώρα ενεργές τιμές.

Όπως προαναφέραμε στη πράξη χρησιμοποιούνται "συνεισφορές" για την διαμόρφωση του MNA συστήματος. Πιο συγκεκριμένα, οι "συνεισφορές" των στοιχείων προστίθενται στον πίνακα συντελεστών A και στο διάνυσμα του δεξιού μέλους καθώς γίνεται η ανάγνωση του αρχείου περιγραφής του κυκλώματος μέσω του συντακτικού αναλυτή που έχουμε υλοποιήσει. Όταν όλα τα στοιχεία έχουν αναγνωστεί, το παραπάνω αραιό μιγαδικό γραμμικό σύστημα (1.12) έχει δημιουργηθεί και είναι έτοιμο προς επίλυση. Συμβολίζοντας τον αρχικό κόμβο ενός κλάδου με $\langle + \rangle$ και τον τελικό κλάδο ενός κόμβου με $\langle - \rangle$, οι συνεισφορές για κάθε στοιχείο θα είναι οι εξής [3]:

- <u>Αντίσταση</u>:
  Το στοιχείο $\langle + \rangle\langle + \rangle$: $+ {}^{1}/_{R} + 0\,j$
  Το στοιχείο $\langle + \rangle\langle - \rangle$: $- {}^{1}/_{R} + 0\,j$
  Το στοιχείο $\langle - \rangle\langle + \rangle$: $- {}^{1}/_{R} + 0\,j$
  Το στοιχείο $\langle - \rangle\langle - \rangle$: $+ {}^{1}/_{R} + 0\,j$

- <u>Πυκνωτής</u>:
  Το στοιχείο $\langle + \rangle\langle + \rangle$: $0 + \omega C\,j$
  Το στοιχείο $\langle + \rangle\langle - \rangle$: $0 - \omega C\,j$
  Το στοιχείο $\langle - \rangle\langle + \rangle$: $0 - \omega C\,j$
  Το στοιχείο $\langle - \rangle\langle - \rangle$: $0 + \omega C\,j$

- <u>Πηνίο</u>:

Το στοιχείο $\langle + \rangle \langle + \rangle : \ 0 - \left({}^{1}/_{\omega\, L}\right)\ j$
Το στοιχείο $\langle + \rangle \langle - \rangle : \ 0 + \left({}^{1}/_{\omega\, L}\right) j$
Το στοιχείο $\langle - \rangle \langle + \rangle : \ 0 + \left({}^{1}/_{\omega\, L}\right) j$
Το στοιχείο $\langle - \rangle \langle - \rangle : \ 0 - \left({}^{1}/_{\omega\, L}\right) j$

- Πηγή Τάσης:
  Το στοιχείο $\langle + \rangle \langle (n-1)+k \rangle : \ +1 + \ 0\, j$
  Το στοιχείο $\langle - \rangle \langle (n-1)+k \rangle : \ -1 + \ 0\, j$
  Το στοιχείο $\langle (n-1)+k \rangle \langle + \rangle : \ +1 + \ 0\, j$
  Το στοιχείο $\langle (n-1)+k \rangle \langle - \rangle : \ -1 + \ 0\, j$
  Και στο δεξί μέλος το στοιχείο $\langle (n-1)+k \rangle : \ V = \ V_{Re} + \ j\, V_{Im}$

- Πηγή Ρεύματος:
  Στο δεξί μέλος το στοιχείο $\langle + \rangle : \ I = I_{Re} - \ j\, I_{Im}$
  Στο δεξί μέλος το στοιχείο $\langle - \rangle : \ I = I_{Re} + \ j\, I_{Im}$

όπου n και k είναι το πλήθος των κόμβων και το πλήθος των πηγών τάσης του κυκλώματος, αντίστοιχα.

Κεφάλαιο 2

Μέθοδοι Επίλυσης Αραιών Γραμμικών Συστημάτων

2.1. Άμεση LU Μέθοδος Επίλυσης

Άμεσες μέθοδοι [4] ονομάζονται εκείνες που τερματίζουν σε προκαθορισμένο αριθμό πράξεων. Οι άμεσες μέθοδοι για την επίλυση αραιών πινάκων αφορούν πολύ πιο περίπλοκους αλγόριθμους σε σχέση με αυτούς για την επίλυση πυκνών πινάκων. Αυτό οφείλεται στο γεγονός ότι πρέπει να γίνεται αποδοτική διαχείριση της εισαγωγής νέων μη μηδενικών στοιχείων(fill-ins) των παραγόντων L και U. Γενικά, μία τυπική άμεση μέθοδος επίλυσης περιλαμβάνει τα εξής βήματα:

1. Αναδιάταξη των γραμμών και των στηλών με σκοπό την αποφυγή διαίρεσης με το μηδέν και για διατήρηση της αριθμητικής ακρίβειας, καθώς και για την μείωση της εισαγωγής νέων μη μηδενικών στοιχείων στους παράγοντες L και U, αντίστοιχα. Αυτή η διαδικασία είναι γνωστή ως συμβολική αναδιάταξη(symbolic ordering).
2. Καθορισμός της κατανομής των μη μηδενικών στοιχείων στους παράγοντες L και U. Αυτή η διαδικασία είναι γνωστή ως συμβολική παραγοντοποίηση (symbolic factorization) και είναι χρήσιμη για την διατήρηση της αραιής δομής των παραγόντων L και U.
3. Υπολογισμός των L και U παραγόντων. Αυτή η διαδικασία είναι γνωστή ως αριθμητική παραγοντοποίηση(numerical factorization).
4. Επίλυση των τριγωνικών συστημάτων $L\,y = b$ και $U\,x = y$ με προς τα εμπρός και προς τα πίσω αντικατάσταση, αντίστοιχα.

Συνήθως τα βήματα 1 και 2, τα οποία θα μπορούσαμε να τα δούμε και ως ένα βήμα, εμπλέκουν μόνο τους γράφους(graphs) των πινάκων και συνεπώς μόνο πράξεις ακεραίων. Τα βήματα 3 και 4 εμπλέκουν πράξεις κινητής υποδιαστολής, με το βήμα 3 να είναι το πιο χρονοβόρο βήμα της μεθόδου επίλυσης.

2.2. Προρρυθμισμένες Επαναληπτικές Μέθοδοι Επίλυσης

Επαναληπτικές ονομάζονται οι μέθοδοι εκείνες οι οποίες παράγουν μία ακολουθία λύσεων, οι οποίες προσεγγίζουν ολοένα και καλύτερα την λύση του γραμμικού συστήματος. Οι μέθοδοι αυτοί τερματίζουν με την χρήση κάποιου κριτηρίου σύγκλισης. Μία κατηγορία επαναληπτικών μεθόδων είναι οι μέθοδοι του Krylov. Αυτές διαφοροποιούνται στο γεγονός ότι αποφεύγουν τις πράξεις μεταξύ πινάκων και αντί για αυτό κάνουν πολλαπλασιασμούς πίνακα διανυσμάτων και δουλεύουν με τα προκύπτοντα διανύσματα. Στην παρούσα εργασία θα υλοποιήσουμε δύο από τις πιο γνωστές μεθόδους του Krylov: την απλή Μέθοδο Συζυγών Κλήσεων (Conjugate Gradient, CG) και μία ευρετική επέκταση αυτής (γνωστή ως BiConjugate Gradient, Bi-CG) [3]. Από αυτές, η πρώτη χρησιμοποιείται μόνο για την επίλυση γραμμικών συστημάτων όπου ο πίνακας των συντελεστών είναι συμμετρικός και θετικά ορισμένος, ενώ η δεύτερη δεν έχει αυτούς τους περιορισμούς. Η καλύτερη τεχνική για την βελτίωση της

απόδοσης αυτών των μεθόδων είναι η χρήση ενός προρρυθμιστή κατάστασης. Παρακάτω περιγράφουμε αναλυτικά την σημασία και τους τρόπους χρήσης των προρρυθμιστών.

2.2.1. Προρρυθμιστές Κατάστασης

Η σύγκλιση των επαναληπτικών μεθόδων που χρησιμοποιούμε εξαρτάται από τις ιδιότητες του πίνακα A του γραμμικού συστήματος:

$$A\,x = b \tag{2.1}$$

και είναι γρήγορη όταν δηλαδή ο φασματικός δείκτης κατάστασης για τον A είναι $K_2(A) \approx 1$. Αν ο πίνακας A τείνει να γίνει μη αντιστρέψιμος, δηλαδή ο φασματικός δείκτης κατάστασης για τον A είναι $K_2(A) \gg 1$, η σύγκλιση των επαναληπτικών μεθόδων μπορεί να γίνει πολύ αργή. Ένας τρόπος για την βελτίωση αυτού του προβλήματος είναι η μετατροπή του συστήματος (2.1) σε ένα ισοδύναμο σύστημα, δηλαδή σε ένα σύστημα που έχει την ίδια λύση με το αρχικό, με πιο ευνοϊκές ιδιότητες προς λύση με επαναληπτική μέθοδο. Η μετατροπή αυτή είναι γνωστή με τον όρο Προρρύθμιση Κατάστασης(Preconditioning) και ο προκύπτων πίνακας, έστω M, ονομάζεται Προρυθμιστής Κατάστασης(Preconditioner) [5, 6].

Έτσι το σύστημα (2.1) αντικαθίσταται από το ισοδύναμο σύστημα:

$$M^{-1}\,Ax = M^{-1}\,b \tag{2.2}$$

ή

$$AM^{-1}\,x = b, \qquad x = M^{-1}\,y \tag{2.3}$$

Το σύστημα αυτό θέλουμε να είναι καλύτερης κατάστασης από το αρχικό σύστημα (2.1). Δηλαδή για το (2.2) θέλουμε $K_2(M^{-1}\,A) \ll K_2(A)$ και ιδανικά $K_2(M^{-1}\,A) \approx K_2(I) = 1$. Ομοίως και για την περίπτωση προρρύθμισης από δεξιά. Επιπλέον, όταν ο Προρυθμιστής M είναι της μορφής $M = M_1\,M_2$, όπου M_1 και M_1είναι άνω και κάτω τριγωνικός πίνακας αντίστοιχα, το σύστημα (2.1) αντικαθίσταται από το ισοδύναμο σύστημα:

$${M_1}^{-1}\,A{M_2}^{-1}\,y = {M_1}^{-1}\,b, \qquad x = {M_2}^{-1}\,y \tag{2.4}$$

ο οποίος είναι και ο τύπος προρρύθμισης κατάστασης που θα χρησιμοποιήσουμε στην περίπτωση της χρήσης τεχνικών που βασίζονται στην ατελή παραγοντοποίηση την οποία και θα εξετάσουμε αναλυτικά, όπως ήδη έχουμε προαναφέρει. Λεπτομερής εφαρμογή του περιγράφεται στην επόμενη ενότητα.

Συγκεντρωτικά, επισημαίνουμε τις προϋποθέσεις που πρέπει να ικανοποιούνται ώστε να είναι αποδοτική η χρήση του Προρρυθμιστή:

- Ο M να προσεγγίζει τον Α ώστε να έχουμε $K_2(M^{-1}\,A) \approx K_2(I) = 1$.
- Ο υπολογισμός και η εφαρμογή του M να έχει χαμηλές απαιτήσεις σε υπολογιστικό κόστος.
- Ο M να έχει χαμηλές απαιτήσεις σε μνήμη.
- Το ισοδύναμο σύστημα, (2.2) ή (2.3) ή (2.4), θα πρέπει να συγκλίνει γρηγορότερα από το αρχικό σύστημα, ή πιο απλά να είναι ευκολότερο να λυθεί.

Είναι φανερό πως οι παραπάνω προϋποθέσεις συναγωνίζονται η μία την άλλη και έτσι είναι απαραίτητο να βρεθεί να επιτευχθεί μία ισορροπία μεταξύ αυτών.

Μετά την εφαρμογή τεχνικών προρρύθμισης που βασίζονται στην ατελή παραγοντοποίηση(ILU techniques) στον πίνακα A του συστήματος (2.1), θα έχουμε:

$$A = LU - R \tag{2.5}$$

όπου L και U είναι ένας κάτω και άνω τριγωνικός πίνακας αντίστοιχα και R είναι ο πίνακας με το υπόλοιπο ή σφάλμα(residual or error) της παραγοντοποίησης ο οποίος περιέχει τα απορριφθέντα στοιχεία. Λαμβάνοντας ως προρρυθμιστή τον $M = LU$, το (2.1) αντικαθίσταται από το ισοδύναμο σύστημα:

$$L^{-1} AU^{-1} y = L^{-1} b, \qquad x = U^{-1} y \tag{2.6}$$

2.2.2. Προρρυθμισμένες Μέθοδοι Συζυγών Κλίσεων

Όταν χρησιμοποιούνται μέθοδοι Krylov, όπως στην περίπτωση μας που χρησιμοποιούμε τις μεθόδους Συζυγών Κλίσεων, δεν είναι απαραίτητο να υπολογίσουμε ρητά τα γινόμενα των πινάκων $M^{-1} A$ ή AM^{-1} ή ${M_1}^{-1} A{M_2}^{-1}$. Κάτι τέτοιο θα ήταν πολύ ακριβό και θα χάναμε την αραιή δομή(sparsity) του πίνακα των συντελεστών του συστήματος προς επίλυση. Έτσι, αντί αυτών, η εφαρμογή του προρρυθμιστή ενσωματώνεται στον αλγόριθμο των επαναληπτικών μεθόδων με τον υπολογισμό γινόμενων πίνακα-διανύσματος με τον A και τις λύσεις του γραμμικού συστήματος της μορφής $M z = r$. Εδώ θα πρέπει να τονιστεί ότι, από πρώτης όψεως, το γεγονός ότι ένα σύστημα $M z = r$ θα λύνεται σε κάθε επανάληψη δίνει επιπλέον κόστος για την επαναληπτική μέθοδο. Στην πραγματικότητα αυτό δεν ισχύει γιατί δεν χρειάζεται να παραγοντοποιούμε τον πίνακα M σε κάθε επανάληψη, αφού παραμένει ο ίδιος σε κάθε επαναληπτικό βήμα. Έτσι η παραγοντοποίηση γίνεται μόνο στην αρχή κι αυτό που επαναλαμβάνεται ανά βήμα είναι οι προς τα πίσω αντικαταστάσεις με διαφορετικά δεξιά μέλη κάθε φορά. Οι πράξεις αυτές δεν έχουν πολύ κόστος και τα οφέλη που έχουμε ουσιαστικά ελαχιστοποιούν το επιπλέον κόστος.

Όπως αναφέραμε και παραπάνω, οι επαναληπτικές μέθοδοι που θα χρησιμοποιήσουμε θα είναι η απλή (CG) και η επεκταμένη (Bi-CG) μέθοδος συζυγών κλίσεων με χρήση προρρυθμιστή. Υπενθυμίζουμε πως η πρώτη χρησιμοποιείται για την επίλυση του συστήματος $A x = b$ όταν ο A είναι συμμετρικός και θετικά ορισμένος. Οι μέθοδοι αυτές βασίζονται στην ελαχιστοποίηση της συνάρτησης $f(\underline{x}) = \frac{1}{2} \underline{x}^T A \underline{x} - \underline{b}^T \underline{x}$. Πιο συγκεκριμένα, σε κάθε επανάληψη, έστω i, υπολογίζεται μία κατεύθυνση αναζήτησης της λύσης(search direction) $\underline{p}^{(i)}$ η οποία είναι κάθετη στην κατεύθυνση που υπολογίστηκε στην προηγούμενη επανάληψη. Η τρέχουσα προσέγγιση της λύσης $\underline{x}^{(i)}$, καθώς και το υπόλοιπο (residual) $\underline{r}^{(i)}$ μετακινείται προς αυτή την κατεύθυνση ως $\underline{x}^{(i)} = \underline{x}^{(i-1)} + a^{(i)} \underline{p}^{(i)}$, όπου $a^{(i)}$ είναι το βήμα που ελαχιστοποιεί την $f(\underline{x}^{(i)})$ επί όλων των δυνατών $\underline{x}^{(i)}$. Παρακάτω δίνεται ο αλγόριθμος για την CG επαναληπτική μέθοδο[3]:

1. $Find\ \underline{x} = initial\ guess\ \underline{x}^{(0)}$
2. $Compute\ initial\ residual: \underline{r} = \ \underline{b} - A\ \underline{x}$
3. $Construct\ Preconditoner\ M$
4. $iter = 0$
5. $While\ (\ \|\underline{r}\|/\|\underline{b}\| > itol\ \ and\ iter < n)\ Do:$
6. $\quad iter = iter + 1;$
7. $\quad Solve\ M\ \underline{z} = \ \underline{r}$
8. $\quad Compute\ rho = \ \underline{r}^T\ \underline{z}$
9. $\quad If\ (iter == 1)\ then$
10. $\quad\quad \underline{p} = \ \underline{z}$
11. $\quad Else$
12. $\quad\quad beta = rho/rho1$
13. $\quad\quad \underline{p} \ = \ \underline{z} + \ beta * \underline{p}$
14. $\quad End\ If$
15. $\quad rho1 = rho$
16. $\quad \underline{q} = A\ \underline{p}$
17. $\quad alpha = \ rho/\underline{p}^T\ \underline{q}$
18. $\quad \underline{x} = \ \underline{x} + \ alpha * \underline{p}$
19. $\quad \underline{r} = \ \underline{r} - \ alpha * \ \underline{q}$
20. $End\ While$

Αλγόριθμος 1: Επαναληπτική Μέθοδος CG

όπου n το μέγεθος του πίνακα συντελεστών Α. Παρακάτω δίνεται ο αλγόριθμος για την Bi-CG επαναληπτική μέθοδο[3]:

1. $Find\ \underline{x} = initial\ guess\ \underline{x}^{(0)}$
2. $Compute\ initial\ residual: \underline{r} = \ \tilde{\underline{r}} \ = \ \underline{b} - A\ \underline{x}$
3. $Construct\ Preconditoner\ M$
4. $Compute\ M^T$
5. $iter = 0$
6. $While\ (\ \|\underline{r}\|/\|\underline{b}\| > itol\ \ and\ iter < n)\ Do:$
7. $\quad iter = iter + 1;$
8. $\quad Solve\ M\ \underline{z} = \ \underline{r}$
9. $\quad Solve\ M^T\ \tilde{\underline{z}} = \ \tilde{\underline{r}}$
10. $\quad Compute\ rho = \ \underline{z}^T\ \tilde{\underline{r}}$
11. $\quad If\ (\ rho < EPS)\ then\ Exit$
12. $\quad If\ (iter == 1)\ then$
13. $\quad\quad \underline{p} = \ \underline{z}$
14. $\quad\quad \tilde{\underline{p}} = \ \tilde{\underline{z}}$
15. $\quad Else$
16. $\quad\quad beta = rho/rho1$
17. $\quad\quad \underline{p} \ = \ \underline{z} + \ beta * \underline{p}$
18. $\quad\quad \tilde{\underline{p}} \ = \ \tilde{\underline{z}} + \ beta * \tilde{\underline{p}}$
19. $\quad End\ If$
20. $\quad rho1 = rho$
21. $\quad \underline{q} = A\ \underline{p}$
22. $\quad \tilde{\underline{q}} = A\ \tilde{\underline{p}}$

23. $omega = \underline{\tilde{p}}^T \underline{q}$
24. $If\ (\ omega ga < EPS)\ then\ Exit$
25. $alpha = rho/omega$
26. $\underline{x} = \underline{x} + alpha * \underline{p}$
27. $\underline{r} = \underline{r} - alpha * \underline{q}$
28. $\underline{\tilde{r}} = \underline{\tilde{r}} - alpha * \underline{\tilde{q}}$
29. $End\ While$

Αλγόριθμος 2: Επαναληπτική Μέθοδος Bi-CG

Στην συνέχεια θα δώσουμε μία σειρά από προρρυθμιστές που βασίζονται στην ιδέα της τεχνικής της ατελούς παραγοντοποίησης και που είναι ενδιαφέρον να μελετήσουμε την χρήση τους με παραπάνω επαναληπτικές μεθόδους.

Κεφάλαιο 3

Προρρυθμιστές που βασίζονται στην Ατελή LU Παραγοντοποίηση

3.1. Ατελής LU Παραγοντοποίηση

Με την τεχνική της ατελούς LU παραγοντοποίησης(Incomplete LU factorization)[5] ουσιαστικά παράγεται μία προσέγγιση των πινάκων L και U της κλασικής απαλοιφής του Gauss[7,8]. Αυτό γίνεται απορρίπτοντας κάποια στοιχεία στους προσεγγιστικούς παράγοντες L και U, τα οποία βρίσκονται σε προκαθορισμένες θέσεις εκτός της διαγωνίου. Η απόρριψη των στοιχείων αυτών γίνεται με τον μηδενισμό τους. Για τον καθορισμό τον στοιχείων που θα απορριφθούν, διαλέγουμε εκ των προτέρων ένα μοτίβο μη-μηδενικών στοιχείων(non-zero pattern)[9], έστω NZP, τέτοιο ώστε:

$$NZP \supset \{(\mathrm{i},\mathrm{j}) \mid \mathrm{i} \neq \mathrm{j}, 1 \leq \mathrm{i},\mathrm{j} \leq \mathrm{n}\}$$

η ILU παραγοντοποίηση μπορεί να υπολογιστεί με τον παρακάτω γενικό αλγόριθμο[5]:

1. $For\ i = 1, \dots, n\ Do$:
2. $\quad For\ \ k = 0, \dots, i-1\ and\ if\ \ (i,k) \in NZP\ Do$:
3. $\quad\quad a_{ik} = a_{ik}/a_{kk}$
4. $\quad\quad For\ j = k+1, \dots, n\ and\ if\ (i,j) \in NZP\ Do$:
5. $\quad\quad\quad a_{ij} = a_{ij} - a_{ik}a_{kj}$
6. $\quad\quad End\ Do$
7. $\quad End\ Do$
8. $End\ Do$

Αλγόριθμος 3: Γενικός Αλγόριθμος ILU

Όσον αφορά την επιλογή του μοτίβου μη-μηδενικών στοιχείων, η πιο απλή περίπτωση είναι να το επιλέξουμε να είναι ακριβώς ίδιο με αυτό του αρχικού πίνακα A. Δηλαδή έχουμε:

$$NZP \equiv NZ(A) = \{(i,j) |\ a_{ij} \neq 0, 1 \leq \mathrm{i},\mathrm{j} \leq \mathrm{n}\}$$

Σημειώνουμε πως για χάριν απλότητας, από εδώ και κάτω θα θεωρήσουμε ότι ο αρχικός πίνακας A δεν έχει μηδενικά στην διαγώνιο. Αργότερα και κατά την υλοποίηση των αλγορίθμων, θα λάβουμε υπόψη μας αυτόν τον περιορισμό και θα κάνουμε τους κατάλληλους ελέγχους. Θεωρώντας το παραπάνω NZP,ο Αλγόριθμος 3 γίνεται:

1. $For\ i = 1, \dots, n\ Do$:
2. $\quad For\ \ k = 0, \dots, i-1\ and\ if\ \ (i,k) \in NZ(A)\ Do$:
3. $\quad\quad a_{ik} = a_{ik}/a_{kk}$
4. $\quad\quad For\ j = k+1, \dots, n\ and\ if \in\ NZ(A)\ \ Do$:
5. $\quad\quad\quad a_{ij} = a_{ij} - a_{ik}a_{kj}$
6. $\quad\quad End\ Do$
7. $\quad End\ Do$
8. $End\ Do$

Αλγόριθμος 4: ILU(0) Παραγοντοποίηση

Η παραπάνω τεχνική της ILU παραγοντοποίησης, είναι γνωστή ως ILU παραγοντοποίηση χωρίς βαθμό πληρότητας(Zero level of fill ILU) και συνήθως αναφέρεται με την συντομογραφία ILU(0).

3.1.1. Ατελής LU Παραγοντοποίηση με Βαθμό Πληρότητας (ILUK)

Με σκοπό την καλύτερη απόδοση και αξιοπιστία της ILU παραγοντοποίησης, ο παραπάνω αλγόριθμος επεκτάθηκε έτσι ώστε να επιτρέπει κάποιο βαθμό πληρότητας ο οποίος υποδεικνύεται από την παράμετρο k. Ο βαθμός πληρότητας αναφέρεται στην τάξη των εισαχθέντων νέων μη-μηδενικών στοιχείων κατά τον υπολογισμό της ILU παραγοντοποίησης. Αξίζει να σημειώσουμε πως η ιδέα πίσω από αυτό είναι ότι ο βαθμός πληρότητας ενός συγκεκριμένου στοιχείου προς εισαγωγή υποδεικνύει και το μέγεθος του στοιχείου. Όσο πιο μεγάλος είναι ο βαθμός πληρότητας, τόσο περισσότερα είναι και τα μη-μηδενικά στοιχεία που επιτρέπεται να εισαχθούν και συνεπώς τόσο πιο ακριβής είναι η προσέγγιση. Πιο συγκεκριμένα, στην ILU(k) ή πιο απλά ILUK, όλα τα στοιχεία με βαθμό πληρότητας που δεν υπερβαίνει το k διατηρούνται, ενώ τα υπόλοιπα απορρίπτονται. Δηλαδή έχουμε:

$$NZP \supset NZ(A) = \{(i,j) \mid lev_{ij} \leq \mathrm{k},\ 1 \leq \mathrm{i,j} \leq \mathrm{n}\}$$

όπου το lev_{ij} είναι η τιμή του βαθμού πληρότητας του στοιχείου στην θέση (i,j) μετά την ενημέρωση των στοιχείων που γίνεται σύμφωνα με τις ακόλουθες σχέσεις:

1. Ο αρχικός βαθμός πληρότητας ενός στοιχείου a_{ij} του αρχικού αραιού πίνακα A ορίζεται ως:
$$lev_{ij} = \begin{cases} 0 & \text{εάν } a_{ij} \neq 0 \text{ ή } i = j \\ \infty & \text{αλλιώς} \end{cases} \quad (3.1)$$

2. Κάθε φορά που το στοιχείο a_{ij} τροποποιείται, το lev_{ij} ενημερώνεται σύμφωνα με την σχέση:
$$lev_{ij} = min\{lev_{ij},\ lev_{ik} + lev_{ij} + 1\} \quad (3.2)$$

Παρατηρούμε ότι ο βαθμός πληρότητας ενός στοιχείου δεν είναι δυνατόν να αυξηθεί. Έτσι αν $a_{ij} \neq 0$ στον αρχικό πίνακα A, τότε το στοιχείο στη θέση (i,j) θα έχει $lev_{ij} = 0$ καθ' όλη την διαδικασία της παραγοντοποίησης. Θεωρώντας το παραπάνω NZP, ο αλγόριθμος 1.1 γίνεται:

1. $For\ all\ nonzero\ elements\ a_{ij}\ define\ lev_{ij} = 0$
2. $For\ i = 1, \dots, n\ Do$:
3. $\quad For\ each\ k = 0, \dots, i-1\ and\ for\ lev_{ij} \leq k$
4. $\quad\quad Compute\ a_{ik} = a_{ik} / a_{kk}$
5. $\quad\quad Compute\ a_{i*} = a_{i*} - a_{ik}\, a_{k*}$
6. $\quad\quad Update\ the\ level\ of\ fill\ of\ the\ nonzero\ a_{ij}{}'s\ using$ (3.2)
7. $\quad End\ Do$
8. $\quad Replace\ any\ element\ in\ row\ i\ with\ lev(a_{ij}) > k\ by\ zero$
9. $End\ Do$

Αλγόριθμος 5: ILUK Παραγοντοποίηση

όπου το a_{i*} υποδηλώνει την i -οστή γραμμή του A.

Παρατηρούμε πως για βαθμό πληρότητας $k = 0$ καταλήγουμε στην ILU(0) (Αλγόριθμος 4) όπου και όλα τα νέα μη-μηδενικά στοιχεία που προκύπτουν κατά την παραγοντοποίηση, απορρίπτονται. Ενώ για $k = 1$, απορρίπτονται όλα τα μη-μηδενικά στοιχεία που παράγονται μετά από το πρώτο επίπεδο. Αυτό σημαίνει πως ουσιαστικά η ILU(1) παραγοντοποίηση προκύπτει λαμβάνοντας ως NZP, το NZP του γινομένου LU των παραγόντων L, U που παράχθηκαν από την ILU(0) παραγοντοποίηση. Αναλόγως λαμβάνεται και το NZP για μεγαλύτερους βαθμούς πληρότητας.

Ο παραπάνω αλγόριθμος έχει όμως κάποια μειονεκτήματα. Πρώτον, το πλήθος τον μη-μηδενικών στοιχείων που εισάγονται και το υπολογιστικό κόστος για την παραγοντοποίηση δεν είναι γνωστό εκ των προτέρων για $k > 0$. Δεύτερον, το κόστος της ενημέρωσης των στοιχείων μπορεί να είναι πολύ υψηλό. Επίσης, πολύ σημαντικό είναι και το γεγονός πως ο βαθμός πληρότητας για μη θετικά ή αρνητικά ορισμένους πίνακες μπορεί τελικά να μην είναι καλός δείκτης του μεγέθους του στοιχείου που απορρίπτεται, κάτι που όμως ήταν και η ιδέα πίσω από την οποία βασίστηκε ο αλγόριθμος ILU(k). Σε μία τέτοια περίπτωση ο αλγόριθμος μπορεί να απορρίψει μεγάλα στοιχεία και έτσι να καταλήξει σε μία μη ακριβή παραγοντοποίηση, δηλαδή με μεγάλο σφάλμα. Για τον λόγο αυτό, υπάρχουν κάποιες εναλλακτικές υλοποιήσεις που δεν βασίζονται στην δομή του A για την επιλογή των απορριφθέντων στοιχείων, αλλά λαμβάνουν υπόψη το μέγεθος αυτών.

3.1.2. Ατελής LU Παραγοντοποίηση με Κατώφλι (ILUT)

Η ILUT [10] είναι μία τεχνική, που λαμβάνει υπόψη της το μέγεθος των στοιχείων κατά την διαδικασία της απόρριψης και επιπρόσθετα το NZP καθορίζεται δυναμικά και όχι εκ των προτέρων, όπως στις υλοποιήσεις που περιγράψαμε πρωτίστως, είναι η γνωστή ως ILUT. Η πιο γνωστή προσέγγιση αυτής είναι η ILUT(τ, p) στην οποία οι κανόνες απόρριψης των στοιχείων βασίζονται σε δύο παραμέτρους: την ανοχή απόρριψη (drop tolerance) τ και το όριο πληρότητας(fill level) p. Η παραγοντοποίηση βασίζεται στους εξής κανόνες:

1. Κατά την ενημέρωση μιας συγκεκριμένης γραμμής, έστω i, ένα στοιχείο a_{ik} απορρίπτεται εάν $|a_{ik}| < \tau_i$, όπου το τ_i είναι η σχετική ανοχή και υπολογίζεται ως το γινόμενο του τ και της νόρμας της i –οστής αυτής γραμμής. Δηλαδή έχουμε:
$$\tau_i = \tau * \| a_{i*} \| \qquad (3.3)$$

2. Μετά την ενημέρωση μιας γραμμής, εφαρμόζεται ο κανόνας 1 για ακόμη μία φορά ώστε να απορριφθούν όλα τα ενημερωμένα στοιχεία που είναι μικρότερα από το τ_i. Στην συνέχεια, διατηρούνται μόνο τα p μεγαλύτερα στοιχεία στο L και U τμήμα κάθε γραμμής.

Ο κανόνας 2 χρησιμοποιείται για τον έλεγχο του πλήθους των στοιχείων που διατηρούνται σε κάθε γραμμή. Ουσιαστικά, το p χρησιμοποιείται για τον περιορισμό των απαιτήσεων σε μνήμη και το τ για τον περιορισμό του υπολογιστικού κόστους.

Με βάση τους παραπάνω κανόνες, ο αλγόριθμος για την ILUT είναι:

1. $w = 0$
2. $For\ i = 1, \dots, n\ Do$:
3. $w = a_{i*}$
4. $For\ k = 0, \dots, i-1\ and\ where\ w_k \neq 0\ Do$:
5. $w_k = w_k / a_{kk}$
6. $Apply\ rule\ 1\ to\ w_k$
7. $If\ w_k \neq 0\ then$
8. $w = w - w_k * u_{k*}$
9. $End\ If$
10. $End\ Do$
11. $Apply\ rule\ 2\ to\ w$
12. $l_{ij} = w_j\ for\ j = 1, \dots, i-1$
13. $u_{ij} = w_j\ for\ j = i, \dots, n$
14. $w = 0$
15. $End\ Do$

Αλγόριθμος 6: ILUT(τ,p) Παραγοντοποίηση

όπου το w είναι η εκάστοτε γραμμή η οποία χρησιμοποιείται κατά την απαλοιφή και w_k είναι το k –οστό στοιχείο αυτής της γραμμής.

Επισημαίνουμε πως πιθανώς διαφορετικοί κανόνες θα μπορούσαν να έχουν χρησιμοποιηθεί στις γραμμές 6 και 11, Αλγόριθμος 6. Για παράδειγμα, αν χρησιμοποιήσουμε τον κανόνα απόρριψης στοιχείων που είναι σε θέσεις που ήταν άδειες στην δομή του αρχικού πίνακα A, τότε θα έχουμε την ILU(0) (Αλγόριθμος 4). Μία γνωστή παραλλαγή του παραπάνω αλγόριθμου είναι η διατήρηση των $nzu(i) + p$ και $nzl(i) + p$ μεγαλύτερων στοιχείων στο L και U τμήμα της i –οστής γραμμής αντίστοιχα, όπου τα $nzu(i)$ και $nzl(i)$ είναι το πλήθος των μη μηδενικών στοιχείων του τμήματος L και αντίστοιχα U της i –οστής γραμμής του πίνακα A. Αυτή η παραλλαγή θα είναι και αυτή που θα χρησιμοποιήσουμε στα παραδείγματα μας αργότερα.

3.1.2.1. ILUT με Οδήγηση Στηλών (ILUTP)

Υπάρχουν περιπτώσεις όπου η ILUT αποτυγχάνει. Η πιο συνηθισμένη περίπτωση είναι όταν ο αρχικός πίνακας A έχει κάποια μηδενικά διαγώνια στοιχεία, δηλαδή μη αποδεκτά για οδηγά στοιχεία. Μία προφανής λύση είναι να δώσουμε μία τυχαία τιμή σε κάθε μηδενικό στοιχείο της διαγωνίου, όμως αυτό θα είχε ως αποτέλεσμα να χάσουμε την επιθυμητή ακρίβεια του προκύπτοντος προρρυθμιστή. Η ιδανική λύση είναι να χρησιμοποιήσουμε οδήγηση στηλών στον αλγόριθμο της ILUT. Επισημαίνουμε πως και η οδήγηση γραμμών θα έλυνε το πρόβλημα, αλλά κάτι τέτοιο δεν θα ήταν πρακτικό αφού θα έπρεπε να σκανάρουμε όλη την γραμμή, αλλά και να εναλλάσσουμε γραμμές δηλαδή να τροποποιούμε την δομή των δεδομένων των γραμμών. Με την εφαρμογή της οδήγησης στηλών όμως, δεν έχουμε αυτό το πρόβλημα και πρέπει μόνο να καταγράφουμε την νέα σειρά των αγνώστων όσο αυτοί μετατίθενται. Επιπλέον, κάποιες επιπρόσθετες επιλογές χρησιμοποιούνται για την βελτίωση του αλγορίθμου της ILUTP:

- Μία επιπλέον παράμετρος ανοχής, γνωστή ως $permtol$ χρησιμοποιείται ώστε να καθοριστεί αν θα γίνει μετάθεση δύο στηλών. Έτσι, στο βήμα i, οι στήλες i και j μετατίθενται, μόνο όταν $permtol * |a_{ij}| > |a_{ii}|$.

- Η οδήγηση στηλών περιορίζεται μόνο σε διαγώνια κομμάτια συγκεκριμένου μεγέθους το οποίο καθορίζεται από την παράμετρο $mband$. Αν το $mband$ έχει τιμή μεγαλύτερη ή ίση του μεγέθους του αρχικού πίνακα A, τότε ουσιαστικά δεν υπάρχει κανένας περιορισμός.
- Εφαρμόζεται αλλαγή κλίμακας(scaling) σε όλες τις γραμμές ώστε το άθροισμα των απόλυτων τιμών των στοιχείων κάθε γραμμής, δηλαδή η 1-νόρμα κάθε γραμμής, να είναι ίση με 1. Ακολούθως, εφαρμόζεται αλλαγή κλίμακας και σε όλες τις στήλες.

Γενικά, η ILUTP είναι πιο εύρωστη από την ILUT, χωρίς όμως αυτό να σημαίνει ότι δεν υπάρχουν και περιπτώσεις που ILUTP αποτυγχάνει. Για παράδειγμα, η εφαρμογή οδήγησης στηλών δεν μπορεί να βοηθήσει σε περιπτώσεις όπου σε κάποιο βήμα της παραγοντοποίησης προκύψει μια ολόκληρη μηδενική γραμμή, έστω i. Βέβαια, είναι προφανές ότι ούτε η επιπρόσθετη εφαρμογή οδήγησης γραμμών θα μπορούσε να βοηθήσει αν στο ίδιο παράδειγμα προέκυπτε μηδενική και ολόκληρη η αντίστοιχη στήλη i. Περισσότερες λεπτομέρειες για την περίπτωση αυτή αναλύονται στην αναφορά [11]. Επιπλέον, συχνό φαινόμενο είναι η ILUTP να έχει υψηλότερες απαιτήσεις σε μνήμη απ' ότι η ILUT, διότι η ILUTP συχνά παράγει περισσότερα στοιχεία προς εισαγωγή.

3.1.3. Αλγεβρικός Αναδρομικός Πολυεπίπεδος Επιλυτής (ARMS)

Ο ARMS είναι ένας προρρυθμιστής που βασίζεται σε τεχνικές ILU παραγοντοποίησης καθώς και σε πολυεπίπεδες τεχνικές, ο οποίος υλοποιεί μία αναδρομική λύση για την κατασκευή του προρρρυθμιστή. Το βασικό στοιχείο για την κατασκευή του προρρυθμιστή είναι ο διαχωρισμός των στοιχείων σε δύο υποσύνολα: το "τραχύ"(coarse) υποσύνολο το οποίο προκύπτει χρησιμοποιώντας "τμηματικά" ανεξάρτητα σύνολα και το "λεπτό"(fine) υποσύνολο το οποίο περιέχει τα υπόλοιπα στοιχεία. Ως ανεξάρτητα σύνολα εννοούμε δύο σύνολα τα οποία δεν είναι συζευγμένα μεταξύ τους. Η ιδέα που κρύβεται πίσω από τον ARMS είναι να σχηματίσουμε το Schur συμπλήρωμα [12] που σχετίζεται με το τραχύ υποσύνολο το οποίο οδηγεί σε μία τμηματική(block) LU παραγοντοποίηση που μπορεί να χρησιμοποιηθεί ως προρρυθμιστής. Η διαδικασία συνεχίζεται αναδρομικά με το Schur συμπλήρωμα που σχετίζεται με το τραχύ υποσύνολο. Συνεπώς, βασικό στοιχείο στον ARMS αποτελεί η εύρεση του τραχύ υποσυνόλου ή αλλιώς ενός ανεξάρτητου συνόλου. Τότε, οι άγνωστοι αναδιατάσσονται έτσι ώστε πρώτοι να είναι αυτοί οι οποίοι σχετίζονται με το ανεξάρτητο σύνολο, ακολουθούμενοι από του υπόλοιπους αγνώστους και έτσι μπορούμε να ελαττώσουμε το σύστημα απαλείφοντας τους πρώτους. Πιο συγκεκριμένα στο εκάστοτε επίπεδο, έστω lev, της αναδρομής, ο ARMS αναδιατάσσει τον πίνακα A στην μορφή:

$$P_{lev}\, A_{lev}\, P_{lev}^{T} = \begin{pmatrix} B_{lev} & F_{lev} \\ E_{lev} & C_{lev} \end{pmatrix} \tag{3.4}$$

όπου P_{lev} είναι ο πίνακας μεταθέσεων και B_{lev} είναι ένας τμηματικός διαγώνιος πίνακας που αντιστοιχεί σε ένα ανεξάρτητο σύνολο. Παρατηρείστε ότι οι μεταθέσεις που εφαρμόζονται στο πίνακα A_{lev} είναι συμμετρικές. Η σχέση (3.4), προσεγγίζεται από την σχέση:

$$\begin{pmatrix} B_{lev} & F_{lev} \\ E_{lev} & C_{lev} \end{pmatrix} \approx \begin{pmatrix} L_{lev} & 0 \\ E_{lev}U_{lev}^{-1} & I \end{pmatrix} \begin{pmatrix} U_{lev} & L_{lev}^{-1}F_{lev} \\ 0 & A_{lev+1} \end{pmatrix} \tag{3.5}$$

όπου ο I είναι ο μοναδιαίος πίνακας, οι L_{lev} και U_{lev} είναι οι ILU παράγοντες του B_{lev} και ο A_{lev+1} είναι μία προσέγγιση για το Schur συμπλήρωμα σε σχέση με το C_{lev}:

$$A_{lev+1} \approx C_{lev} - E_{lev} B_{lev}^{-1} F_{lev} \quad (3.6)$$

Τυπικά δεν είναι ακριβό να επιλυθούν τα γραμμικά συστήματα με τους L_{lev} και U_{lev} αφού αυτά προκύπτουν από τεχνικές ILU παραγοντοποίησης. Συνεπώς το μόνο που χρειαζόμαστε για να βρούμε έναν προρρυθμιστή για τον A_{lev} είναι να επιλύσουμε το ελαττωμένο σύστημα, δηλαδή αυτό που σχετίζεται με το A_{lev+1}. Ο αλγόριθμος της ARMS παραγοντοποίησης είναι ο εξής:

1. $If\ lev == last\ lev\ then$:
2. $\quad Compute\ A_{lev} = L_{lev}\ U_{lev}$ $[with\ ILUT\ or\ ILUTP\ factorization\ of\ A_{lev}]$
3. $Else$
4. $\quad Find\ and\ independent\ set\ permutation\ P_{lev}$
5. $\quad Apply\ permutation\ A = P_{lev}\ A_{lev}\ P_{lev}^{T}$
6. $\quad Compute\ block\ factorization$ (σχέση 3.4)
7. $\quad Call\ ARMS(A_{lev+1})$
8. $End\ If$

Αλγόριθμος 7: ARMS Παραγοντοποίηση

όπου το 2ο βήμα υπολογίζεται συνήθως με χρήση της ILUT ή της ILUTP παραγοντοποίησης, χωρίς όμως αυτό να αποκλείει και την χρήση κάποιας διαφορετικής προσέγγισης. Για τον υπολογισμό της τμηματικής παραγοντοποίησης στο 6ο βήμα με σκοπό της εύρεση της σχέσης (3.5) ώστε να προκύψει ο πίνακας A_{lev+1}, χρησιμοποιείται μερική παραγοντοποίηση ILU, γνωστή ως PILU [13]. Τέλος, για την εύρεση του πίνακα μεταθέσεων στο 4ο βήμα, μπορούν να χρησιμοποιηθούν ποικίλες τεχνικές διάταξης. Στην υλοποίηση του ARMS που θα χρησιμοποιήσουμε, εφαρμόζεται μία απλή ευρετική τεχνική η οποία κατασκευάζει τμήματα με χρήση μίας επιπεδικής προσέγγισης σε συνδυασμό με μία ευρετική τεχνική για την απόρριψη στοιχείων από το τραχύ σύνολο. Οι λεπτομέρειες της υλοποίησης φαίνονται στον ακόλουθο αλγόριθμο:

1. $marker(1:n) = 0$
2. $For\ k = nod := 1, \ldots, n\ and\ if\ marker(nod) == 0\ Do$:
3. $\quad jcount = 0$
4. $\quad If\ (w(nod) < tolind\ )\ then\ AddToF(nod)$
5. $\quad Else$
6. $\quad\quad AddToC(nod);\ LevelSet = \{nod\}$
7. $\quad\quad While\ (jcount < bsize)\ Do$:
8. $\quad\quad\quad For\ each\ j\ in\ current\ LevelSet\ Do$:
9. $\quad\quad\quad\quad If(w(nod) < tolind)\ then\ AddToF(nod)$
10. $\quad\quad\quad\quad Else: AddToC(nod);\ jcount++;\ End\ If$
11. $\quad\quad\quad End\ For$
12. $\quad\quad End\ While$
13. $\quad\quad For\ each\ j\ in\ LevelSet\ Do$:
14. $\quad\quad\quad For\ each\ k\ in\ adj(j)\ Do$:
15. $\quad\quad\quad\quad If\ marker(k) == 0\ AddToF(k); End\ If$
16. $\quad\quad\quad EndFor$
17. $\quad\quad EndFor$
18. $\quad EndIf$
19. $EndFor$

Αλγόριθμος 8: Εύρεση του ανεξάρτητου συνόλου με ταξινόμηση με βάρη

όπου το w είναι ένα διάνυσμα με βάρη το οποίο χρησιμοποιείται για να καθοριστεί πότε ένας κόμβος θα ανατεθεί στο C σύνολο και υπολογίζεται ως εξής:

$$w(i) = \frac{\widehat{w}(i)}{max_{j=1,\dots,n}\ \widehat{w}(i)} \tag{3.7}$$

όπου

$$\widehat{w}(i) = \frac{|a_{ii}|}{\sum_{j=1}^{n}|a_{ij}|}\ ,\ \ 0 \leq\ \widehat{w}(i)\ \leq 1 \tag{3.8}$$

Επίσης, χρησιμοποιείται και η παράμετρος ανοχής $tolind$ με την οποία τελικά καθορίζεται ποιες γραμμές θα ανατεθούν στο C σύνολο. Συγκεκριμένα, μία γραμμή απορρίπτεται εάν το $w(i)$ είναι κάτω από το $tolind$. Τέλος, χρησιμοποιείται και η παράμετρος $bsize$ η οποία καθορίζει το μικρότερο επιτρεπτό μέγεθος για το Schur τμήμα. Έτσι, ο ARMS θα τερματίσει είτε όταν φτάσει στο τελευταίο επίπεδο, είτε όταν το μέγεθος του C συνόλου του επόμενου επιπέδου γίνει μικρότερο της δοθείσης παραμέτρου $bsize$.

Αξίζει να σημειώσουμε πως για προκαθορισμένο πλήθος επιπέδων ίσο με μηδέν ή για $bsize\ \geq\ |A|$, τότε το $last_{lev} = 0$ και ο ARMS καταλήγει να εφαρμόζει ILUT ή ILUTP παραγοντοποίηση, ανάλογα με το ποια τεχνική προσέγγισης χρησιμοποιείται στην 2η γραμμή του ARMS (Αλγόριθμος 7).

Για περισσότερες λεπτομέρειες για τον ARMS αλλά και γενικότερα για τις πολυεπίπεδες τεχνικές επίλυσης ανατρέξτε στις αναφορές [13, 14, 15, 16, 17, 18, 19].

3.1.3.1. ARMS με μη-συμμετρικές μεταθέσεις (ARMS-ddPQ)

Με σκοπό την καλή ποιότητα της παραγοντοποίησης, αναπτύχθηκε μία παραλλαγή του βασικού ARMS, ο οποίος υποστηρίζει μη-συμμετρικές μεταθέσεις. Έτσι, ο πίνακας μεταθέσεων P εφαρμόζεται στις γραμμές του A, ενώ σε αντιδιαστολή με τον ARMS, ένας άλλος πίνακας μεταθέσεων Q εφαρμόζεται στις στήλες του A. Οπότε τώρα, στο εκάστοτε επίπεδο, έστω lev, της αναδρομής, ο ARMS- ddPQ αναδιατάσσει τον πίνακα A ως εξής:

$$P_{lev}\, A_{lev}\, Q_{lev}^{T} = \begin{pmatrix} B_{lev} & F_{lev} \\ E_{lev} & C_{lev} \end{pmatrix} \tag{3.9}$$

Επισημαίνουμε επίσης πως, σε αντιδιαστολή με τον ARMS, δεν υπάρχει συγκεκριμένη απαίτηση για την δομή του πίνακα B_{lev}. Σε γενικές γραμμές οι πίνακες P_{lev} και Q_{lev} επιλέγονται με τέτοιο τρόπο ώστε ο B_{lev} να περιέχει τις γραμμές με την μεγαλύτερη δυνατη διαγώνια κυριαρχία και λίγα μη-μηδενικά στοιχεία, έτσι ώστε να περιοριστούν οι εισαγωγές νέων στοιχείων. Αντιστοίχως τώρα κατά το 6ο βήμα του ARMS (Αλγόριθμος 7) θα έχουμε:

$$\begin{pmatrix} B_{lev} & F_{lev} \\ E_{lev} & C_{lev} \end{pmatrix} \approx \begin{pmatrix} L_{lev} & 0 \\ E_{lev}U_{lev}^{-1} & I \end{pmatrix} \begin{pmatrix} U_{lev} & L_{lev}^{-1}F_{lev} \\ 0 & A_{lev+1} \end{pmatrix} \tag{3.10}$$

Αναφέρουμε ότι οι πίνακες P και Q σε κάθε επίπεδο, προκύπτουν σε τρία βήματα: το βήμα προεπιλογής, το βήμα ταιριάσματος και τέλος το βήμα ολοκλήρωσης ταιριάσματος. Για περισσότερες λεπτομέρειες ανατρέξτε στην αναφορά [13].

2ο μέρος:
Βιβλιοθήκες και Υλοποιήσεις

Κεφάλαιο 4

Χρήση Έτοιμων Βιβλιοθηκών

4.1. Βιβλιοθήκη CXsparse

Το CXSparce πακέτο περιέχει συναρτήσεις διαχείρισης αραιών μιγαδικών πινάκων και συστημάτων υλοποιημένες σε γλώσσα C. Πιο συγκεκριμένα, το CXSparce είναι μια επεκταμένη έκδοση του αρχικού πακέτου CSparse που αναπτύχθηκε για τις ανάγκες του βιβλίου "*Direct Methods for Sparse Linear Systems"* του *Timothy Davis* [4] ώστε να υποστηρίζει την διαχείριση αραιών μιγαδικών πινάκων.

Υπενθυμίζουμε πως όταν αναφερόμαστε σε έναν αραιό πίνακα εννοούμε από θεωρητικής άποψης ένα πίνακα με τάξης $O(n)$ μη μηδενικά στοιχεία και από υπολογιστικής άποψης έναν πίνακα ο οποίος έχει τέτοια αναπαράσταση ώστε να αποθηκεύονται και να συμμετέχουν σε υπολογισμούς μόνο τα μη μηδενικά στοιχεία. Για την αποθήκευση ενός αραιού μιγαδικού πίνακα η CXSparce χρησιμοποιεί δύο είδη δομών αποθήκευσης:

- Μορφή Τριστοιχίας (Triplet): Αποτελείται από δύο διανύσματα τα οποία περιέχουν τους αριθμοδείκτες των γραμμών και των στηλών αντίστοιχα των μη μηδενικών στοιχείων του πίνακα και από ένα διάνυσμα που περιέχει τις τιμές των μη μηδενικών στοιχείων. Είναι φανερό πως και τα τρία διανύσματα έχουν μέγεθος όσο τα συνολικά μη μηδενικά στοιχεία του πίνακα, έστω nz. Η μορφή Τριστοιχίας είναι κατάλληλη για αρχική εισαγωγή των μη μηδενικών στοιχείων του αραιού πίνακα αλλά δεν είναι κατάλληλη για πράξεις.

- Μορφή Συμπιεσμένων Στηλών (Compressed Column, CSC): Αποτελείται από ένα διάνυσμα x με τις τιμές των μη μηδενικών στοιχείων, ένα διάνυσμα i με τους αριθμοδείκτες των στηλών των στοιχείων που υπάρχουν στο διάνυσμα x και ένα διάνυσμα p με τις τοποθεσίες όπου ξεκινάει μία γραμμή στο διάνυσμα x. Το διάνυσμα p είναι μεγέθους $n + 1$, ενώ τα άλλα δύο διανύσματα είναι μεγέθους nz, όπως και στην μορφή Τριστοιχίας. Η μορφή Συμπιεσμένων Στηλών, σε αντίθεση με την μορφή Τριστοιχίας είναι κατάλληλη για εκτέλεση πράξεων μεταξύ αραιών πινάκων και διανυσμάτων.

Η δομή δεδομένων που χρησιμοποιεί το πακέτο CXSparce για την αποθήκευση των αραιών πινάκων και με τις δύο μορφές που μόλις περιγράψαμε είναι η εξής:

```
typedef struct cs_ci_sparse     /* matrix in compressed-column or triplet form */
{
  int nzmax ;                   /* maximum number of entries */
  int m ;                       /* number of rows */
  int n ;                       /* number of columns */
  int *p ;                      /* column pointers (size n+1) or col indices (size nzmax) */
  int *i ;                      /* row indices, size nzmax */
  cs_complex_t *x ;             /* numerical values, size nzmax */
  int nz ;                      /* # of entries in triplet matrix, -1 for compressed-col */
} cs_ci ;
```

όπου ο τύπος δεδομένων cs_complex_t είναι ισοδύναμος του τύπου complex double ο οποίος υποστηρίζει την αποθήκευση μιγαδικών αριθμών με πραγματικό και φανταστικό μέρος ακρίβειας double.

Για παράδειγμα, έστω ο πίνακας:

$$A = \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix} \tag{4.1}$$

όπου $\alpha_{ij} \in \mathbb{C}$ και $\alpha_{20} = \alpha_{01} = \alpha_{13} = \alpha_{12} = \alpha_{32} = \alpha_{03} = \alpha_{13} = \alpha_{23} = 0$. Με χρήση της παραπάνω δομής δεδομένων για την αποθήκευση του πίνακα θα έχουμε $nzmax = 8$ και $n = m = 4$ και συγκεκριμένα για αποθήκευση του πίνακασε μορφή τριστοιχίας, θα έχουμε:

$p \rightarrow$	0	2	0	1	2	0	1	3
$i \rightarrow$	0	0	1	1	2	3	3	3
$x \rightarrow$	α_{00}	α_{02}	α_{10}	α_{11}	α_{22}	α_{30}	α_{31}	α_{33}

Επίσης, για την αποθήκευση του πίνακα σε μορφή συμπιεσμένων στηλών, θα έχουμε:

$p \rightarrow$	0	3	5	7	8			
$i \rightarrow$	0	1	3	1	3	0	2	3
$x \rightarrow$	α_{00}	α_{10}	α_{30}	α_{11}	α_{31}	α_{02}	α_{22}	α_{33}

Σε αυτό το σημείο καλό είναι να δείξουμε πως γίνεται εύκολα ο εκ των προτέρων υπολογισμός του πλήθους μη μηδενικών στοιχείων του A, nz, το οποίο δίνεται ως παράμετρος σε κάποιες συναρτήσεις του πακέτου CXSparce που θα χρησιμοποιήσουμε. Κάθε στοιχείο του κυκλώματος συνεισφέρει συγκεκριμένο πλήθος μη μηδενικά στοιχείων στον πίνακα A και συγκεκριμένα:

- Κάθε αντίσταση, πυκνωτής, πηνίο και πηγή τάσης που δεν συνδέεται στο κόμβο αναφοράς 0 συνεισφέρει 4 μη μηδενικά στοιχεία.
- Κάθε αντίσταση, πυκνωτής, πηνίο και πηγή τάσης που συνδέεται στο κόμβο αναφοράς 0 συνεισφέρει 1 μη μηδενικό στοιχείο.
- Κάθε πηγή ρεύματος δεν συνεισφέρει κανένα μη μηδενικό στοιχείο.

Παρακάτω περιγράφουμε τις συναρτήσεις του πακέτου αυτού, που θα χρησιμοποιήσουμε στις υλοποιήσεις μας:

- Η εκχώρηση μνήμης για έναν πίνακα σε μορφή Τριστοιχίας , ο οποίος πρέπει να έχει δηλωθεί ως cs_ci *A, γίνεται με χρήση της συνάρτησης cs_ci_spalloc η οποία ορίζεται ως εξής:

```
/*Function for allocating the appropriate memory space for a sparse matrix in triplet or
CRC format.
@param m = Number of rows.
@param n  = Number of columns.
@param nzmax = Number of maximum number of non-zero elements.
@param values = Flag.    (0: only pattern will be allocated, 1: both pattern and values).
@param triplet = Flag.    (0: matrix will be stored in the CRC format, 1: in triplet format).
@return Pointer. (Success: to the struct describing the matrix, Otherwise: NULL). */
cs_ci *cs_ci_spalloc (int m, int n, int nzmax, int values, int t) ;
```

- Η μετατροπή ενός αραιού πίνακα από μορφή Τριστοιχίας σε μορφή Συμπιεσμένων Στηλών γίνεται με την συνάρτηση cs_ci_compress η οποία ορίζεται ως εξής:

```
/*Function for converting a matrix from triplet to CRC format. The columns of new ma-
trix are not sorted and duplicate entries may be present.
@param T  = Sparse matrix in triplet format.
@return Pointer. (Success: to the struct describing the matrix, Otherwise: NULL).*/
cs_ci *cs_ci_compress (const cs_ci *T) ;
```

- Όταν ο πίνακας αποθηκεύεται σε μορφή Τριστοιχίας, διαφορετικά μη μηδενικά στοιχεία με ίδιους αριθμοδείκτες γραμμής και στήλης διατηρούνται ως έχουν. Αυτό μπορεί να συμβεί για παράδειγμα από συνεισφορές διαφορετικών κυκλωματικών στοιχείων στην ίδια θέση του πίνακα. Μετά την μετατροπή του πίνακα σε μορφή Συμπιεσμένων Στηλών αυτά τα στοιχεία επίσης διατηρούνται ως έχουν. Για την συγχώνευση αυτών χρησιμοποιούμε την συνάρτηση cs_ci_dupl η οποία ορίζεται ως εξής:

```
/*Function that removes and sums duplicate entries in a sparse matrix.
@param A = The sparse matrix.
@return Integer. (1: success, 0: failure).*/
int cs_ci_dupl(cs_ci *A);
```

- Η ελευθέρωση ενός πίνακα που έχει δηλωθεί ως cs_ci *A γίνεται με χρήση της συνάρτησης cs_ci_spfree η οποία ορίζεται ως εξής:

```
/*Function for deallocating the allocated memory space for a sparse matrix in the CRC format.
@param A = Pointer to the matrix.
@return NULL.*/
cs_ci *cs_ci_spfree (cs_ci *A);
```

- Ο υπολογισμός του ανάστροφου ενός πίνακα σε μορφή συμπιεσμένων στηλών γίνεται με χρήση της συνάρτησης cs_ci_transpose η οποία ορίζεται ως εξής:

```
/*Function that is used for computing the tranpose of a sparse matrix.
@param A      = The sparse matrix in CRC form.
@param values = Flag. (0: only pattern will be transposed, 1: both pattern and values).
@return Pointer. (Success: to the struct describing the transpose matrix, Else: NULL).*/
cs_ci *cs_ci_transpose(const cs_ci *A, int values);
```

Επίσης, περιγράφουμε συνοπτικά και συναρτήσεις του πακέτου αυτού που χρησιμοποιήσουμε ειδικά για την υλοποίηση της άμεσης LU μεθόδου επίλυσης:

- Η συμβολική αναδιάταξη και συμβολική παραγοντοποίηση πίνακα σε μορφή συμπιεσμένων στηλών γίνεται με χρήση της συνάρτησης cs_ci_sqr η οποία ορίζεται ως εξής:

```
/*Symbolic ordering and analysis for LU or QR.
@param order  = The ordering method that will be used (0:natural, 1:Chol, 2:LU, 3:QR).
@param A      = The sparse matrix in CRC form.
@param qr     = The factorization method that will be used (0: LU, 1: QR).
@ return  Pointer to cs_ci_symbolic struct describing symbolic LU factorization.*/
cs_cis *cs_ci_sqr (int order, const cs_ci *A, int qr);
```

- Η αριθμητική παραγοντοποίηση LU ενός πίνακα μορφή συμπιεσμένων στηλών γίνεται με χρήση της συνάρτησης cs_ci_lu η οποία ορίζεται ως εξής:

```
/* Numerical LU factorization.
@param A      = The sparse matrix in CRC form.
@param S      = The symbolic LU factorization of A.
@param tol    = tolerance in order to find pivot element.
@return Pointer to cs_numeric struct describing L,U factors & partial pivoting vector.*/
cs_cin *cs_lu (const cs_ci  *A, const cs_cis *S, double tol);
```

- Ο υπολογισμός της αναδιάταξης ενός διανύσματος γίνεται με την συνάρτηση cs_ci_ipvec η οποία ορίζεται ως εξής:

```
/*Function that computes the permutation x = P'b of a vector.
@param p      = The permutation vector.
@param b      = Input vector.
@param x      = Output vector.
@param n      = Vector length.
@return integer (1: success, 0: failure).*/
int cs_ci_ipvec(const int *p, cs_complex_t  *b, cs_complex_t *x, int n);
```

- Η επίλυση των τριγωνικών συστημάτων $L\ y = b$ και $U\ x = y$ με προς τα εμπρός και προς τα πίσω αντικατάσταση, αντίστοιχα γίνεται με τις συναρτήσεις cs_ci_lsolve και cs_ci_usolve, αντίστοιχα. Οι συναρτήσεις αυτές ορίζονται ως εξής:

```
/* Function for solving a sparse lower triangular system Lx = b.
@param L        = The lower triangular matrix.
@param x        = The right-hand side vector on input and the solution on output.
@return integer (1: success, 0: failure).*/
int cs_ci_lsolve(const cs_ci *L, cs_comple_t *x);
```

και

```
/* Function for solving a sparse upper triangular system  Ux=b.
@param U        = The upper triangular matrix.
@param x        = The right-hand side vector on input and the solution on output.
@return integer (1: success, 0: failure).*/
int cs_ci_usolve (const cs_ci *U, cs_complex_t *x);
```

Για την χρήση των συναρτήσεων της βιβλιοθήκης CXSparce από ένα τμήμα κώδικα του εργαλείου μας γίνεται:

- Συμπερίληψη (#include) των αρχείου κεφαλίδας (header files) “cs.h” και “UFconfig.h” της βιβλιοθήκης.
- Μεταγλώττιση και σύνδεση της βιβλιοθήκης με χρήση του αρχείου libcxsparse.a που έχουμε δημιουργήσει.

4.2. Βιβλιοθήκη ZITSOL

Το ZITSOL πακέτο παρέχει συναρτήσεις που κατασκευάζουν προρρυθμιστές για την επίλυση αραιών μιγαδικών συστημάτων και είναι υλοποιημένες σε γλώσσα C. Πιο συγκεκριμένα, το ZITSOL είναι μια επεκταμένη έκδοση του αρχικού πακέτου ITSOL που αναπτύχθηκε βασιζόμενη στις επιστημονικές δημοσιεύσεις του διακεκριμένου καθηγητή της επιστήμης των υπολογιστών στο Πανεπιστήμιο της Μινεσότα, *Youcef Saad,* πάνω στην επίλυση αραιών γραμμικών συστημάτων, ώστε να υποστηρίζει την διαχείριση αραιών μιγαδικών συστημάτων.

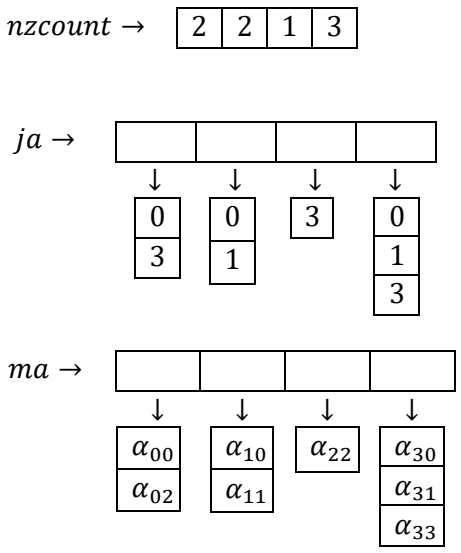
Για την αποθήκευση ενός αραιού μιγαδικού πίνακα η ZITSOL χρησιμοποιεί την παρακάτω δομή αποθήκευσης:

- Μορφή Συμπιεσμένων Γραμμών (Compressed Row, CSC): Αποτελείται από ένα διάνυσμα ma με τις τιμές των μη μηδενικών στοιχείων, ένα διάνυσμα ja με τους αριθμοδείκτες των στηλών και ένα διάνυσμα $nzcount$ με το πλήθος των μη μηδενικών στοιχείων κάθε γραμμής. Τονίζουμε ότι τα διανύσματα ma και ja αναπαρίστανται ως ένα διάνυσμα μεγέθους n με κάθε θέση του, έστω i, να είναι ένας δείκτης σε ένα διάνυσμα που περιέχει τα $nzcount[i]$ μη μηδενικά στοιχεία της i-οστής γραμμής του πίνακα και τους αριθμοδείκτες των στηλών που βρίσκονται τα $nzcount[i]$ μη μηδενικά στοιχεία της i-οστής γραμμής του πίνακα, αντίστοιχα.

Η δομή δεδομένων που χρησιμοποιεί το πακέτο ZITSOL για την αποθήκευση των αραιών πινάκων και με την μορφή που μόλις περιγράψαμε είναι η εξής:

```
typedef struct zSpaFmt *csptr;
typedef struct zSpaFmt {
  int n;
  int *nzcount;              /*  length of each row */
  complex double **ma;  /* pointer-to-pointer to store nonzero entries */
  int **ja;                  /* pointer-to-pointer to store column indices  */
} zSparMat;
```

Ο πίνακας (4.1) σε μορφή συμπιεσμένων στηλών με χρήση της παραπάνω δομής παριστάνεται ως εξής:



Παρακάτω περιγράφουμε τις συναρτήσεις του πακέτου αυτού, που θα χρησιμοποιήσουμε στις υλοποιήσεις μας για την κατασκευή και την επίλυση του Προρρυθμιστή στις επαναληπτικές μας μεθόδους:

- Η κατασκευή του Προρρυθμιστή με χρήση της Ατελούς LU παραγοντοποίησης με βαθμό πληρότητας ή πιο απλά του Προρρυθμιστή ILUK γίνεται με χρήση της συνάρτησης zilukC. Τονίζουμε πως πριν την κλήση της συνάρτησης αυτής θα πρέπει να έχουμε καθορίσει τον επιθυμητό βαθμό πληρότητας. Η συνάρτηση αυτή ορίζεται ως εξής:

```
/* ILUK preconditioner incomplete LU factorization with level of fill dropping
@param  lofM   = Level of fill: (all entries with level of fill > lofM are dropped).
@param csmat  = Matrix stored in SpaFmt format.
@param  lu       = Pointer to a ILUKSpar struct.
@param  fp       = file pointer for error log.
@return  ierr (0: success, -1: error in lofC, -2: zero diagonal found).
@return  lu->n   = dimension of the matrix,
            lu->L  = L part stored in SpaFmt format,
            lu->D  = Diagonals,
            lu->U  = U part stored in SpaFmt format.
Notes: All the diagonals of the input matrix must not be zero*/
int zilukC( int lofM, csptr csmat, iluptr lu, FILE *fp );
```

Επισημαίνουμε πως για την συγκριτική μελέτη που παρουσιάζεται στο Κεφάλαιο 7, έχουμε ορίσει κατά την κλίση της συνάρτησης αυτής ως επιθυμητό βαθμό πληρότητας $lofM = 5$.

- Η επίλυση του παραπάνω Προρρυθμιστή, δηλαδή του ILUK, γίνεται με χρήση της συνάρτησης zlusolC, η οποία ορίζεται ως εξής:

```
/*Performs a forward followed by a backward solve for LU matrix as produced by ILUK
@param  y        = Right-hand-side
@param  x        = Solution on return
@param  lu       = LU matrix as produced by iluk.
@return  the x parameter*/
int zlusolC( complex double *y, complex double *x, iluptr lu ) ;
```

- Η κατασκευή του Προρρυθμιστή με χρήση της Ατελούς LU παραγοντοποίησης με κατώφλι ή πιο απλά του Προρρυθμιστή ILUT γίνεται με χρήση της συνάρτησης zilutD. Τονίζουμε πως πριν την κλήση της συνάρτησης αυτής θα πρέπει να έχουμε καθορίσει τον επιθυμητό όριο πληρότητας στους προκύπτοντες παράγοντες L και U και την επιθυμητή ανοχή απόρριψης, αντίστοιχα. Η συνάρτηση αυτή ορίζεται ως εξής:

```
/* ILUT factorization with dual truncation.
@param amat   =  Matrix stored in SpaFmt struct.
@param ilusch  =  Pointer to ILUTfac struct.
@param lfil[5]   =  Number nonzeros in L-part.
            lfil[6]   = Number nonzeros in U-part.
@param droptol[5]        = Threshold for dropping small terms in L during factorization.
            droptol[6]        = Threshold for dropping small terms in U.
@return  (ilusch) =  Contains L and U factors in an LUfact struct.
@return  integer (0: successful, 1: the elimination  process  has generated a  row in
                        L or U whose length is >n, 2: memory allocation error,
                        5: illegal value for lfil or last, 6: zero row encountered).*/
int zilutD(csptr amat, double *droptol, int *lfil, ilutptr ilusch);
```

Επισημαίνουμε πως για την συγκριτική μελέτη που παρουσιάζεται στο Κεφάλαιο 7, έχουμε ορίσει κατά την κλίση της συνάρτησης αυτής ως όριο πληροτητας $lfill[5] = lfill[6] = 20$ και ως ανοχή απόρριψης $droptol[5] = droptol[6] = 0.01$.

- Η επίλυση του παραπάνω Προρρυθμιστή, δηλαδή του ILUT, γίνεται με χρήση των συναρτήσεων zLsol και zUsol. Οι συναρτήσεις πρέπει να καλούνται με την σειρά που την αναφέραμε. Η συνάρτηση zLsol ορίζεται ως εξής:

```
/*This function does the forward solve L x = b.
@param mata   = The matrix (in SpaFmt form).
@param b      = A vector.
@return x     = The solution of L x = b.*/
void zLsol(csptr mata, complex double *b, complex double *x);
```

Και η συνάρτηση zUsol ορίζεται ως εξής:

```
/*This function does the backward solve U x = b.
@param mata   =  The matrix (in SpaFmt form).
@param b      =  A vector.
@return x     = The solution of U x = b.*/
void zUsol(csptr mata, complex double *b, complex double *x)
```

- Η κατασκευή του Προρρυθμιστή με χρήση της ατελούς LU παραγοντοποίησης με κατώφλι και με οδήγηση στηλών ή πιο απλά του Προρρυθμιστή ILUTP γίνεται με χρήση της συνάρτησης zilutpC. Τονίζουμε πως πριν την κλήση της συνάρτησης αυτής θα πρέπει να έχουμε καθορίσει τον επιθυμητό όριο πληρότητας στους προκύπτοντες παράγοντες L και U και την επιθυμητή ανοχή απόρριψης, αντίστοιχα, όπως και στον ILUT Προρρυθμιστή. Θα πρέπει επίσης να καθορίσουμε και δύο επιπλέον παραμέτρους, μία ανοχής και μία που αφορά την οδήγηση στηλών. Η συνάρτηση αυτή ορίζεται ως εξής:

```
/* ILUTP -- ILUT with column pivoting.
@param amat   =  Matrix stored in SpaFmt struct.
@param lfil[5]  =  Number nonzeros in L-part
       lfil[6]  =  Number nonzeros in U-part
@param droptol[5] = Threshold for dropping small terms in L during factorization.
       droptol[6] = Threshold for dropping small terms in U.
@param permtol     = Tolerance ratio used to  determine whether or not to permute
                     two columns.  At step i columns i and j are permuted when
                          abs(a(i,j))*permtol > abs(a(i,i))
@param mband  =  Permuting is done within a band extending to mband diagonals only
                 (0: no pivoting, n: pivot is searched in whole column)
@param (ilusch) = Contains L and U factors in an LUfact struct.
@return  iperm   =  Reverse permutation array.
@return integer (0: success, 1: the elimination  process  has generated a  row in
                 L or U whose length is >n, 2: memory allocation error,
                 5: illegal value for lfil or last, 6: zero row encountered).*/
int zilutpC (csptr amat, double *droptol, int *lfil, double permtol, int mband, ilutptr lusch);
```

Επισημαίνουμε πως για την συγκριτική μελέτη που παρουσιάζεται στο Κεφάλαιο 7, έχουμε ορίσει κατά την κλίση της συνάρτησης αυτής ως όριο πληροτητας $lfill[5] = lfill[6] = 20$ και ως ανοχή απόρριψης $droptol[5] = droptol[6] = 0.01$, καθώς και επιπλέον παραμέτρους $permtol = 0.99$ και $mband = n$.

- Η επίλυση του παραπάνω Προρρυθμιστή, δηλαδή του ILUTP, γίνεται με χρήση των συναρτήσεων z SchLsol και zSchUsol. Οι συναρτήσεις πρέπει να καλούνται με την σειρά που την αναφέραμε. Η συνάρτηση zSchLsol ορίζεται ως εξής:

```
/* This function does the forward solve L x = y after applying scaling and permutations.
@param ilusch  = The LU matrix as provided from the ILU functions.
@param y       = The right-hand-side vector.
@return y      = Solution of L x = y. [overwritten].*/
void zSchLsol(ilutptr ilusch, complex double *y);
```

Η συνάρτηση zSchUsol ορίζεται ως εξής:

```
/* This function does the backward solve U x = y after applying column pivoting, scaling
and permutations.
@param ilusch = The LU matrix as provided from the ILU functions.
@param y      = The right-hand-side vector.
@return y     = Solution of U x = y. [overwritten on y]*/
void zSchUsol(ilutptr ilusch, complex double *y);
```

- Η κατασκευή του Αλγεβρικού Αναδρομικού Πολυεπίπεδου Προρρυθμιστή ή πιο απλά του Προρρυθμιστή ARMS γίνεται με χρήση της συνάρτησης zarms2. Τονίζουμε πως πριν την κλήση της συνάρτησης θα πρέπει να καθοριστεί ένα πλήθος παραμέτρων. Επιγραμματικά αναφέρουμε πως οι παράμετροι $ipar[0:3]$ αφορούν γενικές προτιμήσεις για την εκτέλεση του αλγορίθμου, όπως το μέγιστο πλήθος των επιπέδων και την εκτέλεση του απλού ή του επεκταμένου ARMS. Οι παράμετροι $ipar[10:13]$ αφορούν προτιμήσεις για τα ενδιάμεσα επίπεδα όπως για το αν θα εφαρμοστεί αλλαγή κλίμακας ή/και μεταθέσεις. Ομοίως και οι παράμετροι $ipar[14:17]$ αλλά για το τελευταίο επίπεδο του αλγορίθμου. Επίσης οι παράμετροι $droptol[0:4]$ και $droptol[5:6]$ είναι οι παράμετροι ανοχής για τα διάφορα τμήματα(blocks) των ενδιάμεσων και του τελευταίου επιπέδου αντίστοιχα. Ομοίως για τις παραμέτρους $lfil[0:4]$ και $lfil[5:6]$ που καθορίζουν τα όρια πληρότητας. Τέλος, πρέπει να καθοριστεί και μία ακόμη παράμετρος ανοχής, η $tolind$. Αναλυτικότερα, η συνάρτηση zarms2 ορίζεται ως εξής:

```
/*  Multi-level Block  ILUT or ILUTP Preconditioner.
@param amat = original matrix A stored in CRC format.
@param ipar[0:17] = Integer array to store parameters for arms construction (arms2)
      ipar[0]:= nlev.  Number of levels allowed.
      ipar[1]:= level-reordering option to be used. (0; ARMS, 1:ARMS-ddPQ).
      ipar[2]:= bsize. The size of  each block can vary and is >= bsize
                        (for ddPQ: this is only the smallest size of the  last level).
      ipar[3]:= iout   (1: statistics on the run are printed to FILE *ft).
      ipar[4-9] not used. Set to zero.
      ipar[10-13] == meth[0:3] = method flags for interlevel blocks.
      ipar[14-17] == meth[0:3] = method flags for last level block.
            if meth[0]==1 nonsymmetric permutations are used for last schur complement,
            if meth[1]==1 ILUTP is used instead of ILUT,
            if meth[2]==1 diagonal row scaling in both cases,
            if meth[3]==1 diagonal column scaling in both cases.
@param  ft  =  File for printing statistics on run
@param tolind   = Tolerance parameter used by the indset function.
@param droptol  = Threshold parameters for dropping elements in ILU factorization.
      droptol[0:4] = related to the multilevel  block factorization.
      droptol[5:6] = related to ILU factorization of last block.
      droptol[0] = threshold for dropping  in L [B].
      droptol[1] = threshold for dropping  in U [B].
      droptol[2] = threshold for dropping  in L^{-1} F.
      droptol[3] = threshold for dropping  in E U^{-1}.
      droptol[4] = threshold for dropping  in Schur complement.
      droptol[5] = threshold for dropping  in L in last block.
      droptol[6] = threshold for dropping  in U in last block.
@param lfil  = lfil[0:6] is an array containing the fill-in parameters.
      lfil[0] = amount of fill-in kept  in L [B].
      lfil[1] = amount of fill-in kept  in U [B].
      lfil[2] = amount of fill-in kept  in E L\inv
      lfil[3] = amount of fill-in kept  in U \inv F
      lfil[4] = amount of fill-in kept  in S    .
      lfil[5] = amount of fill-in kept  in L_S  .
      lfil[6] = amount of fill-in kept  in U_S
@return  (PreMat)  = Arms data structure which consists of two parts:  levmat and ilsch.
      ++(levmat)= Permuted and sorted matrices for each level of the block  factorization.
      ++(ilsch) = Contains the block C and an ILU  factorization (matrices L and U)
                  for the last Schur complement [S ~ C - E inv(B) F ].
@return ipar[0]   = Number of levels found (may differ from input value)*/
@return integer (0: success, 1: failure)
int zarms2(csptr Amat, int *ipar, double *droptol, int *lfil, double tolind, arms PreMat,
            FILE *ft);
```

Επισημαίνουμε πως για την συγκριτική μελέτη που παρουσιάζεται στο Κεφάλαιο 9, έχουμε ορίσει κατά την κλίση της συνάρτησης αυτής ως το πλήθος των επιπέδων $ipar[0] = 20$, το μέγεθος του τελευταίου Schur συμπληρώματος $ipar[2] = 600$ και έχουμε ενεργοποιήσει τις μη συμμετρικές μεταθέσεις(δηλαδή την εφαρμογή του ARMS-ddPQ), την αλλαγή κλίμακας στα ενδιάμεσα καθώς και στο τελευταίο επίπεδο και τέλος την αναδιάταξη και την χρήση του ILUTP στο τελευταίο επίπεδο. Για την ενεργοποίηση των προαναφερθέντων θέσαμε $ipar[1] = ipar[12] = ipar[14] = ipar[15] = ipar[16] = ipar[17] = 1.$ Οι υπόλοιπες συνιστώσες της παραμέτρου $ipar$ τέθηκαν ίσες με μηδέν. Επιπλέον, ορίσαμε όλες τις συνιστώσες του ορίου πληρότητας ίσες με $lfill = 20$ και την ανοχή απόρριψης ως $droptol[0] = droptol[1] = droptol[4] = 0.001$, $droptol[2] = droptol[3] = 0.0001$ και $droptol[5] = droptol[6] = 0.01$. Τέλος, η παράμετρος ανοχής ορίστηκε ως $tolind = 0.7$.

- Η επίλυση του παραπάνω Προρρυθμιστή, δηλαδή του ARMS, γίνεται με χρήση της συνάρτησης zarmsol2, η οποία ορίζεται ως εξής:

```
/* Combined preconditioning operation. Combines the left and right actions.
@param  x       =  Right-hand side to be operated on by the preconditioner.
@on return  x   =  Output result of operation [overwritten].
@return integer (0: success, 1: failure).*/
int zarmsol2(complex double *x,  arms Prec);
```

Από το πακέτο αυτό θα χρησιμοποιήσουμε και κάποιες ακόμη βοηθητικές συναρτήσεις για τις υλοποιήσεις μας. Παρακάτω περιγράφουμε συνοπτικά τις συναρτήσεις αυτές:

- Η εύρεση του αντίστροφου ενός πίνακα σε μορφή συμπιεσμένων γραμμών γίνεται με την συνάρτηση zSparTran η οποία ορίζεται ως εξής:

```
/* Finds the transpose of a matrix stored in SpaFmt format.
@param amat   = A matrix stored in SpaFmt format.
@param job    = Flag. (1: fill the values of the matrix (bmat), 0: fill only the pattern).
@param flag   = Flag. (1: the matrix has been filled, 0: matrix not filled).
@on return bmat = The transpose of amat stored in SpaFmt format.
@return integer (0: success, 1: memory allocation error).*/
int zSparTran(csptr amat, csptr bmat, int job, int flag);
```

- Η κλιμάκωση κάθε γραμμής και στήλης ενός πίνακα σε μορφή συμπιεσμένων γραμμών ώστε η νόρμα αυτής να είναι ίση με 1, γίνεται με χρήση της συνάρτησης zroscalC και zcoscalC, αντίστοιχα. Οι συναρτήσεις αυτές ορίζονται ως εξής:

```
/*This routine scales each row of mata so that the norm is 1.
@param mata   = The matrix stored in SpaFmt form.
@param nrm    = Type of norm (0 (\infty),  1 or 2).
@on return diag  = diag[j] = 1/norm(row[j]).
@return integer (0: normal return, j: row j is a zero row).*/
int zroscalC(csptr mata, double *diag, int nrm);
```

και

```
/*This routine scales each column of mata so that the norm is 1.
@param mata   = The matrix stored in SpaFmt form.
@param nrm    = Type of norm (0 (\infty),  1 or 2 ).
@ on return diag  = diag[j] = 1/norm(row[j])
@return integer (0: normal return, j: column j is a zero column*/
int zcoscalC(csptr mata, double *diag, int nrm);
```

- Η επιλογή των πινάκων P και Q για την αναδιάταξη των γραμμμών και των στηλών αντίστοιχα ενός πίνακα σε μορφή συμπιεσμένων γραμμών γίνεται με χρήση της συνάρτησης zPQper, η οποία ορίζεται ως εξής:

```
/* Algorithm for nonsymmetric block selection.
@param mat    =  Matrix in SpaFmt format.
@param tol    =  A tolerance for excluding a row from B block.
@param bsize  =  Not used here
@on return Pord  = Row permutation array.
@on return Qord  = Column permutation array.
@on return nnod  = Number of elements in the B-block.
int zPQperm(csptr mat, int bsize, int *Pord, int *Qord, int *nnod, double tol);
```

- Τέλος, η εφαρμογή των παραπάνω P και Q αναδιατάξεων στον αντίστοιχο πίνακα, γίνεται με χρήση των συναρτήσεων zrpermC και zcpermC, αντίστοιχα. Οι συναρτήσεις αυτές ορίζονται ως εξής:

```
/*This function permutes the rows of a matrix in SpaFmt format.
@param amat    = A matrix stored in SpaFmt format.
@param perm    =  The permutation P.
@on return amat = PA stored in SpaFmt format.
@return integer (0: successful return, 1: memory allocation error).*/
int zrpermC(csptr mat, int *perm);
```

και

```
/*This function permutes the columns of a matrix in SpaFmt format.
@param mat     = A matrix stored in SpaFmt format.
@param perm    = The permutation Q
@on return mat = AQ stored in SpaFmt format.
@return integer (0: successful return, 1: memory allocation error).*/
int zcpermC(csptr mat, int *perm);
```

Για την χρήση των συναρτήσεων της βιβλιοθήκης ZITSOL από ένα τμήμα κώδικα του εργαλείου μας γίνεται:

- Συμπερίληψη (#include) των αρχείου κεφαλίδας (header files) "zdefs.h", "zheads.h" και "zprotos.h" της βιβλιοθήκης.
- Μεταγλώττιση και σύνδεση της βιβλιοθήκης με χρήση του αρχείου libzitsol.a που έχουμε δημιουργήσει. Επισημαίνουμε πως για την μεταγλώττιση της βιβλιοθήκης είναι απαραίτητη η εγκατάσταση και η μεταγλώττιση του προγράμματος f2c, το οποίο μεταγλωττίζει πηγαίο κώδικα από γλώσσα fortran77 σε γλώσσα C. Αυτό γίνεται επειδή η βιβλιοθήκη ZITSOL χρησιμοποιεί κάποιες συναρτήσεις της fortran έκδοσης των βιβλιοθηκών BLAS και LAPACK, οι οποίες μπορούν να αναγνωριστούν από το μεταγλωττιστή gcc που χρησιμοποιούμε, μόνο μέσω της μετατροπής που κάνει το πρόγραμμα f2c.

Κεφάλαιο 5

Υλοποιήσεις

Παρακάτω θα περιγράψουμε λεπτομερώς την πρακτική υλοποίηση του εργαλείου προσομοίωσης κυκλωμάτων για AC ανάλυση που αναπτύξαμε. Πιο συγκεκριμένα, αρχικά θα δώσουμε μία περιγραφή για την απαιτούμενη μορφή του αρχείου εισόδου το οποίο θα περιέχει την περιγραφή του κυκλώματος. Στην συνέχεια, θα περιγράψουμε ξεχωριστά κάθε μία από τις λειτουργικότητες του εργαλείου μας, κάθε μία εκ των οποίων περιέχεται σε διαφορετικά αρχεία κώδικα, τα οποία είναι τα εξής:

Λειτουργικότητα	**Αρχεία Κώδικα**	
Συντακτικός αναλυτής	parser.h	parser.c
Απεικόνιση	mapping.h	mapping.c
Διαμόρφωση του MNA Συστήματος	acmthds.h	acmthds.c
Προρρυθμισμένες Επαναληπτικές Μέθοδοι Επίλυσης Μιγαδικών Αραιών Γραμμικών Συστημάτων	ac.h	ac.c
Άμεση LU Μέθοδος Επίλυσης Αραιών Μιγαδικών Γραμμικών Συστημάτων	acdirect.h	acdirect.c
Βοηθητικές Συναρτήσεις	aclinsys.h	aclisys.c
Κυρίως Πρόγραμμα		main.c

Πίνακας 1: Λειτουργικότητες Υλοποιήσεων και Αρχεία Κώδικα

5.1. Αρχείο Περιγραφής Κυκλώματος

Το κύκλωμα περιγράφεται με την χρήση ενός αρχείου, γνωστό ως netlist, το οποίο αποτελεί το αρχείο εισόδου(αρχείο εισαγωγής δεδομένων) του εργαλείου μας. Το αρχείο αυτό περιέχει τα στοιχεία του κυκλώματος (αντιστάσεις, πυκνωτές, πηνία και πηγές τάσεις και ρεύματος) και περιγράφει πως αυτά συνδέονται μεταξύ τους κάνοντας χρήση αλφαριθμητικών που αντιστοιχούν στους διάφορους κόμβους του κυκλώματος. Αυτό ο τρόπος περιγραφής, ονομάζεται συμβολισμός κόμβων(nodal notation) ο οποίος χρησιμοποιείται για την περιγραφή του αρχείου εισόδου στα περισσότερα γνωστά προγράμματα προσομοίωσης κυκλωμάτων (προγράμματα SPICE) [20,21]. Επίσης στο αρχείο εισόδου υπάρχουν πληροφορίες που “ενεργοποιούν” το είδος των αναλύσεων που πρέπει να γίνουν και με ποιες παραμέτρους καθώς και κάποιες επιπλέον επιλογές(options) για την προσομοίωση.

Αναλυτικότερα, κάθε στοιχείο του κυκλώματος περιγράφεται σε μία γραμμή του αρχείου, η οποία θα πρέπει να έχει την εξής μορφή:

Αντίσταση:	*R<name> <positive_node> <negative_node> <value>*
Πυκνωτής:	*C<name> <positive_node> <negative_node> <value>*
Πηνίο:	*L<name> <positive_node> <negative_node> <value>*

όπου τα *<name>*, *<positive_node>* και *<negative_node>* είναι αλφαριθμητικά, ενώ το *<value>* είναι αριθμός. Τονίζουμε πως, κατά σύμβαση, η γείωση πρέπει να δηλώνεται ως ο κόμβος 0.
Και για τις πηγές:

Πηγή Ρεύματος: *I<name> <DC_value> AC <magnitude> <phase> <trasient_spec>* Πηγή Τάσης: *V<name> <DC_value> AC <magnitude> <phase> <transient_spec>*

Επισημαίνουμε πως για την περίπτωση της AC ανάλυσης με την οποία ασχολούμαστε, χρησιμοποιούμε μόνο την AC τιμή κάθε πηγής η οποία δίνεται σε πολική μορφή με καθορισμό του μέτρου και της φάσης από τις τιμές *<magnitude>* και *<phase>* αντίστοιχα. Τονίζουμε ότι η τιμή της φάσης θα πρέπει να δίνεται σε μοίρες, κι όχι σε ακτίνια. Σημαντικό είναι επίσης να τονίσουμε πως όσες πηγές που περιγράφονται στο αρχείο εισόδου δεν έχουν προσδιορισμένη την AC τιμή του, θωρούνται μηδενικές για την ανάλυση εναλλασσόμενου ρεύματος. Προσοχή όμως, πρέπει να υπάρχει τουλάχιστον μία πηγή με μη μηδενική AC τιμή στο αρχείο εισόδου ώστε να μπορέσει να εφαρμοστεί ανάλυση εναλλασσόμενου ρεύματος.

Για την ενεργοποίηση της AC ανάλυσης, το αρχείο εισόδου θα πρέπει να περιέχει μία δήλωση της μορφής:

.AC <sweep_type> <np> <start _frequency_value> <end_frequency_value>

όπου το *<sweep_type>* πρέπει να είναι LIN (στο παρόν εργαλείο υποστηρίζεται μόνο γραμμική σάρωση της συχνότητας). Με την εντολή αυτή δηλώνεται πως για την ανάλυση εναλλασσόμενου ρεύματος επιθυμούμε να γίνει σάρωση της περιοχής συχνοτήτων μεταξύ *<start_frequency_value>* και *<end_frequency_value>* και συγκεκριμένα σε *<np>* πλήθος συχνοτήτων, ισαπέχουσων γραμμικά. Η παραπάνω δήλωση, πρέπει να ακολουθείτε από μία δήλωση που αφορά την εκτύπωση των αποτελεσμάτων. Αυτή, αφορά μόνο ένα υποσύνολο μεταβλητών και το οποίο καθορίζεται με μία εντολή της μορφής:

.PLOT V(<nodeX>) V(<nodeY>) … V(<nodeZ>)

όπου οι *<nodeX>, <nodeY>, … , <nodeZ>* είναι οι κόμβοι των οποίων η τάση θέλουμε να εκτυπωθεί στο αρχείο εξόδου. Επισημαίνουμε πως το εργαλείο μας παράγει αρχείο εξόδου με το όνομα "*AC_RESULTS.TR*". Εκεί εκτυπώνονται τα αποτελέσματα των κόμβων που καθορίζονται στην αντίστοιχη *.PLOT* δήλωση για κάθε *.AC* δήλωση του αρχείου εισόδου. Πιο συγκεκριμένα, τα ζητούμενα αποτελέσματα εκτυπώνονται συναρτήσει της κάθε κυκλικής συχνότητας *ω* που προκύπτει από τις συχνότητες σάρωσης της αντίστοιχης *.AC* δήλωσης και για την οποία εκτελείται ανάλυση εναλλασσόμενου ρεύματος. Κάθε αποτέλεσμα εκτυπώνεται σε πολική μορφή με καθορισμό του μέτρου και της φάσης.

Τέλος, στο αρχείου εισόδου θα πρέπει απαραίτητα να γίνεται η προσθήκη των επιλογών για χρήση αραιών πινάκων και χρήση συγκεκριμένης μεθόδου επίλυσης. Η προκαθορισμένη μέθοδος επίλυσης μιγαδικών αραιών γραμμικών συστημάτων είναι η άμεση LU μέθοδος. Οπότε, για την ενεργοποίηση αυτής δεν χρειάζεται κάποια επιπλέον επιλογή και αρκεί η επιλογή για χρήση αραιών πινάκων ως εξής:

. OPTIONS SPARSE

Τονίζουμε επίσης πως σε αυτήν την έκδοση του εργαλείου μας δεν υποστηρίζεται χρήση πυκνών πινάκων.

Για την ενεργοποίηση χρήσης της επεκταμένης μεθόδου συζυγών κλίσεων(Bi-CG) για την επίλυση μιγαδικών αραιών γραμμικών συστημάτων, χρησιμοποιείται επιπλέον η επιλογή ITER, ως εξής:

```
.OPTIONS SPARSE ITER
```

Για την ενεργοποίηση χρήσης της απλής μεθόδου συζυγών κλίσεων(CG) για την επίλυση μιγαδικών αραιών γραμμικών συστήματων, χρησιμοποιείται επιπλέον η επιλογή SPD ως εξής:

```
.OPTIONS SPARSE ITER SPD
```

μέσω της οποίας δηλώνεται ότι ο πίνακας συντελεστών είναι συμμετρικός και θετικά ορισμένος(symmetric and positive definite).

5.2. Συντακτικός Αναλυτής

Ο συντακτικός αναλυτής (Parser) διαβάζει το αρχείο εισόδου και βρίσκει πως πρέπει να συνδεθούν τα διάφορα κυκλωματικά στοιχεία σε κάθε κόμβο. Τονίζουμε πως χρησιμοποιούμε τον συντακτικό αναλυτή που υλοποιήθηκε στα πλαίσια του προπτυχιακού μαθήματος *"Προσομοίωση Κυκλωμάτων"* και ο οποίος επεκτάθηκε στα πλαίσια της παρούσας εργασίας ώστε για τις πηγές να υποστηρίζεται ο καθορισμός και AC τιμής και να αναγνωρίζεται κατάλληλα κάθε *.AC* δήλωση. Η διεπαφή και η υλοποίηση αυτού περιέχονται στα αρχεία κώδικα **parser.h** και **parser.c** αντίστοιχα.

Κάθε κυκλωματικό στοιχείο που διαβάζεται αποθηκεύεται σε μία δομή δεδομένων (struct) τύπου *element_t* και προστίθεται σε μία συνδεδεμένη λίστα list που περιέχει όλα τα στοιχεία, με χρήση της συνάρτησης *addElement*. Παρακάτω δίνεται αναλυτικά η δομή δεδομένων που χρησιμοποιείται, καθώς και το πρωτότυπο της συνάρτησης *addElement*:

```
// this struct represents a circuit element stored in a single node of the linked list
struct element_t{
  char *name;              // starts with one of [V,I,R,C,L,D,M,Q]
  short terminal_count;    // the number of terminals of the element
  char *terminal_a;        // the positive terminal
  char *terminal_b;        // the negative terminal
  char *terminal_c;        // used by BJT and MOS
  char *terminal_d;        // used by MOS
  double value;
  double L;                // used in MOS
  double W;                // used in MOS
  int mna_terminal_a;      // the positive terminal used by the Modified Nodal Analysis
                              (sequential node IDs)
  int mna_terminal_b;      // the negative terminal used by the Modified Nodal Analysis
                              (sequential node IDs)
  struct ac_source_t *ac_source;
  struct transient_source_t *transient_source;
  struct element_t *next;
  };
struct element_t *list;      // the linked list's head
struct element_t *addElement(char *name, short terminal_count, char *terminal_a,
                             char *terminal_b, char *terminal_c, char* terminal_d,
                             double value, double L, double W,
                             struct ac_source_t *ac_source,
```

```
struct transient_source_t *transient_source);
```

Είναι προφανές πως για τους σκοπούς της παρούσας εργασίας δεν χρησιμοποιούνται οι μεταβλητές εκείνες που αφορούν τρανζίστορς (MOS και BJT), καθώς και οι δομή δεδομένων transient_source_t. Επίσης η δομή δεδομένων ac_source_t η οποία χρησιμοποιείται για την αποθήκευση της AC τιμής μίας πηγής ορίζεται ως εξής:

```
// holds the info for an AC source
struct ac_source_t{
   double magnitude;
   double phase;
};
```

Επιπλέον, κάθε *.AC* δήλωση που διαβάζεται αποθηκεύεται σε μία δομή δεδομένων (struct) τύπου dotAC_t και προστίθεται στο πίνακα dotAC ο οποίος περιέχει όλα τα στοιχεία όλες τις *.AC* δηλώσεις. Η δομή αυτή ορίζεται ως εξής:

```
//the array of structs that holds the info for all the parsed .AC statements
struct dotAC_t{
   char sweep_type[MAX_STRING_SIZE];
   int npoints;
   double fstart;
   double fend;
    // an array to all the node names of the corresponding  .PLOT statement
   char plot_node_name[MAX_PLOT_PER_DC][MAX_STRING_SIZE];
   // the count of the node names in the corresponding .PLOT statement
   int plot_node_ctr;
}dotAC[MAX_AC_STATEMENTS];
```

Μερικές άλλες χρήσιμες, για την περίπτωση που εξετάζουμε, μεταβλητές που ορίζει ο συντακτικός αναλυτής είναι:

```
int dotACctr;              // the count of .AC statements
short dotACflag;           // turns true if a .AC statement exists
short optionSPD;           // is TRUE if there is an SPD option used
short optionITER;          // is TRUE if there is an ITER option used
short optionSPARSE;        // is TRUE if there is an SPARSE option used
int voltage_sources_count;         // the voltage sources count
int elements_count;                // the count of all the circuit's elements
```

Σημειώνουμε πως δεν παρουσιάζουμε τον υπόλοιπο κώδικα των αρχείων αυτών διότι δεν έχουν καμία πρακτική αξία για την κατανόηση των υλοποιήσεων των θεμάτων που αναλύουμε στην παρούσα εργασία.

5.3. Απεικόνιση

Κατά την απεικόνιση (mapping) γίνεται αντιστοίχηση των αλφαριθμητικών ονομάτων των κόμβων που αναγνώστηκαν από τον συντακτικό αναλυτή σε ακέραιες τιμές. Η αντιστοίχηση αυτή γίνεται με την βοήθεια ενός πίνακα κατακερματισμού(hash table). Οι νέες ακέραιες τιμές των τερματικών κόμβων κάθε κυκλωματικού στοιχείου αποθηκεύονται στις μεταβλητές mna_terminal_a και mna_terminal_b της δομής δεδομένων element_t που αποθηκεύεται η πληροφορία για κάθε στοιχείο.

Σημειώνουμε πως δεν παρουσιάζουμε τα αρχεία κώδικα **mapping.h** και **mapping.c** τα οποία περιέχουν την υλοποίηση της απεικόνισης, διότι δεν έχουν καμία πρακτική αξία για την κατανόηση των υλοποιήσεων των θεμάτων που αναλύουμε στην παρούσα εργασία.

5.4. Διαμόρφωση του MNA Συστήματος

Η διαμόρφωση του γραμμικού συστήματος MNA γίνεται με χρήση της συνάρτησης AC_calculateAbSparse που υλοποιήσαμε στο εργαλείο μας, η οποία ορίζεται ως εξής:

```
/* This function calculates the sparse matrix A in compressed column avd row form and the vec-
tor b of the A*x=b linear system (for AC analysis) .
@param N        = The count of the nodes used by the Modified Nodal Analysis
@param K        = The count of the voltage sources
@param omega = Current ω frequency
@on return As_col       = The coefficient matrix in CSC form
@on return As_row       = The coefficient matrix in CRS form
@on return b            = The right-hand-side vector*/
void AC_calculateAbSparse(int N, int K, double omega);
```

Αναλυτικότερα, η συνάρτηση αυτή βασίζεται στα εξής βασικά βήματα. Αρχικά, βρίσκει το πλήθος των μη μηδενικών στοιχείων του A, nz, το οποίο στην συνέχεια δίνεται ως παράμετρος στην συνάρτηση cs_ci_spalloc για την εκχώρηση μνήμης του A σε μορφή τριστοιχίας. Στην συνέχεια διατρέχεται η λίστα των στοιχείων, list, υπολογίζοντας και καταχωρώντας την "συνεισφορά" του εκάστοτε στοιχείου στον πίνακα συντελεστών A και στο διάνυσμα δεξιού μέλους b. Υπενθυμίζουμε πως οι σχέσεις για την συνεισφορά κάθε στοιχείου περιγράφηκαν αναλυτικά στο Κεφάλαιο 1(Ενότητα 1.1.1). Μετά το πέρας αυτής της διαδικασίας το MNA σύστημα έχει διαμορφωθεί. Όμως ο πίνακας συντελεστών A είναι σε μορφή τριστοιχίας και περιέχει "διπλότυπα". Για την μετατροπή του σε μορφή συμπιεσμένων στηλών χωρίς "διπλότυπα" χρησιμοποιείται η συνάρτηση cs_ci_compress και η συνάρτηση cs_ci_dupl.

Αργότερα και κατά την κατασκευή των προρρυθμιστών για τις επαναληπτικές μεθόδους θα χρειαστούμε τον πίνακα συντελεστών σε μορφή συμπιεσμένων γραμμών που είναι η δομή δεδομένων που χρησιμοποιεί το πακέτο ZITSOL για την αποθήκευση των αραιών πινάκων. Έτσι, για την μετατροπή του από μορφή συμπιεσμένων στηλών σε μορφή συμπιεσμένων γραμμών χρησιμοποιούμε την συνάρτηση cs_ci_transpose. Όπως περιγράψαμε αναλυτικά στο Κεφάλαιο 4, η συνάρτηση αυτή βρίσκει τον ανάστροφο ενός πίνακα που είναι σε μορφή συμπιεσμένων στηλών. Αποδεικνύεται όμως το εξής: η μορφή συμπιεσμένων στηλών είναι η μορφή συμπιεσμένων γραμμών για τον ανάστροφο πίνακα. Έτσι αν έχουμε τον A σε μορφή συμπιεσμένων στηλών και βρούμε τον αντίστροφό του A^T , αυτός είναι ο A σε μορφή συμπιεσμένων γραμμών.

Υπενθυμίζουμε πως όλες οι συναρτήσεις που αναφέρουμε ότι χρησιμοποιεί η συνάρτηση AC_calculateAbSparse υλοποιούνται στο πακέτο CXSparse, το οποίο περιγράψαμε αναλυτικά στο Κεφάλαιο 4. Η διεπαφή και η υλοποίηση, του τμήματος κώδικα του εργαλείου μας που υλοποιεί τη διαμόρφωση του MNA συστήματος που μόλις περιγράψαμε, περιέχονται στα αρχεία **acmthds.h** και **acmthds.c** αντίστοιχα, τα οποία δίνονται στο Παράρτημα Α.

5.5. Άμεση LU Μέθοδος Επίλυσης Μιγαδικών Αραιών Γραμμικών Συστημάτων

Η υλοποίηση της άμεσης LU μεθόδου επίλυσης, τις οποίας τα βήματα αναλύσαμε στο Κεφάλαιο 2 γίνεται στην συνάρτηση AC_calculateLUSparse η οποία υλοποιεί την συμβολική αναδιάταξη καθώς την συμβολική και αριθμητική LU παραγοντοποίηση (βήματα 1,2 και 3) και στην συνάρτηση AC_solveASparse η οποία επιλύει τα τριγωνικά συστήματα $L\,y = b$ και $U\,x = y$ με προς τα εμπρός και προς τα πίσω αντικατάσταση (βήμα 4). Οι συναρτήσεις αυτές ορίζονται ως εξής:

```
/* The direct LU factorization method for solving complex linear systems.
@on return c_Ss            = The matrix after symbolic ordering.
@on return c_Ns            = The L, U factors and column pivoting vector.*/
void AC_calculateLUSparse();
```

και

```
/*Solves LU x = b. (b=x solution on output.)
@param N        = The count of the nodes used by the Modified Nodal Analysis
@param K        = The count of the voltage sources
@on return c_b = the solution vector*/
void AC_solveASparse(int N, int K);
```

5.6. Προρρυθμισμένες Επαναληπτικές Μέθοδοι Επίλυσης Μιγαδικών Αραιών Γραμμικών Συστημάτων

Η υλοποίηση των προρρυθμισμένων επαναληπτικών μεθόδων CG και Bi-CG, τις οποίες αναλύσαμε στο Κεφάλαιο 2 γίνεται στις συναρτήσεις AC_methodCGSparse και AC_methodBiCGSparse, αντίστοιχα, που υλοποιήσαμε στο εργαλείο μας. Οι συναρτήσεις αυτές ορίζονται ως εξής:

```
/*The iterative preconditioned Conjugate Gradient method for solving complex linear systems.
@param N = the count of the nodes used by the Modified Nodal Analysis
@param K= the count of the voltage sources
@param itol = convergence tolerance. Default value is optionITOL = 1e-3
@return iter = current iteration number
@On return c_x = the solution vector of linear system*/
int AC_methodCGSparse(int N, int K, double itol);
```

και

```
/*The iterative preconditioned Bi-Conjugate Gradient method for solving complex linear systems.
@param N = the count of the nodes used by the Modified Nodal Analysis
@param K= the count of the voltage sources
@param itol = convergence tolerance. Default value is optionITOL = 1e-3
@return iter = current iteration number
@on return c_x = the solution vector of linear system*/
int AC_methodBiCGSparse(int N, int K, double itol);
```

Αναλυτικότερα, οι συναρτήσεις αυτές βασίζονται στα εξής βασικά βήματα. Αρχικά, ο πίνακας συντελεστών ανάγεται από την δομή δεδομένων που χρησιμοποιεί το πακέτο CXSparce για την αποθήκευση των αραιών πινάκων στην δομή δεδομένων που χρησιμοποιεί το πακέτο ZITSOL, ώστε να μπορεί να χρησιμοποιηθεί στην συνέχεια από τις συναρτήσεις της ZITSOL που κατασκευάζουν τους διάφορους προρρυθμιστές που περιγράψαμε στο Κεφάλαιο 3. Αυτό γίνεται με το εξής κομμάτι κώδικα:

```
  for(i=0; i<As_row->n; i++){
     A_prec->nzcount[i] = As_row->p[i+1] - As_row->p[i];
     A_prec->ja[i] = (int*)malloc((A_prec->nzcount[i])*sizeof(int));
     A_prec->ma[i] = (complex double*)malloc((A_prec->nzcount[i])*sizeof(complex double));
     for(j=0; j<A_prec->nzcount[i]; j++){
        A_prec->ja[i][j] = As_row->i[(As_row->p[i])+j];
        A_prec->ma[i][j] = As_row->x[(As_row->p[i])+j];
     }
}
```

όπου ο As_row είναι ο πίνακας συντελεστών σε μορφή συμπιεσμένων γραμμών αποθηκευμένος στη δομή δεδομένων του πακέτου CXSparce και A_prec είναι ο πίνακας σε μορφή συμπιεσμένων γραμμών στη δομή δεδομένων που χρησιμοποιεί το πακέτο ZITSOL. Στην συνέχεια γίνεται η κατασκευή του επιθυμητού προρρυθμιστή που έχει καθοριστεί κατά την μεταγλώττιση του εργαλείου μας μέσω της επιλογής(macro flag) ILUK, ILUTD, ILUTP και CARMS για χρήση των αντίστοιχων προρρυθμιστών ή της επιλογής DIAG για χρήση του απλούστερου προρρυθμιστή με στοιχεία μόνο τα διαγώνια στοιχεία του πίνακα συντελεστών, αναφερόμενος ως διαγώνιος προρρυθμιστής. Η κατασκευή των πρώτων γίνεται, όπως ήδη έχουμε αναφέρει, με χρήση συναρτήσεων του ZITSOL πακέτου. Έπειτα, κατά τα γνωστά, γίνεται ο υπολογισμός του υπολοίπου και ξεκινούν οι επαναλήψεις της μεθόδου μέχρι να επέλθει σύγκλιση. Κατά την διάρκεια μίας επανάληψης χρειάζεται να λυθεί το $M\,\underline{z} = \underline{r}$, όπου M είναι ο προρρυθμιστής που κατασκευάστηκε πριν την έναρξη των επαναλήψεων. Η επίλυση αυτού γίνεται επίσης με συναρτήσεις του πακέτου ZITSOL εκτός από την περίπτωση που ο M είναι ο απλός διαγώνιος προρρυθμιστής. Επισημαίνουμε πως η επίλυση του απλού διαγώνιου προρρυθμιστή, καθώς και πράξεις όπως για παράδειγμα ο πολλαπλασιασμός πίνακα-διανύσματος και η εύρεση της νόρμας διανύσματος γίνονται σε ξεχωριστές συναρτήσεις που δηλώνονται και υλοποιούνται στο αρχείο **aclinsys.h** και **aclinsys.c**, αντίστοιχα. Η υλοποίηση αυτών θα δοθεί στην επόμενη ενότητα.

Η διεπαφή και η υλοποίηση, του τμήματος κώδικα του εργαλείου μας που τις προρρυθμισμένες επαναληπτικές μεθόδους CG και Bi-CG με τα βήματα που μόλις περιγράψαμε, περιέχονται στα αρχεία **ac.h** και **ac.c** αντίστοιχα, τα οποία δίνονται στο Παράρτημα Β. Επισημαίνουμε πως στην μέθοδο Bi-CG, πριν την έναρξη των επαναληπτικών βημάτων, είναι απαραίτητη η εύρεση όχι μόνο ενός προρρυθμιστή M, αλλά και του ανάστροφού του M^T, ο οποίος χρησιμοποιείται για την επίλυση του γραμμικού συστήματος $M^T\,\underline{\tilde{z}} = \underline{\tilde{r}}$ σε κάθε επαναληπτικό βήμα. Για τον υπολογισμό αυτού, βρίσκουμε αρχικά τον ανάστροφο πίνακα συντελεστών A, όπου ο A είναι αποθηκευμένος στη δομή δεδομένων που χρησιμοποιεί το πακέτο ZITSOL για την αποθήκευση αραιών πινάκων, και στην συνέχεια τον δίνουμε ως είσοδο στις διάφορες μεθόδους του ZITSOL που χρησιμοποιούμε για την κατασκευή προρρυθμιστή. Αυτό έχει ως αποτέλεσμα να λαμβάνουμε και τον ανάστροφο προρρυθμιστή M^T.

5.7. Βοηθητικές Συναρτήσεις

Για τις υλοποιήσεις των προρρυθμισμένων επαναληπτικών μεθόδων που περιγράφηκαν στην προηγούμενη ενότητα, χρησιμοποιήθηκαν κάποιες βοηθητικές συναρτήσεις. Στην συνέχεια, θα περιγράψουμε την λειτουργικότητα των σημαντικότερων εξ αυτών:

- Η συνάρτηση polarToAlgebraic μετατρέπει έναν μιγαδικό αριθμό από πολική σε τριγωνομετρική μορφή δηλαδή από μορφή $\mu\acute{\epsilon}\tau\rho o \angle\, \varphi\acute{\alpha}\sigma\eta$ σε μορφή $\alpha + \beta\, j$ όπου:

  $$\alpha = \mu\acute{\epsilon}\tau\rho o * \cos\,(\varphi\acute{\alpha}\sigma\eta)$$

  και

  $$\beta = \mu\acute{\epsilon}\tau\rho o * \sin\,(\varphi\acute{\alpha}\sigma\eta)$$

  όπου sin και cos οι συναρτήσεις υπολογισμού του ημιτόνου και του συνημίτονου αντίστοιχα που ορίζονται στην βιβλιοθήκη της C “math.h”. Επιπλέον, επισημαίνουμε πως η $\varphi\acute{\alpha}\sigma\eta$ στην πολική μορφή είναι σε μοίρες, ενώ οι συναρτήσεις sin και cos παίρνουν ως όρισμα μία ποσότητα σε ακτίνια, οπότε

και θα πρέπει να κάνουμε κατάλληλη μετατροπή της φάσης. Η συνάρτηση ορίζεται ως εξής:

```
/*Converts a complex number from polar to algebraic form.
@param magnitude = Magnitude of complex number in polar form.
@param phase = Phase of complex number in polar form.
@return c = Complex number in algebraic form.*/
cs_complex_t polarToAlgebraic(double magnitude, double phase);
```

- Η συνάρτηση algebraicToPolar μετατρέπει έναν μιγαδικό αριθμό από τριγωνομετρική σε πολική μορφή δηλαδή από μορφή $\alpha + \beta\, j$ σε μορφή $\mu\acute{\epsilon}\tau\rho o \angle\, \varphi\acute{\alpha}\sigma\eta$, όπου:

$$\mu\acute{\epsilon}\tau\rho o = \sqrt{(\alpha^2 + \beta^2)}$$

και

$$\varphi\acute{\alpha}\sigma\eta = \begin{cases} \tan^{-1}(\beta/\alpha) & \alpha\nu\ \alpha > 0 \\ \tan^{-1}(\beta/\alpha) + \pi & \alpha\nu\ \alpha < 0\ \kappa\alpha\iota\ \beta \geq 0 \\ \tan^{-1}(\beta/\alpha) - \pi & \alpha\nu\ \alpha < 0\ \kappa\alpha\iota\ \beta < 0 \\ \pi/2 & \alpha\nu\ \alpha = 0\ \kappa\alpha\iota\ \beta > 0 \\ -\pi/2 & \alpha\nu\ \alpha = 0\ \kappa\alpha\iota\ \beta < 0 \\ \delta\epsilon\nu\ o\rho\acute{\iota}\zeta\epsilon\tau\alpha\iota & \alpha\nu\ \alpha = 0\ \kappa\alpha\iota\ \beta = 0 \end{cases}$$

Για τον υπολογισμό της φάσης χρησιμοποιείται η συνάρτηση atan2 η οποία ορίζεται στην βιβλιοθήκη της C “math.h” και δίνει τα παραπάνω αποτελέσματα σε κάθε μία από τις περιπτώσεις του πραγματικού α και του φανταστικού μέρους β. Η συνάρτηση ορίζεται ως εξής:

```
/* Convert a complex number from algebraic to polar form.
@param c = Complex number in algebraic form.
@return res = Struct that contains magnitude and phase of complex number in polar
form.*/
struct polar_complex *algebraicToPolar(cs_complex_t c);
```

- Η Συνάρτηση complexAbs υπολογίζει την απόλυτη τιμή ενός μιγαδικού αριθμού, έστω $c = \alpha + \beta\, j$, η οποία ορίζεται ως εξής:

$$|c| = \sqrt{\alpha^2 + \beta^2}$$

Η συνάρτηση ορίζεται ως εξής:

```
/*Returns a complex number's absolute value.
@param c = Complex number in algebraic form.
@return  = Absolute value of complex number.*/
double complexAbs(cs_complex_t c);
```

- Η Συνάρτηση complexNormVector υπολογίζει την ευκλείδεια νόρμα ενός μιγαδικού n-διάστατου διανύσματος, έστω u, η οποία ορίζεται ως εξής:

$$\|\underline{u}\| = \sqrt{|u_1|^2 + |u_2|^2 + \cdots + |u_n|^2}$$

όπου $$|u_i|^2 = u_i \cdot \bar{u}_i = (\alpha + \beta\, j)(\alpha - \beta\, j) = \alpha^2 + \beta^2$$

Η συνάρτηση ορίζεται ως εξής:

```
/*Returns the norm Euclidean of a complex vector.
@param v = Complex vector in algebraic form.
@param n = Size of complex vector.
@return =  Norm of complex vector.*/
double complexNormVector(cs_complex_t *v, int n);
```

- Η συνάρτηση complexMultVectorVector υπολογίζει το Ευκλείδειο εσωτερικό γινόμενο δύο μιγαδικών n-διάστατων διανυσμάτων, έστω $\underline{u}$ και $\underline{v}$, το οποίο ορίζεται ως εξής:

$$\underline{u} \cdot \underline{v} = u_1 \bar{v}_1 + u_2 \bar{v}_2 + \cdots + u_n \bar{v}_n$$

όπου $u_i \bar{v}_i = (\alpha + \beta j)(\gamma - \delta j)$

Η συνάρτηση ορίζεται ως εξής:

```
/*Multiplies two complex vectors and returns the result (Euclidean inner product)
@param r = The one complex vector multiplicand in algebraic form
@param z = The other complex vector multiplicand in algebraic form
@param n = The size of both complex vector
@return sum = The Euclidean inner product of the two vectors*/
cs_complex_t complexMultVectorVector(cs_complex_t *r, cs_complex_t *z, int n);
```

- Η συνάρτηση complexMultMatrixVectorSparse πολλαπλασιάζει έναν αραιό μιγαδικό πίνακα που βρίσκεται σε μορφή συμπιεσμένων στηλών με ένα μιγαδικό διάνυσμα και επιστρέφει το αποτέλεσμα. Ορίζεται ως εξής:

```
/*Multiplies the size n*n sparse matrix A with the complex vector x and the result is
stored at complex vector y.
@param A = Matrix multiplicand in CSC format of CXsparse library struct type
@param x =  Complex vector multiplicand in algebraic form
@param y =  Where the result is stored
@param n = The size of complex vector
@On return  y = The result complex vector*/
void complexMultMatrixVectorSparse(cs_ci *A, cs_complex_t *x, cs_complex_t *y, int n);
```

- Η συνάρτηση complexMultMatrixVectorTransSparse πολλαπλασιάζει τον ανάστροφο πίνακα ενός αραιού μιγαδικού πίνακα που βρίσκεται σε μορφή συμπιεσμένων στηλών με ένα μιγαδικό διάνυσμα και επιστρέφει το αποτέλεσμα. Ορίζεται ως εξής:

```
/*Multiplies the size n*n transposed sparse matrix A with the complex vector x and the
result is stored at complex vector y.
@param A = Matrix multiplicand in CSC format of CXsparse library struct type.
@param x =  Complex vector multiplicand in algebraic form.
@param y =  Where the result is stored.
@param n = The size of complex vector.
@On return  y = The result complex vector.*/
void complexMultMatrixVectorTransSparse(cs_ci *A, cs_complex_t *x, cs_complex_t *y ,
                                                                        int n);
```

- Η συνάρτηση AC_preconditionerSolve επιλύει το γραμμικό σύστημα $M\ z = r$ στην περίπτωση που ο M είναι ο απλός διαγώνιος προρρυθμιστής. Στην περίπτωση αυτή το M είναι ένα διάνυσμα και όχι ολόκληρος πίνακας. Ορίζεται ως εξής:

```
/*Multiplies the complex vector M with the complex vector r and the result is stored at
complex vector z.
@param M = Complex vector in algebraic form (diagonal preconditioner).
@param z = Where result is stored.
@param  n = The size of complex vector.
@on return z = The solution complex vector.*/
void AC_preconditionerSolve(cs_complex_t *M, cs_complex_t *r, cs_complex_t *z, int n);
```

Αναλυτικά, η διεπαφή και οι υλοποιήσεις των παραπάνω συναρτήσεων αλλά και μερικών ακόμη βοηθητικών συναρτήσεων περιέχονται στα αρχεία **aclinsys.h** και **aclinsys.c** αντίστοιχα, τα οποία δίνονται στο Παράρτημα Γ.

5.8. Κυρίως Πρόγραμμα

Το κυρίως πρόγραμμα υλοποιείται στο αρχείο **main.c**. Σε αυτό αρχικά καλούνται οι συναρτήσεις για την συντακτική ανάλυση και την απεικόνιση. Στην συνέχεια, ανάλογα με τις επιλογές που αναγνώστηκαν από το αρχείο εισόδου, καλείται η συνάρτηση διαμόρφωσης του MNA συστήματος και είτε η άμεση LU μέθοδος, είτε η επαναληπτική μέθοδος CG, είτε η επαναληπτική μέθοδος Bi-CG μέσω κατάλληλων συναρτήσεων τις οποίες περιγράψαμε στις προηγούμενες ενότητες. Το βασικό κομμάτι κώδικα που κάνει την επιλογή της κατάλληλης συνάρτησης για την επίλυση του προκύπτοντος μιγαδικού γραμμικού αραιού συστήματος είναι το εξής:

```
for(i=0; i<dotACctr; i++){
    ac_step = (dotAC[i].fend - dotAC[i].fstart) / (dotAC[i].npoints + 1);
    for(freq=dotAC[i].fstart; freq<=dotAC[i].fend; freq+=ac_step){
      omega = 2*3.14159265*freq;    //ω = 2*π*f
      AC_calculateAbSparse(N, K, omega);
      if(optionITER){
        if(optionSPD){
          iter = AC_methodCGSparse(N, K, optionITOL);
        }
        else{
          iter = AC_methodBiCGSparse(N, K, optionITOL);
        }
      }
      else{
       AC_calculateLUSparse();
       AC_solveASparse(N, K);
      }
      writeACResultsToFile(omega, i, flag, N, K);
    }
```

όπου η συνάρτηση writeACResultsToFile, όπως δηλώνει και το όνομά της, γράφει το διάνυσμα του αποτελέσματος ή κάποιο υποσύνολο αυτού στο αρχείο “AC_RESULTS.TR” για κάθε συχνότητα ω που αντιστοιχεί στην εκάστοτε συχνότητα f της δοθείσας περιοχής σάρωσης συχνοτήτων.

Σημειώνουμε πως δεν παρουσιάζουμε τον υπόλοιπο κώδικα του αρχείου **main.c,** διότι δεν έχει καμία πρακτική αξία για την κατανόηση των υλοποιήσεων των θεμάτων που αναλύουμε στην παρούσα εργασία.

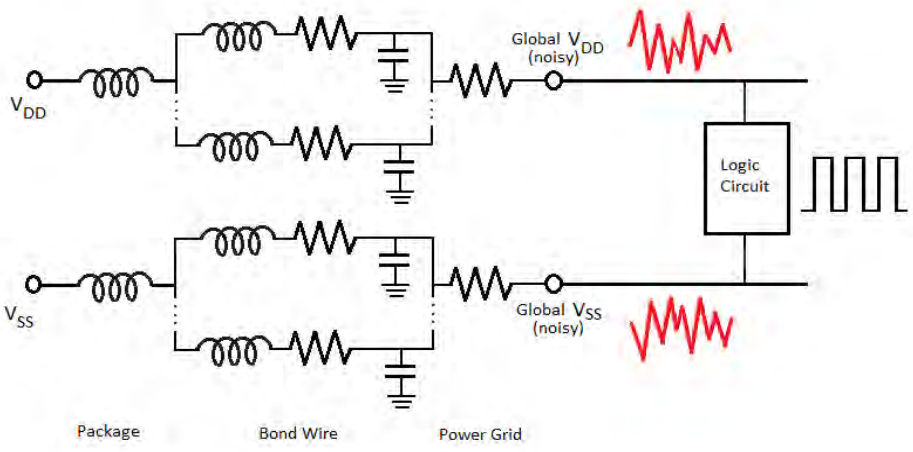
3ο μέρος:
Αριθμητικά Αποτελέσματα

Κεφάλαιο 6

Περιοχή Εφαρμογής

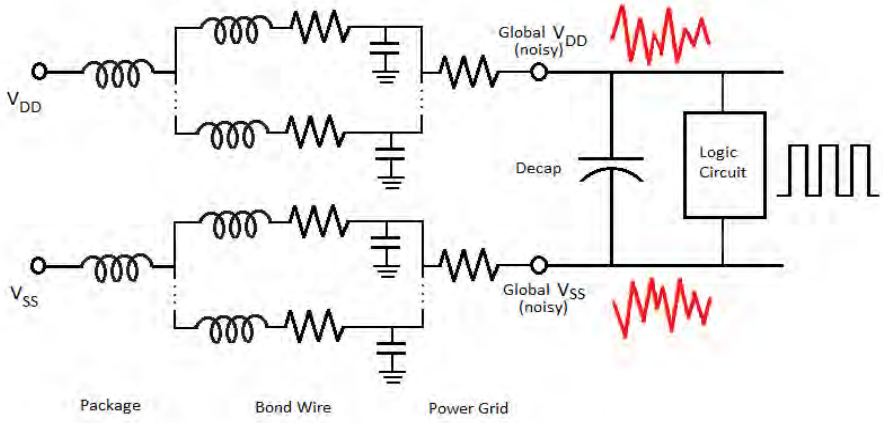
6.1. Δίκτυα Τροφοδοσίας (Power Grids)

Καθώς η τεχνολογία αναπτύσσετε, οι απαιτήσεις ισχύος στα ολοκληρωμένα κυκλώματα αυξάνονται, με αποτέλεσμα να περιέχουν τεράστια δίκτυα διανομής ισχύος ή πιο απλά δίκτυα τροφοδοσίας. Η διασύνδεση σε ένα δίκτυο τροφοδοσίας μοντελοποιείται σαν ένα κύκλωμα από αντιστάσεις και πηνία. Λόγω της ύπαρξης αντίστασης και επαγωγής καθ΄ όλη την έκταση του δικτύου τροφοδοσίας εμφανίζεται σημαντικός θόρυβος στην τάση τροφοδοσίας του δικτύου (Σχήμα 1). Πιο συγκεκριμένα, λόγω της παρασιτικής αντίστασης εμφανίζεται πτώση τάσης μεταξύ της τάσης τροφοδοσίας και των λογικών πυλών γνωστή ως IR-drop, ενώ λόγω της παρασιτικής επαγωγής (αλλά και του ρυθμού μεταβολής των ρευμάτων μετάβασης) εμφανίζεται επαγωγικός θόρυβος γνωστός ως Ldi/dt [21].



Σχήμα 1: Δίκτυο Τροφοδοσίας

Ο θόρυβος έχει δυσμενή επίπτωση στην αξιοπιστία του κυκλώματος και συνεπώς η απόδοση και η λειτουργικότατα του υποβιβάζονται. Είναι λοιπόν προφανές πως το δίκτυο τροφοδοσίας πρέπει να σχεδιάζεται ώστε ο θόρυβος να είναι σε ένα ανεκτό επίπεδο για να εξασφαλιστεί η αξιοπιστία ενός κυκλώματος. Για να μειωθεί ο θόρυβος στην τάση τροφοδοσίας, συνήθως ένας επαρκής αριθμός πυκνωτών αποσύζευξης συνδέονται μεταξύ των κόμβων του δικτύου τροφοδοσίας τάσης και των κόμβων του δικτύου γείωσης (Σχήμα 2). Οι πυκνωτές αποσύζευξης αποθηκεύουν ηλεκτρικό φορτίο και “παρέχουν” το φορτίο αυτό σε περίπτωση διακυμάνσεων της τάσης και με αυτό τον τρόπο μειώνουν τις διακυμάνσεις της τάσης τροφοδοσίας και μειώνουν τις επιπτώσεις του θορύβου της τροφοδοσίας στα “γειτονικά” κυκλώματα.



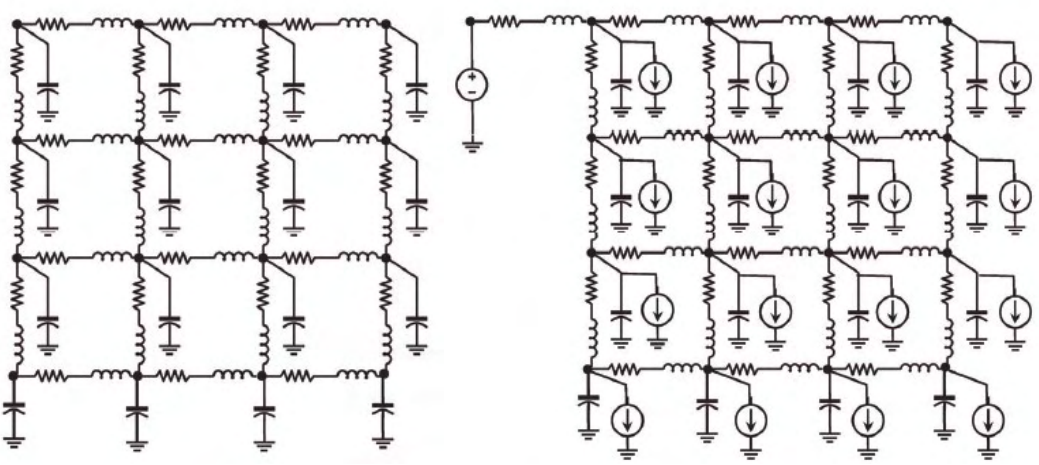
Σχήμα 2: Δίκτυο Τροφοδοσίας Σχήματος 1 με προσθήκη πυκνωτή αποσύζευξης (Decap)

Συμβολή της AC ανάλυσης

Η AC ανάλυση εφαρμόζεται για τον υπολογισμό εκείνου του θορύβου της τροφοδοσίας που εξαρτάται από την συχνότητα και συνεισφέρει στον καθορισμό του μεγέθους των πυκνωτών αποσύζευξης έτσι ώστε να αντισταθμιστεί ο θόρυβος αυτός. Επίσης, και σε περιπτώσεις "γειτονικών" κυκλωμάτων, όπως στα Systems on-Chip (SoC), που περιέχουν και ψηφιακά και αναλογικά κυκλώματα και όπου το περιεχόμενο του θορύβου τροφοδοσίας συναρτήσει της συχνότητας επηρεάζει το αναλογικό κομμάτι σε συγκεκριμένες συχνότητες, η AC ανάλυση είναι χρήσιμη διότι συνεισφέρει στον υπολογισμό των πυκνωτών αποσύζευξης για την μείωση των επιπτώσεων του θορύβου αυτού στο αναλογικό κομμάτι.

Συμβατικό μοντέλο κατανάλωσης ρεύματος

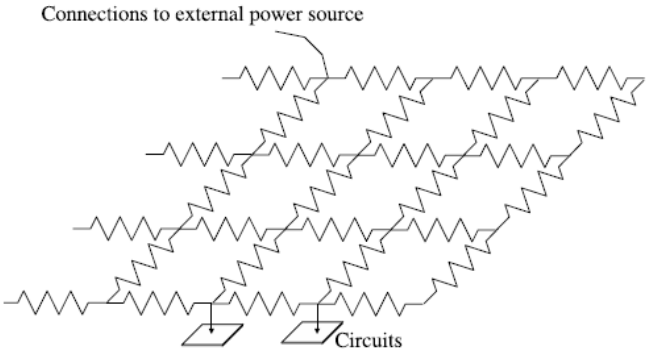
Τα σύγχρονα ολοκληρωμένα κυκλώματα μπορούν να περιέχουν δισεκατομμύρια τρανζίστορ. Αυτά τα τρανζίστορ είναι "συστατικά" των λογικών πυλών, οι οποίες είναι οι καταναλωτές ρεύματος του δικτύου τροφοδοσίας. Καθώς εναλλάσσονται, κάθε λογική πύλη καταναλώνει ρεύμα από την τροφοδοσία μέσω του δικτύου(grid). Αφού τα τρανζίστορ είναι μη-γραμμικά στοιχεία, ο συμβατικός τρόπος μοντελοποίησης των τρανζίστορ είναι η προσομοίωση επιμέρους ομάδων κυκλωμάτων, περιλαμβανομένων και των τρανζίστορ και των παρασιτικών στοιχείων στη διασύνδεση τροφοδοσίας και στην συνέχεια η αντικατάσταση κάθε επιμέρους ομάδας από μία ιδανική πηγή ρεύματος (Σχήμα 3). Οι κυματομορφές ρεύματος που αντιστοιχούν στις πηγές αυτές χρησιμεύουν ως πηγές διέγερσης για το δίκτυο. Με αυτό το τρόπο το πρόβλημα γίνεται γραμμικό. Παρόλο που αυτό το μοντέλο είναι κάπως διαισθητικό, η χρήση του είναι περίπλοκη και οι προσομοιώσεις με αυτό το μοντέλο χρειάζονται πολύ χρόνο λόγω του μεγάλου μεγέθους του δικτύου και του μεγάλου αριθμού των πηγών ρεύματος. Για τον λόγο αυτό είναι σημαντικό να μπορούμε να προσομοιώσουμε με αποδοτικό τρόπο τα κυκλώματα που μοντελοποιούν τα δίκτυα τροφοδοσίας[22].



Σχήμα 3: (Αριστερά) Ένα RLC μοντέλο από ένα on-chip δίκτυο κατανομής ισχύος, (Δεξιά) Ένα πλήρες RLC μοντέλο και συμβατικό μοντέλο κατανάλωσης ρεύματος ως ιδανική πηγή ρεύματος.

6.2. Προβλήματα Αναφοράς (Benchmarks)

Για την εφαρμογή των υλοποιήσεών μας, ως προβλήματα αναφοράς χρησιμοποιήθηκαν τα γραμμικά κυκλώματα εισόδου που παρέχονται από την IBM και τα οποία αντιστοιχούν σε πραγματικά δίκτυα τροφοδοσίας(Power Grids) [23]. Τα συγκεκριμένα δίκτυα τροφοδοσίας, λόγω του ότι σχεδιάστηκαν για εφαρμογή DC ανάλυσης, αποτελούνται μόνο από αντιστάσεις (Σχήμα 4). Επίσης, οι πηγές είναι συνεχής, δηλαδή έχουν μόνο DC (direct current) τιμές. Για τους σκοπούς της AC ανάλυσης την οποία εξετάζουμε, πρέπει τα κυκλώματα εισόδου να περιέχουν τουλάχιστον μία εναλλασσόμενη πηγή, ενώ σκόπιμο είναι για τον απαραίτητο έλεγχο των υλοποιήσεων μας να περιέχουν εκτός από αντιστάσεις και κάποια πηνία ή/και πυκνωτές. Συνεπώς, τροποποιήσαμε τα δίκτυα τροφοδοσίας της IBM ώστε να εξυπηρετούν τους σκοπούς μας. Έτσι, προσθέσαμε σε κάθε πηγή και AC τιμή, επιπρόσθετα της DC τιμής. Υπενθυμίζουμε πως η AC τιμή αποτελείται από το μέτρο και την φάση. Η τιμή του μέτρου της AC τιμής κάθε πηγής τέθηκε ίση με την DC αυτής, ενώ η τιμή της φάσης είτε τέθηκε μηδέν, είτε τυχαία σε μία τιμή. Για απλοποίηση στους υπολογισμούς δεν προβήκαμε σε προσθήκη πυκνωτών και πηνίων στα κυκλώματα εισόδου αυτά. Η απλοποίηση έγκειται στο γεγονός ότι σε κάθε βήμα της AC ανάλυσης όπου αλλάζει η συχνότητα λειτουργίας, επηρεάζεται μόνο η τιμή των πηνίων και των πυκνωτών, όπως εξηγήσαμε αναλυτικά στο Κεφάλαιο 1 (Ενότητα 1.1.1). Συνεπώς, για κύκλωμα εισόδου που αποτελείται μόνο από αντιστάτες, σε κάθε βήμα της AC ανάλυσης, το προκύπτων γραμμικό σύστημα θα είναι το ίδιο και επομένως και το πλήθος των επαναλήψεων που απαιτούνται για την σύγκλιση μιας επαναληπτικής μεθόδου θα είναι το ίδιο. Τέλος, προσθέσαμε σε κάθε δίκτυο τροφοδοσίας, κατάλληλες εντολές για την ενεργοποίηση της AC ανάλυσης και την εκτύπωση των επιθυμητών αποτελεσμάτων. Συγκεκριμένα, ορίσαμε την περιοχή σάρωσης των συχνοτήτων από $1\,Hz$ έως $100\,Hz$ καθώς και εκτέλεση της AC ανάλυσης σε 100 συχνοτήτες στην περιοχή αυτή, ισαπέχουσων γραμμικά. Η εκτύπωση των αποτελεσμάτων έγινε για τυχαίο υποσύνολο 10 κόμβων του συνολικού πλήθους των κόμβων.



Σχήμα 4: Ισοδύναμο κύκλωμα ενός μικρού κομματιού ενός τυπικού δικτύου τροφοδοσίας της IBM

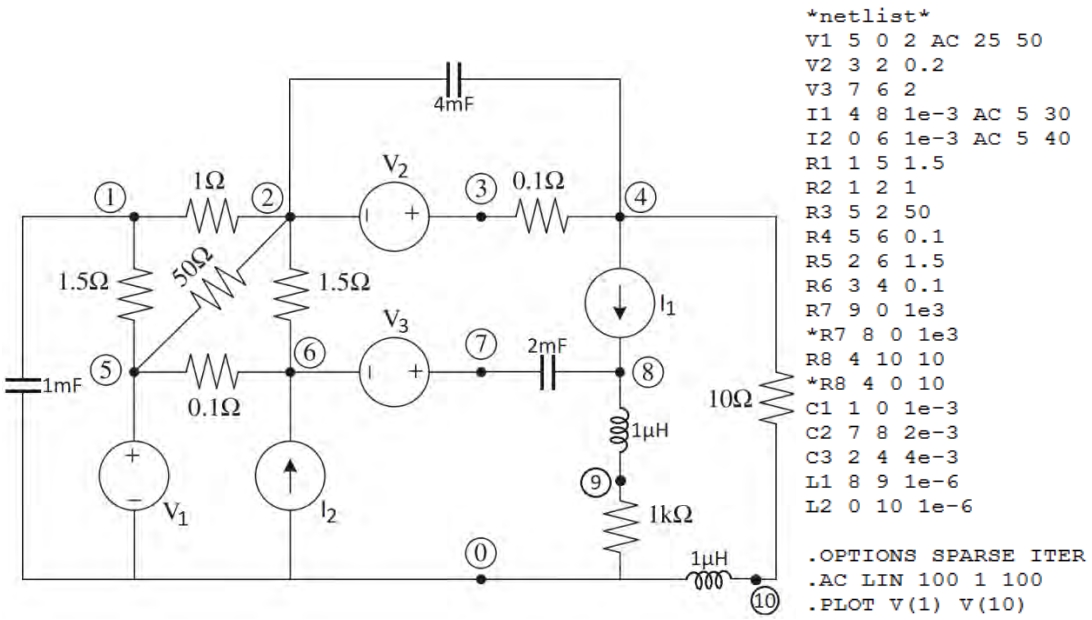
Στον ακόλουθο πίνακα, δίνονται συνοπτικά τα σημαντικότερα στοιχεία των τελικών κυκλωμάτων εισόδου που χρησιμοποιήσαμε για την συγκριτική μας μελέτη.

Name	#Elements	#V sources	#I sources	#Nodes	#Nonzeros
ac1	55 109	14 308	10 774	30 636	176 509
ac2	246 581	330	37 926	127 236	833 630
ac3	1 603 581	955	201 054	851 582	5 607 243
ac4	1 838 583	962	276 976	953 581	6 243 542
ac5	2 156 735	539 087	540 800	1 079 308	6 462 909
ac6	3 246 725	836 816	761 484	1 670 492	9 939 821
acnew1	2 710 976	930 599	357 930	1 461 039	9 409 319
acnew2	2 711 240	955	357 930	1 461 039	9 410 375

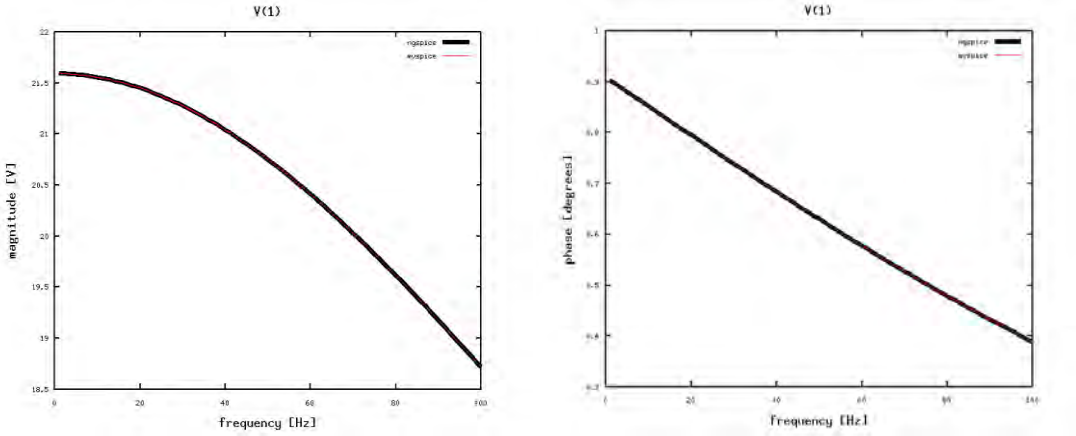
Πίνακας 2: IBM Power Grid Benchmarks

6.2.1. Έλεγχος αριθμητικών αποτελεσμάτων

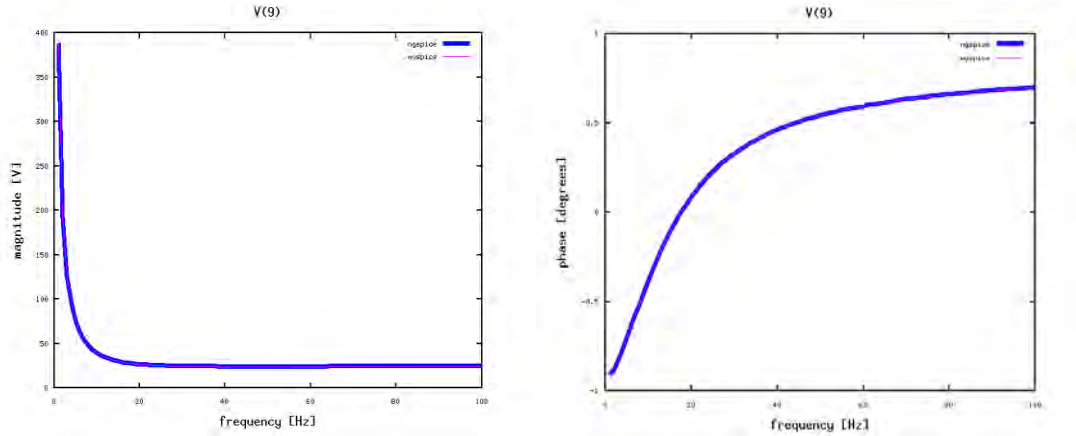
Για την επιβεβαίωση της ορθής λειτουργίας των υλοποιήσεών μας και στην περίπτωση κυκλωμάτων εισόδου που περιέχουν πηνία και πυκνωτές, χρησιμοποιήσαμε κάποια μικρότερα κυκλώματα εισόδου. Για τον έλεγχο των αριθμητικών αποτελεσμάτων χρησιμοποιήθηκε το πρόγραμμα προσομοίωσης NGSPICE. Ενδεικτικά, για το κύκλωμα του Σχήμα 5, δίνουμε γραφικά τα αποτελέσματα για τους κόμβους $V(1)$ και $V(9)$ όπως προέκυψαν από την προσομοίωση του με το NGSPICE, καθώς και το δικό μας εργαλείο προσομοίωσης.



Σχήμα 5: Κύκλωμα Εισόδου



Σχήμα 6: Αποτελέσματα πλάτους και φάσης της τάσης του κόμβου 1 συναρτήσει της συχνότητας



Σχήμα 7: Αποτελέσματα πλάτους και φάσης της τάσης του κόμβου 9 συναρτήσει της συχνότητας

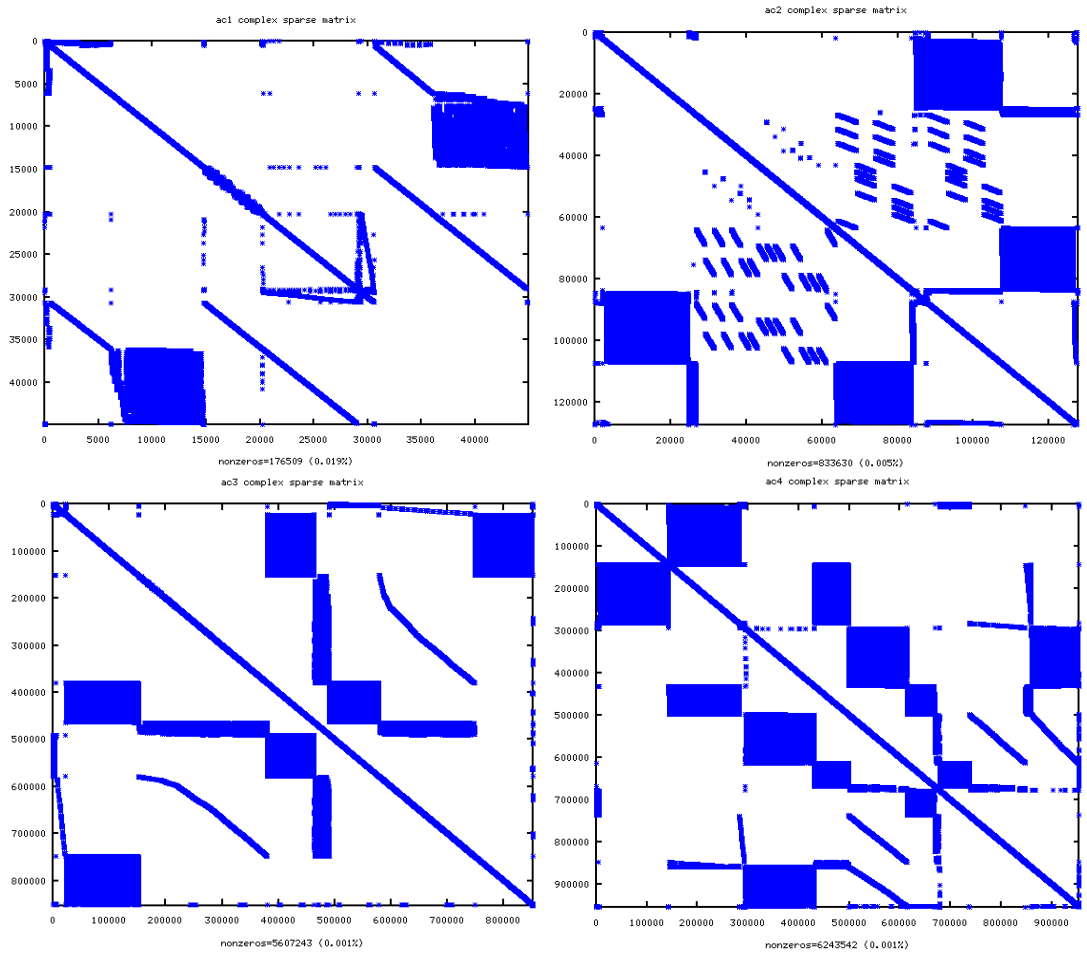
Παρατηρούμε πως τα αποτελέσματα συμπίπτουν τόσο για τον υπολογισμό του πλάτους της τάσης στους κόμβους, όσο και για τον υπολογισμό της φάσης. Αυτό επιβεβαιώνει πως οι υλοποιήσεις του εργαλείου μας είναι ορθές.

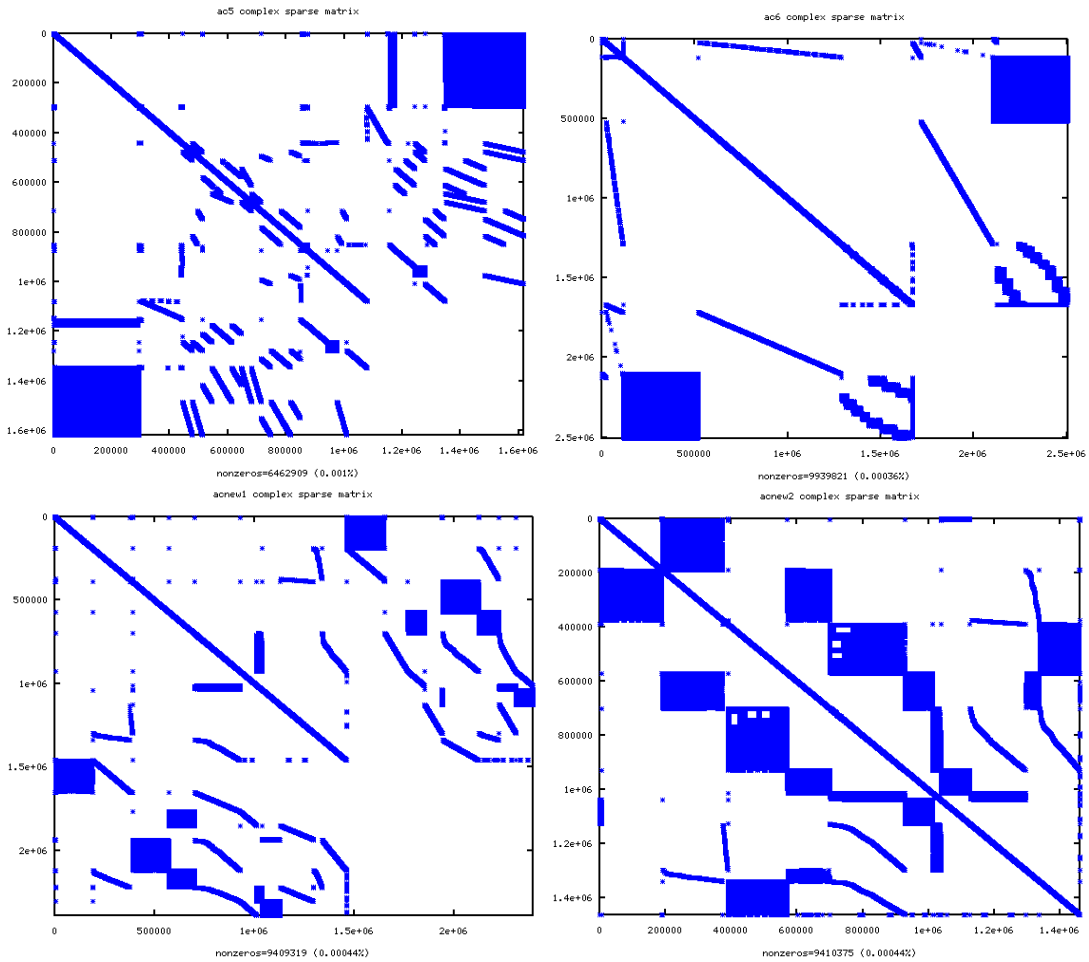
Κεφάλαιο 7

Συγκριτική Μελέτη

7.1. Σύγκριση Απόδοσης Προρρυθμιστών

Συγκρίνουμε την απόδοση της επαναληπτικής μεθόδου Bi-CG με χρήση των Προρρυθμιστών ILUK, ILUT, ILUTP και ARMS τους οποίους περιγράψαμε αναλυτικά στο Κεφάλαιο 3, για την προσομοίωση των δικτύων τροφοδοσίας που παρουσιάζονται στο Κεφάλαιο 6 (Πίνακας 2) [24, 25]. Το Σχήμα 8 απεικονίζει την δομή του πίνακα συντελεστών του προκύπτοντος γραμμικού συστήματος προς επίλυση για κάθε ένα από τα δίκτυα αυτά. Αυτό είναι χρήσιμο για να παρατηρήσουμε την κατανομή των μη μηδενικών στοιχείων κάθε κυκλώματος και να βγάλουμε κάποια προκαταρκτικά συμπεράσματα για την εφαρμογή των Προρρυθμιστών που μελετάμε σε καθένα από αυτά.





Σχήμα 8: Δομή πινάκων συντελεστών των κυκλωμάτων ac1 και ac2 (πάνω), ac3 και ac4(μέση-πάνω), ac5 και ac6(μέση-κάτω) και acnew1 και acnew2(κάτω)

Παρατηρούμε πως για τα κυκλώματα ac1, ac5, ac6 και acnew1 αναμένουμε οι ILUT και ILUT να μην καταφέρουν να κατασκευάσουν έναν προρρυθμιστή για την επίλυση του προκύπτοντος γραμμικού συστήματος, διότι ο αντίστοιχος πίνακας συντελεστών περιέχει μηδενικά στην διαγώνιο. Ωστόσο, οι ILUTP και ARMS αναμένουμε να καταφέρουν να κατασκευάσουν ένα αποτελεσματικό προρρυθμιστή αφού λύνουν το πρόβλημα αυτό χρησιμοποιώντας την τεχνική της οδήγησης.

Στις δοκιμές που ακολουθούν επιτρέπουμε μέγιστο πλήθος επαναλήψεων όσο και το μέγεθος n του πίνακα συντελεστών και ορίζουμε την ανοχή $itol$ για το κριτήριο τερματισμού ως $1.0 * 10^{-3}$. Στους ακόλουθους πίνακες, παραθέτουμε για κάθε κύκλωμα ένα πίνακα με τον χρόνο σε δευτερόλεπτα(seconds) για την κατασκευή του εκάστοτε προρρυθμιστή ($Prec\ Time$) και το πλήθος των επαναλήψεων της Bi-CG μεθόδου με χρήση του εκάστοτε προρρυθμιστή ($\#Iterations$) καθώς και τον αντίστοιχο συνολικό χρόνο για τον υπολογισμό του προρρυθμιστή και την εκτέλεση των επαναλήψεων ($Total\ Time$).

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.001	10082	68.10
ILUK	f	f	f
ILUT	f	f	f
ILUTP	0.22	3	0.29
ARMS	0.50	4	0.61

Πίνακας 3: Αποτελέσματα για το ac1 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.01	717	17.62
ILUK	0.44	24	1.88
ILUT	0.14	39	1.72
ILUTP	3.29	5	3.76
ARMS	2.94	11	4.22

Πίνακας 4: Αποτελέσματα για το ac2 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.03	f	f
ILUK	14.35	262	238.59
ILUT	0.91	884	256.25
ILUTP	9.11	f	f
ARMS	15.81	f	f

Πίνακας 5: Αποτελέσματα για το ac3 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.03	903	189.16
ILUK	5.41	44	28.81
ILUT	1.29	64	22.85
ILUTP	97.20	7	102.88
ARMS	47.03	15	69.23

Πίνακας 6:Αποτελέσματα για το ac4 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.04	6996	1964.85
ILUK	f	f	f
ILUT	f	f	f
ILUTP	235.32	97	315.61
ARMS	82.61	11	99.14

Πίνακας 7: Αποτελέσματα για το ac5 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.09	21195	7052.75
ILUK	f	f	f
ILUT	f	f	f
ILUTP	108.73	19	150.58
ARMS	88.93	12	127.08

Πίνακας 8: Αποτελέσματα για το ac6 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.07	15821	5562.21
ILUK	f	f	f
ILUT	f	f	f
ILUTP	278.39	f	f
ARMS	90.95	9	192.59

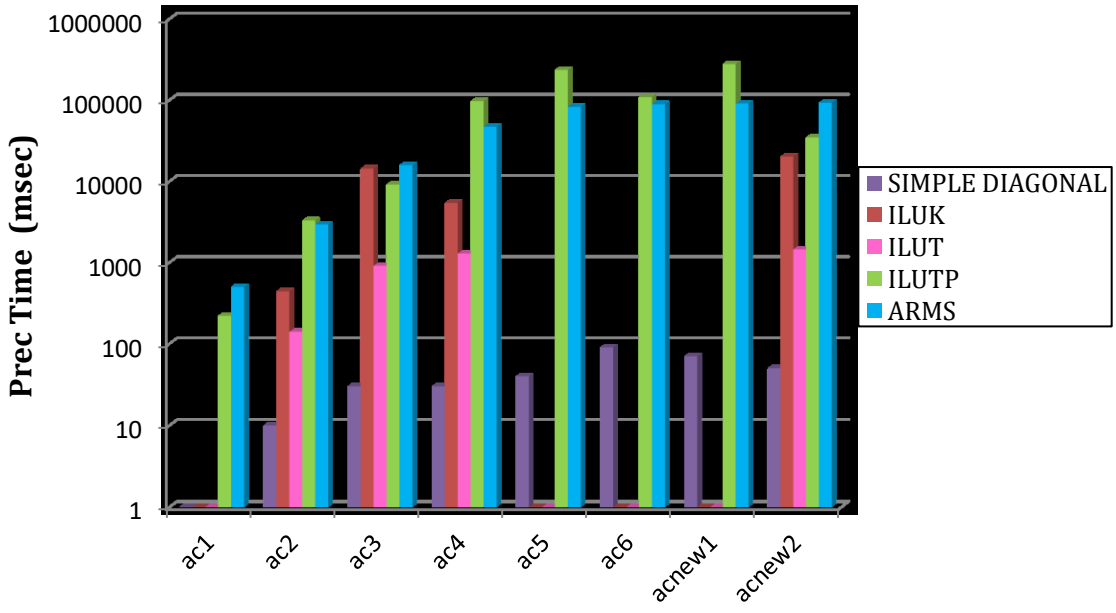
Πίνακας 9: Αποτελέσματα για το acnew1 benchmark

Preconditioner	Prec Time	#Iterations	Total Time
SIMPLE DIAGONAL	0.05	f	f
ILUK	20.13	281	370.67
ILUT	1.45	1051	511.22
ILUTP	34.72	f	f
ARMS	92.73	f	f

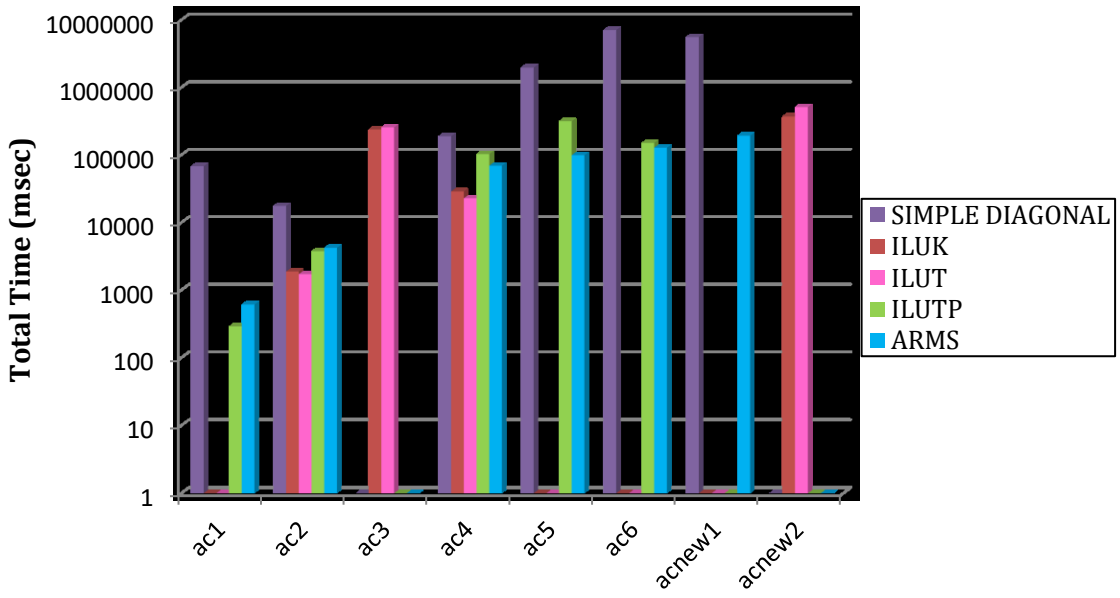
Πίνακας 10: Αποτελέσματα για το acnew2 benchmark

όπου με f (fail) σημειώνεται ο $Prec\ Time$ όταν η μέθοδος αποτυγχάνει να κατασκευάσει τον αντίστοιχο προρρυθμιστή και μόνο τα $\#Iterations$ και $Total\ Time$ όταν κατασκευάζεται ο αντίστοιχος προρρυθμιστής αλλά η Bi-CG δεν καταφέρνει να συγκλίνει. Έτσι, παρατηρούμε πως όπως αναμέναμε για τα κυκλώματα ac1, ac5, ac6 και acnew1 οι ILUT και ILUT αποτυγχάνουν να κατασκευάσουν τον αντίστοιχο προρρυθμιστή για την επίλυση του προκύπτοντος γραμμικού συστήματος. Επίσης, για τα κυκλώματα ac3 και acnew2, οι ILUTP και ARMS δεν καταφέρνουν να συγκλίνουν, ενώ για το acnew1 δεν καταφέρνει να συγκλίνει μόνο ο ILUTP.

Τα συγκεντρωτικά αποτελέσματα για τον χρόνο υπολογισμού του εκάστοτε προρρυθμιστή, καθώς και του συνολικού χρόνου εκτέλεσης της επαναληπτικής μεθόδου Bi-CG με χρήση του εκάστοτε προρρυθμιστή, φαίνονται στα παρακάτω σχήματα:



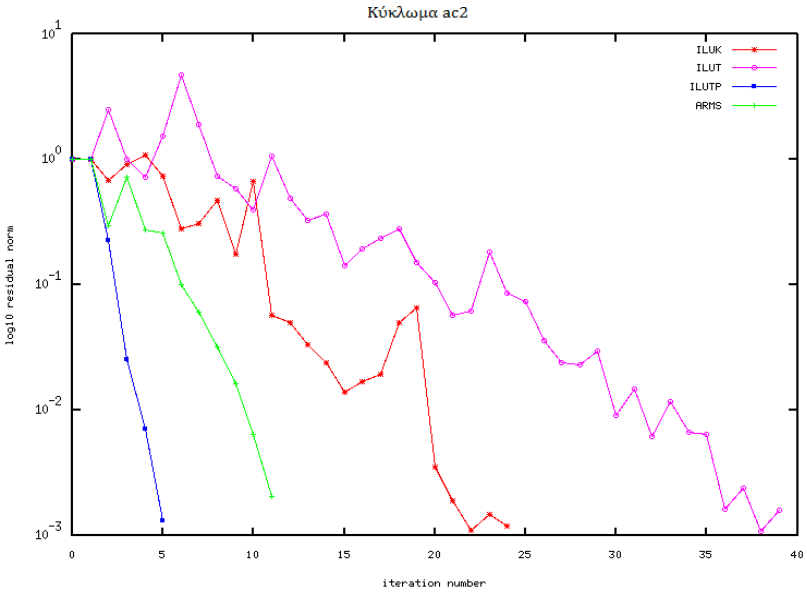
Σχήμα 9: Συγκριτικά Αποτελέσματα Χρόνου Κατασκευής Προρρυθμιστών



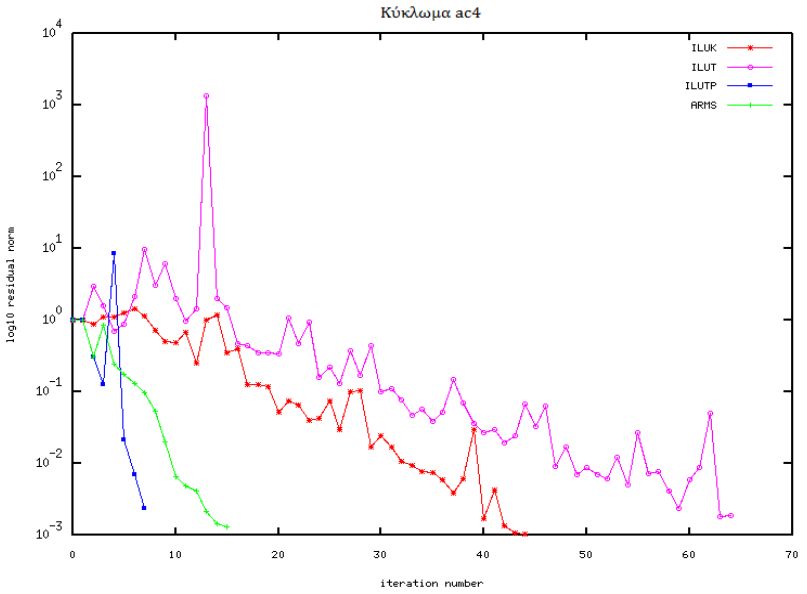
Σχήμα 10: Συγκριτικά Αποτελέσματα Συνολικού Χρόνου Εκτέλεσης της Προρρυθμισμένης Bi-CG

Από άποψη χρόνου εκτέλεσης, είναι φανερό πως ο απλός διαγώνιος προρρυθμιστής είναι πιο αργός σε σχέση με όλους τους υπόλοιπους προρρυθμιστές. Οι προρρυθμιστές ILUK και ILUT δίνουν πολύ κοντινούς χρόνους εκτέλεσης, με τον ILUT να είναι ελαφρώς πιο γρήγορος. Ομοίως για τους προρρυθμιστές ILUTP και ARMS, όπου εδώ ο ARMS είναι ελαφρώς πιο γρήγορος. Συγκεντρωτικά, οι προρρυθμιστές ILUK και ILUT φαίνεται να είναι πιο γρήγοροι σε σχέση με τους υπόλοιπους, όμως βασικό τους μειονέκτημα είναι ότι δεν μπορούν να διαχειριστούν την περίπτωση που ο πίνακας συντελεστών περιέχει μηδενικά στοιχεία έτσι ώστε να κατασκευάσουν τον αντίστοιχο προρρυθμιστή. Από την άλλη, οι προρρυθμιστές ILUTP και ARMS καταφέρνουν πάντα να διαχειριστούν αυτήν την περίπτωση, αλλά δεν είναι λίγες οι φορές που αποτυγχάνουν να συγκλίνουν.

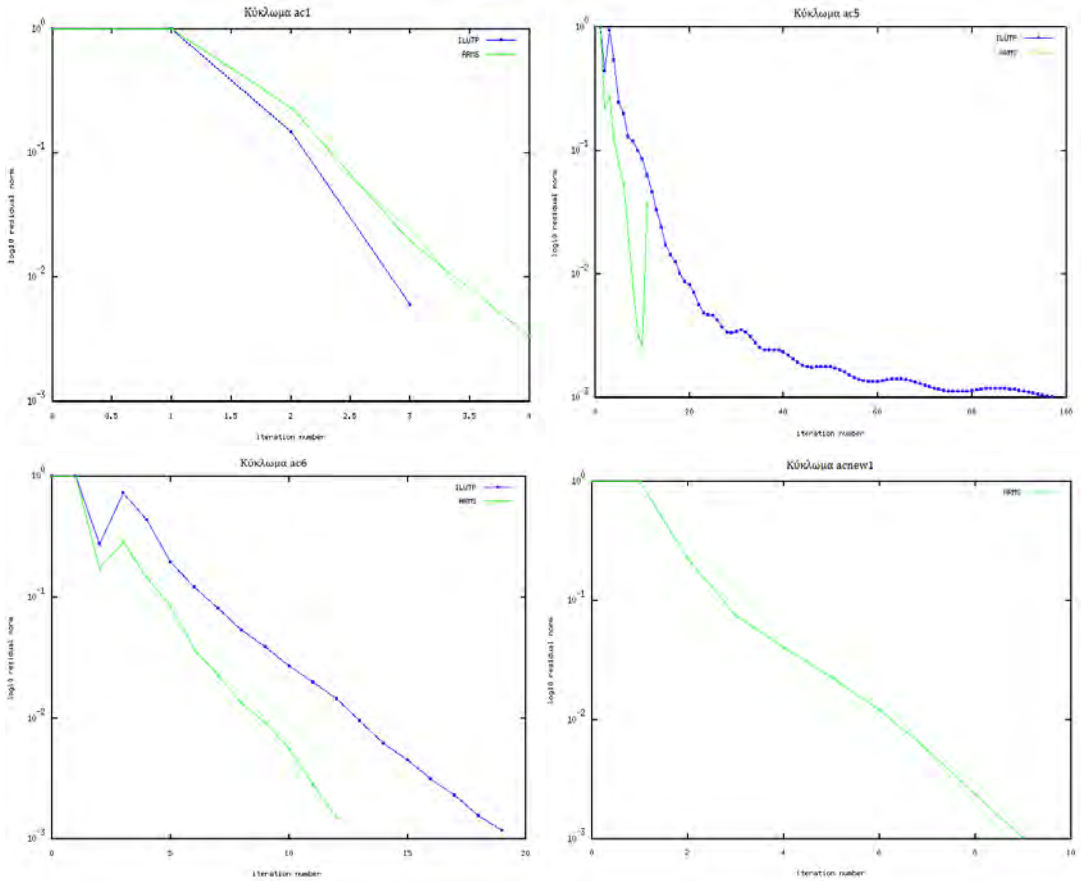
Από άποψη απαιτούμενου χρόνου για την κατασκευή του κάθε προρρυθμιστή, οι ILUTP και ARMS χρειάζονται περισσότερο χρόνο, αλλά αυτό συμβαίνει διότι τελικώς επιτυγχάνουν την κατασκευή ενός πιο πυκνού προρρυθμιστή το οποίο έχει ως αποτέλεσμα την γρηγορότερη σύγκλιση δηλαδή την σύγκλιση σε λιγότερες επαναλήψεις. Για παράδειγμα, για το κύκλωμα ac3, η ILUT συγκλίνει σε 64 επαναλήψεις για χρόνο $Prec\ Time = 1.29$, ενώ η ILUTP μόλις σε 7, για χρόνο όμως $Prec\ Time = 97.20$. Επίσης, όπως φαίνεται στα παρακάτω σχήματα, από άποψη πλήθους επαναλήψεων και νόρμας υπολοίπου, φαίνεται πως οι ILUTP και ARMS δίνουν με διαφορά καλύτερα αποτελέσματα, αφού επιτυγχάνουν, πέραν της πιο γρήγορης σύγκλισης, και ομαλότερη σύγκλιση σε σχέση με του προρρυθμιστές ILUT και ILUK. Με τον όρο ομαλή σύγκλιση εννοούμε πως σε κάθε επανάληψη γίνεται όλο και καλύτερη προσέγγιση της λύσης, κάτι που είναι προφανές πως δεν συμβαίνει πολύ πιο έντονα με τους προρρυθμιστές ILUT και ILUK απ' ότι με τους ILUTP και ARMS. Όσον αφορά την μεταξύ τους σύγκριση των ILUK και ILUT, ο πρώτος συγκλίνει σε αρκετά λιγότερες επαναλήψεις από τον πρώτο αλλά προφανώς ο χρόνος που χρειάζεται για την κατασκευή του προρρυθμιστή είναι αρκετά μεγαλύτερος. Επιπλέον, για την σύγκριση των ILUTP και ARMS, παρατηρούμε πως ο ARMS συγκλίνει πιο γρήγορα στις δυσκολότερες περιπτώσεις, ενώ χρειάζεται λιγότερο χρόνο για την κατασκευή του αντίστοιχου προρρυθμιστή και φαίνεται επίσης πως αποτυγχάνει να συγκλίνει σε λιγότερες περιπτώσεις.



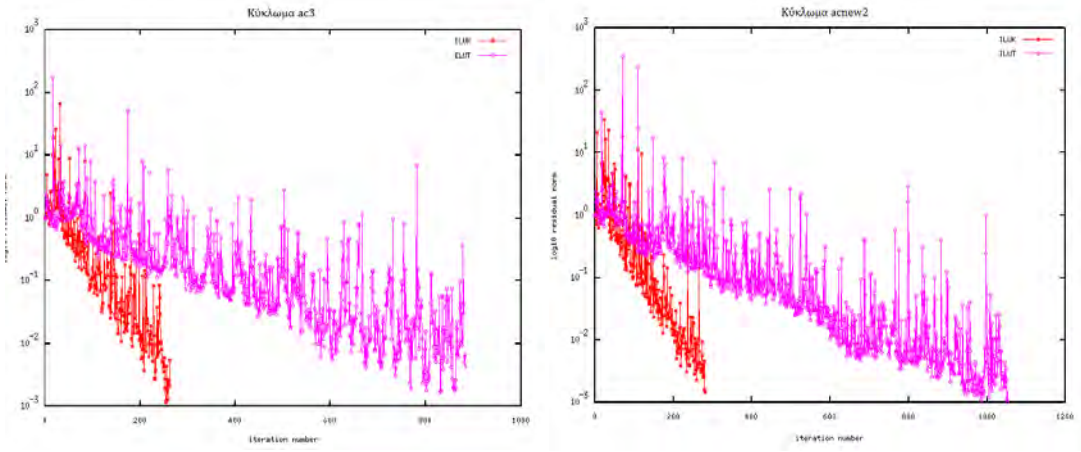
Σχήμα 11: Προφίλ Σύγκλισης για το κύκλωμα ac2



Σχήμα 12: Προφίλ Σύγκλισης για το κύκλωμα ac4



Σχήμα 13: Προφίλ Σύγκλισης για τα κυκλώματα ac1, ac5, ac6 και acnew1



Σχήμα 14: Προφίλ Σύγκλισης για τα κυκλώματα ac3 και acnew2

όπου στα σχήματα με τα προφίλ σύγκλισης των κυκλωμάτων δεν δίνεται η σύγκλιση για χρήση του απλού διαγώνιου προρρυθμιστή αφού η σύγκλιση αυτού είναι με μεγάλη διαφορά χειρότερη σε σχέση με τους υπόλοιπους προρρυθμιστές. Επίσης, είναι προφανές πως στις περιπτώσεις όπου γίνεται σύγκριση των αποτελεσμάτων μόνο για μερικούς από τους προρρυθμιστές τους οποίους εξετάζουμε, αυτό συμβαίνει διότι με την χρήση των υπόλοιπων προρρυθμιστών η επαναληπτική μέθοδος δεν κατάφερε να συγκλίνει.

7.2. Σύγκριση της άμεσης μεθόδου LU με την επαναληπτική μέθοδο Bi-CG

Παρά τις τεράστιες προσπάθειες που έχουν γίνει τις τελευταίες δεκαετίες για την ανάπτυξη επαναληπτικών αριθμητικών μεθόδων για την επίλυση μεγάλων γραμμικών συστημάτων, παρατηρείται ότι αυτές εξακολουθούν να μην χρησιμοποιούνται τόσο συχνά όσο οι άμεσες μέθοδοι. Η βασική αδυναμία των επαναληπτικών επιλυτών είναι η μη επαρκής αξιοπιστία τους, αφού υπάρχουν φορές που είτε δεν συγκλίνουν είτε απλά καταρρέουν. Πιο απλά, ένας επαναληπτικός επιλυτής μπορεί να είναι πολύ αποτελεσματικός σε ένα συγκεκριμένο τύπο προβλήματος ή για συγκεκριμένες ιδιότητες του πίνακα συντελεστών του συστήματος προς επίλυση και έτσι η περιοχή εφαρμογής του είναι σχετικά περιορισμένη. Από την άλλη, οι άμεσες μέθοδοι επίλυσης συνήθως είναι υπερβολικά αργές για την επίλυση μεγάλων προβλημάτων όπου ο πίνακας συντελεστών περιέχει στοιχεία της τάξης μερικών εκατοντάδων χιλιάδων και πόσο μάλλον όταν αυτός περιέχει στοιχεία της τάξης εκατομμυρίων. Σε αυτές τις περιπτώσεις, οι επαναληπτικές μέθοδοι επίλυσης συχνά είναι πολύ πιο αποτελεσματικές από τις άμεσες.

Ενδιαφέρον είναι να συγκρίνουμε την συμπεριφορά των άμεσων μεθόδων για την επίλυση των αρκετά μεγάλων γραμμικών συστημάτων που προκύπτουν από τα κυκλώματα του Πίνακας 2 σε σχέση με αυτή των επαναληπτικών μεθόδων. Για τον σκοπό αυτό υλοποιήσαμε την άμεση μέθοδο επίλυσης LU για μιγαδικά συστήματα, με την βοήθεια συναρτήσεων του πακέτου CXSparse που περιγράψαμε στο Κεφάλαιο 4. Η υλοποίηση αυτή δίνεται στο Παράρτημα Β. Στον ακόλουθο πίνακα, παραθέτουμε για κάθε κύκλωμα ένα πίνακα με τον χρόνο σε δευτερόλεπτα που χρειάστηκε για την επίλυση του προκύπτοντος γραμμικού συστήματος, η άμεση μέθοδος LU, καθώς και τον αντίστοιχο καλύτερο και χειρότερο χρόνο που χρειάστηκε η επαναληπτική μέθοδος Bi-CG με χρήση των διαφορετικών προρρυθμιστών που δοκιμάσαμε στην προηγούμενη ενότητα.

	Direct Solver LU	Iterative Solver Bi-CG	
Name	**Time**	**Worst Time**	**Best Time**
ac1	0.32	f	0.29
ac2	17.54	17.62	1.72
ac3	f	f	238.59
ac4	f	189.16	22.85
ac5	f	f	99.14
ac6	f	f	127.08
acnew1	f	f	192.59
acnew2	f	f	370.67

Πίνακας 11: Σύγκριση Αποτελεσμάτων Άμεσης LU και Bi-CG

όπου *f* για την άμεση και για την επαναληπτική μέθοδο σημαίνει ότι μετά από αναμονή ενός εύλογου χρονικού διαστήματος η μέθοδος τερματίστηκε ή δεν σύγκλινε αντίστοιχα.

Παρατηρούμε πως η άμεση μέθοδος επιλύει αποτελεσματικά μόνο τα προκύπτοντα γραμμικά συστήματα των δύο μικρότερων κυκλωμάτων των οποίων οι πίνακες συντελεστών περιείχαν πλήθος στοιχείων της τάξης των 200 και 800 χιλιάδων αντίστοιχα, με σημαντική διαφορά στο χρόνο επίλυσης για το πρώτο αρκετά μικρότερο σύστημα. Για τα προκύπτοντα γραμμικά συστήματα των υπόλοιπων κυκλωμάτων η άμεση μέθοδος δεν καταφέρνει να δώσει αποτελέσματα σε ένα εύλογο χρονικό διάστημα. Υπενθυμίζουμε πως οι πίνακες συντελεστών αυτών των κυκλωμάτων είναι της τάξης των 5 έως 9 εκατομμυρίων, δηλαδή πολύ μεγαλύτεροι από τους δύο πρώτους. Από την άλλη μεριά, η επαναληπτική μέθοδος καταφέρνει να επιλύσει όλα τα

συστήματα με χρήση κάποιου από τους προρρυθμιστές και μάλιστα και σε καλύτερο χρόνο από την άμεση μέθοδο ακόμη και για τις δύο πρώτες περιπτώσεις. Αξιοσημείωτο είναι πως στην περίπτωση του κυκλώματος ac2, ακόμη και ο χειρότερος χρόνος της επαναληπτικής μεθόδου που προκύπτει με χρήση του απλού διαγώνιου προρρυθμιστή, είναι συγκρίσιμος με αυτόν της άμεσης μεθόδου. Ωστόσο, δεν πρέπει να παραβλέψουμε το γεγονός ότι και η επαναληπτική μέθοδος με χρήση κάποιου από τους προρρυθμιστές πολλές φορές αποτυγχάνει να επιλύσει το προκύπτον γραμμικό σύστημα σε αρκετές περιπτώσεις κυκλωμάτων.

Συμπερασματικά, μπορούμε να πούμε πως οι άμεσες μέθοδοι είναι προτιμότερες σε σχέση με τις επαναληπτικές για περιπτώσεις επίλυσης συστημάτων με πλήθος στοιχείων της τάξης μέχρι και μερικών εκατοντάδων χιλιάδων, με σημαντικό προβάδισμα για την επίλυση συστημάτων της τάξης μερικών χιλιάδων. Από την άλλη, οι επαναληπτικές μέθοδοι είναι σαφώς πιο αποδοτικές για την επίλυση μεγαλύτερων συστημάτων της τάξης εκατομμυρίων. Ωστόσο, απομένει πολύ δουλειά να γίνει ώστε να βελτιώσουμε την ευρωστία των τελευταίων.

Συμπεράσματα

Ο βασικός στόχος αυτής της εργασίας ήταν να δώσουμε μία εικόνα για την καλύτερη κατανόηση της συμπεριφοράς των τεχνικών προρρύθμισης όταν χρησιμοποιούνται στην επαναληπτική μέθοδο Συζυγών Κλίσεων για την επίλυση μιγαδικών αραιών γραμμικών συστημάτων. Ειδικότερα, μελετήσαμε την περίπτωση τεσσάρων προρρυθμιστών που βασίζονται στην ILU παραγοντοποίηση και τους εφαρμόσαμε για επίλυση συστημάτων που προκύπτουν κατά την προσομοίωση δικτύων διανομής ισχύος ολοκληρωμένων κυκλωμάτων.

Τα αριθμητικά αποτελέσματα έδειξαν πως δεν μπορούμε να συμπεράνουμε με ασφάλεια την βέλτιστη επιλογή ενός από τους τέσσερις προρρυθμιστές που μελετήσαμε. Γενικά, μπορούμε να πούμε ότι ενδείκνυται η χρήση του ARMS για την κατασκευή ενός ισχυρού προρρυθμιστή και για την σχετικά ομαλή και γρήγορη σύγκλιση της επαναληπτικής μεθόδου. Όμως, η χρήση του ILUT είναι πιο συμφέρουσα από άποψη χρόνου για την επίλυση σχετικά εύκολων προβλημάτων, αφού η κατασκευή του αντίστοιχου προρρυθμιστή είναι πιο φθηνή. Μπορούμε, επίσης, να χρησιμοποιήσουμε την ILUT ως ένα μέσο για να εντοπίσουμε πιο δύσκολα προβλήματα ή προβλήματα που χρειάζονται ειδική μεταχείριση, όπως για παράδειγμα η ύπαρξη μηδενικών στην κύρια διαγώνιο του πίνακα συντελεστών, έτσι ώστε να χρησιμοποιήσουμε στην συνέχεια ένα πιο ισχυρό προρρυθμιστή όπως τον ARMS.

ΠΑΡΑΡΤΗΜΑ (Κώδικας Υλοποιήσεων)

Α .Υλοποίηση Διαμόρφωσης του MNA Συστήματος

```
#ifndef ACMTHDS_H
#define ACMTHDS_H

#include "cs.h"

int nz;    // nonzeros count
void AC_calculateAbSparse(int N, int K, double omega);

#endif
```

Αρχείο 1: acmthds.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include "cs.h"
#include "parser.h"
#include "aclinsys.h"
#include "acmthds.h"
#include "ac.h"

/*This function calculates the sparse matrix A in compressed column avd row form and the vec-
tor b of the A*x=b linear system (for AC analysis).
@param N = The count of the nodes used by the Modified Nodal Analysis.
@param K= The count of the voltage sources.
@param omega = Current ω frequency.
@on return As_col = The coefficient matrix in CSC form.
@on return As_row = The coefficient matrix in CRS form.
@on return b = The right-hand-side vector.*/

void AC_calculateAbSparse(int N, int K, double omega){
  cs_complex_t g;                    // the admittance g of a resistor or an inductor or a caparcitor
  cs_complex_t value = 0.0 + 0.0*I;          // the complex value of a source
  int pos, neg;                      //the possitive and negative terminal respectively
  struct element_t *elem;
  cs_ci  *As_triplet;
  int k_ctr=1;                       // the voltage sources' counter. Goes from 0 to K
  int i;
  int n=N-1+K;

  // check if the circuit is appropriate for Modified Nodal Analysis
  for(elem=list; elem!=NULL; elem=elem->next)
    if(elem->name[0] != 'R' && elem->name[0] != 'L' && elem->name[0] != 'C' &&
       elem->name[0] != 'V' && elem->name[0] != 'I'){
         printf("Circuit must be consisting only by passive elements in order to perform Modified
                 Nodal Analysis.\n");
         exit(1);
    }
```

```
// find the number of non-zeros
  nz = 0;
  for(elem=list; elem!=NULL; elem=elem->next){
    if((elem->name[0] == 'R')||(elem->name[0] == 'L')||(elem->name[0] == 'C')||
       (elem->name[0] == 'V')){
        if(elem->mna_terminal_a!=0 && elem->mna_terminal_b!=0) nz+=4;
        else nz+=1;
    }
  }
  nnz = nz; // number of non-zeros, used in iterative solvers
  As_triplet = cs_ci_spalloc(n, n, nz, 1, 1);

  // search the list and add every element's contribution to matrix As and vector b
  for(elem=list; elem!=NULL; elem=elem->next){
    pos = elem->mna_terminal_a;
    neg = elem->mna_terminal_b;

    if(elem->name[0] == 'R'){
      g = (1 / elem->value) + 0.0*I;
      // the contribution of the resistors to matrix As
      if(pos!=0 && neg!=0){
        cs_ci_entry(As_triplet, pos-1, neg-1, (-1.0)*g);
                cs_ci_entry(As_triplet, neg-1, pos-1, (-1.0)*g);
                cs_ci_entry(As_triplet, pos-1, pos-1, g);
                cs_ci_entry(As_triplet, neg-1, neg-1, g);
      }
      else if(neg==0) cs_ci_entry(As_triplet, pos-1, pos-1, g);
      else if(pos==0) cs_ci_entry(As_triplet, neg-1, neg-1, g);
    }
    else if(elem->name[0] == 'L'){
      g = 0.0 + ((-1.0)/(omega*elem->value))*I;
      // the contribution of the inductors to matrix As
      if(pos!=0 && neg!=0){
        cs_ci_entry(As_triplet, pos-1, neg-1, (-1.0)*g);
                cs_ci_entry(As_triplet, neg-1, pos-1, (-1.0)*g);
                cs_ci_entry(As_triplet, pos-1, pos-1, g);
                cs_ci_entry(As_triplet, neg-1, neg-1, g);
      }
      else if(neg==0) cs_ci_entry(As_triplet, pos-1, pos-1, g);
      else if(pos==0) cs_ci_entry(As_triplet, neg-1, neg-1, g);
    }
    else if(elem->name[0] == 'C'){
      g = 0.0 + (omega*elem->value)*I;
      // the contribution of the capacitors to matrix As
      if(pos!=0 && neg!=0){
        cs_ci_entry(As_triplet, pos-1, neg-1, (-1.0)*g);
                cs_ci_entry(As_triplet, neg-1, pos-1, (-1.0)*g);
                cs_ci_entry(As_triplet, pos-1, pos-1, g);
                cs_ci_entry(As_triplet, neg-1, neg-1, g);
      }
      else if(neg==0) cs_ci_entry(As_triplet, pos-1, pos-1, g);
      else if(pos==0) cs_ci_entry(As_triplet, neg-1, neg-1, g);
    }
    else if(elem->name[0] == 'V'){

      if(elem->ac_source!=NULL){
        value = polarToAlgebraic(elem->ac_source->magnitude, elem->ac_source->phase);
      }
      else if(elem->ac_source == NULL) value = 0.0 + 0.0*I;
```

```
    // the contribution of the voltage sources to matrix As
      if(neg!=0){
                  cs_ci_entry(As_triplet, neg-1, N-1+k_ctr-1, (-1.0));
                  cs_ci_entry(As_triplet, N-1+k_ctr-1, neg-1, (-1.0));
      }
      if(pos!=0){
                  cs_ci_entry(As_triplet, pos-1, N-1+k_ctr-1, 1.0);
                  cs_ci_entry(As_triplet, N-1+k_ctr-1, pos-1, 1.0);
      }
      // the contribution of the voltage sources to vector b
      c_b[N-1+k_ctr-1] = c_b[N-1+k_ctr-1] + value;
      k_ctr++;
    }
    else if(elem->name[0] == 'I'){
      if(elem->ac_source!=NULL){
        value = polarToAlgebraic(elem->ac_source->magnitude, elem->ac_source->phase);
      }
      else if(elem->ac_source == NULL) {value = 0.0 + 0.0*I;}
      // the contribution of the current sources to vector b
      if(pos!=0) c_b[pos-1] = c_b[pos-1] + ((-1.0)*value);
      if(neg!=0) c_b[neg-1] = c_b[neg-1] + value;
    }
  }//end for

  //convert from triplet format to CSC format
  As_col = cs_ci_compress(As_triplet);
  cs_ci_spfree(As_triplet);
  cs_ci_dupl(As_col);

   if(optionITER){
  //convert from CSC format to CSR format
    As_row = NULL;
    As_row = cs_ci_transpose(As_col, 1);
    if(As_row==NULL){
      printf("Error in cs_transpose\n");
      exit(1);
    }
  }
}
```

Αρχείο 2: acmthds.c

Β. Υλοποίηση Άμεσης LU Μεθόδου Επίλυσης

```
#ifndef ACDIRECT_H
#define ACDIRECT_H

#include "cs.h"

cs_ci *c_Cs;
cs_cis *c_Ss;
cs_cin *c_Ns;

void AC_calculateLUSparse();
void AC_solveASparse(int N, int K);
#endif
```

Αρχείο 3: acdirect.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include <sys/time.h>
#include "acdirect.h"
#include "cs.h"
#include "parser.h"
#include "aclinsys.h"
#include "acmthds.h"
#include "ac.h"

/*The direct LU factorization method for solving complex linear systems.*/
void AC_calculateLUSparse(){
   c_Ss = cs_ci_sqr(2, As_col, 0);
   c_Ns = cs_ci_lu(As_col, c_Ss, 1);
   cs_ci_spfree(As_col);
}
/*Solves LU x = b. (b=x solution on output).*/
void AC_solveASparse(int N, int K){
   int n = N-1+K;
   cs_ci_ipvec(c_Ns->pinv, c_b, c_x, n);
   cs_ci_lsolve(c_Ns->L, c_x);
   cs_ci_usolve(c_Ns->U, c_x);
   cs_ci_ipvec(c_Ss->q, c_x, c_b, n);
}
```

Αρχείο 4: acdirect.c

Γ. Υλοποίηση Προρρυθμισμένων Επαναληπτικών Μεθόδων Επίλυσης

```
#ifndef AC_H
#define AC_H

#include "cs.h"
#include "zdefs.h"
#include "zheads.h"
#include "zprotos.h"

#define EPS (1e-14)

int nnz;                    //number of nonzeros
cs_complex_t *c_x;          // the solution vector
cs_complex_t *c_b;          // vector b of linear system A*x=b
cs_ci *As_col;              // sparse matrix A in compressed column form
cs_ci *As_row;              // sparse matrix A in compressed row form
cs_complex_t *r;
cs_complex_t *z;
cs_complex_t *p;
cs_complex_t *q;
cs_complex_t *res;
cs_complex_t *r_tilde;
cs_complex_t *z_tilde;
cs_complex_t *p_tilde;
cs_complex_t *q_tilde;
cs_complex_t *M;            //the complex jacobi preconditioner matrix
//for ZITSOL's methods
csptr A_prec;      // sparse matrix A for preconditioner methods
iluptr M_new;      // the ILUK preconditioner matrix
ilutptr M_new2;  // the ILUT preconditioner matrix
arms M_new3;    // the ARMS preconditioner matrix
csptr A_prec_T;             // the transpose ILUK preconditioner matrix
iluptr M_new_T;             // the transpose ILUT preconditioner matrix
ilutptr M_new2_T;           // the transpose ILUT preconditioner matrix
arms M_new3_T;              // the transpose ARMS preconditioner matrix
cs_complex_t *y;
cs_complex_t *y_tilde;
int lofM;
int *lfill;
double *droptol;
int *iwork, *uwork; int nB;
double tolind;
int *ipar;
double permtol;
int mband;

int AC_methodCGSparse(int N, int K, double itol);
int AC_methodBiCGSparse(int N, int K, double itol);
#endif
```

Αρχείο 5: ac.h

Η επαναληπτική μέθοδος CG υλοποιείται από την συνάρτηση AC_methodCGSparse, η οποία βρίσκεται στο αρχείο **ac.c**. Για καλύτερη κατανόηση θα δώσουμε τα κομμάτια κώδικα που κατασκευάζουν και επιλύουν τον εκάστοτε προρρυθμιστή ξεχωριστά από το υπόλοιπο κώδικα που υλοποιεί την CG μέθοδο.

```
/* The iterative preconditioned Conjugate Gradient method for solving complex linear systems.*/
int AC_methodCGSparse(int N, int K, double itol){
  int i, j, iter, temp;
  cs_complex_t rho, rho1, alpha, beta;
  double normB;
  int n=N-1+K;
  init_ac(n);
#ifndef DIAG
  //form A matrix in zSparMat struct type
  A_prec = (zSparMat*)malloc(sizeof(zSparMat));
  if(A_prec==NULL){
    printf("malloc of A_prec failed\n");
    exit(0);
  }
  if(zsetupCS(A_prec, As_row->n)){
    printf("setupCS error\n");
    exit(0);
  }
  for(i=0; i<As_row->n; i++){
    A_prec->nzcount[i] = As_row->p[i+1] - As_row->p[i];
    A_prec->ja[i] = (int*)malloc((A_prec->nzcount[i])*sizeof(int));
    A_prec->ma[i] = (complex double*)malloc((A_prec->nzcount[i])*sizeof(complex double));
    for(j=0; j<A_prec->nzcount[i]; j++){
      A_prec->ja[i][j] = As_row->i[(As_row->p[i])+j];
      A_prec->ma[i][j] = As_row->x[(As_row->p[i])+j];
    }
  }
#endif
  /*Form the preconditioner ILU preconditioner matrix.
    See code below "μέρος 2".*/

  //res=A*x
  complexMultMatrixVectorSparse(As_col, c_x, res, n);
  //r=b-A*x
  for(i=0; i<n; i++) r[i] = c_b[i]-res[i];
  iter=0;
  normB=complexNormVector(r, n);
  if(normB==0.0) normB=1.0;
  while((complexNormVector(r, n)/normB>itol) && iter<N){
    iter=iter+1;
    /*Solve M z = r.
      See code below "μέρος 3".*/

    rho=complexMultVectorVector(r, z, n); //inner product
    if(!(cabs(rho)-EPS > 0.0)){
      printf("\nBi-CG algorithm failure due to zero rho.\n");
      exit(0);
    }
    if(iter==1){
      for(i=0;i<n;i++) p[i] = z[i];
    }
    else{
      beta=rho/rho1;
      for(i=0; i<n; i++) p[i] = z[i] + beta * p[i];
    }
    rho1=rho;
    //q=A*p
    complexMultMatrixVectorSparse(As_col, p, q, n);
    alpha = rho/complexMultVectorVector(p, q, n);
```

```
    for(i=0;i<n;i++){
      c_x[i] = c_x[i] + alpha * p[i];
      r[i] = r[i] - alpha * q[i];
    }
#ifdef PRINT_ITERS
    printf("  iter:%5d", iter); fflush(stdout); printf("\b\b\b\b\b\b\b\b\b\b\b");
#endif
    }
  free_ac();
  return iter;
}
```

Αρχείο 6: ac.c (μέρος 1)

Κατασκευή Προρρυθμιστών για την CG

- Για την κατασκευή του ILUK Προρρυθμιστή έχουμε:

```
#ifdef ILUK
  M_new = (zILUSpar*)malloc(sizeof(zILUSpar));
  if(M_new==NULL){
    printf("malloc of M_new failed\n");
    exit(0);
  }
  lofM = 5;   //level of fill

  temp = zilukC(lofM, A_prec, M_new, stdout);
  printf("RESULT OF ILUK: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 2α)

- Για την κατασκευή του ILUT Προρρυθμιστή έχουμε:

```
#if defined (ILUTD) || defined (ILUTP)
  M_new2 = (zIluSpar*)malloc(sizeof(zIluSpar));
  if(M_new2==NULL){
    printf("malloc of M_new failed\n");
    exit(0);
  }
  if(zsetupILUT(M_new2, n)){
    printf("setupILUT error\n");
    exit(0);
  }
  M_new2->perm = NULL;
  M_new2->rperm = NULL;
  M_new2->D1 = NULL;
  M_new2->D2 = NULL;
  M_new2->perm2 = NULL;
#endif
#ifdef ILUTD
  lfill[5]= lfill[6] = 20;              //amount of fill-in kept in L, U
  droptol[5] = droptol[6] = 0.01; //threshold for dropping  in L, U

  temp = zilutD(A_prec, droptol, lfill, M_new2);
  printf("RESULT OF ILUTD: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 2β)

- Για την κατασκευή του ILUTP Προρρυθμιστή έχουμε:

```
#if defined (ILUTD) || defined (ILUTP)
  M_new2 = (zIluSpar*)malloc(sizeof(zIluSpar));
  if(M_new2==NULL){
     printf("malloc of M_new failed\n");
     exit(0);
  }
  if(zsetupILUT(M_new2, n)){
     printf("setupILUT error\n");
     exit(0);
  }
  M_new2->perm = NULL;
  M_new2->rperm = NULL;
  M_new2->D1 = NULL;
  M_new2->D2 = NULL;
  M_new2->perm2 = NULL;
#endif
#ifdef ILUTP
  M_new2->D1 = (double *) Malloc(n*sizeof(double), "ilutp:iluschD1" );
  temp=zroscalC(A_prec, M_new2->D1, 1);          //row scaling
  if (temp) printf("ERROR in roscalC - one row is zero\n");

  M_new2->D2 = (double *) Malloc(n*sizeof(double), "ilutp:iluschD1" );
  temp=zcoscalC(A_prec, M_new2->D2, 1);          //column scaling
  if (temp) printf("ERROR in coscalC - one column is a zero\n");

  iwork = (int *) Malloc(n*sizeof(int), "ilutp:3" );
  uwork = (int *) Malloc(n*sizeof(int), "ilutp:3.5" );
  zPQperm(A_prec, n, uwork, iwork, &nB, tolind) ;  //permutations
  zrpermC(A_prec,uwork);
  zcpermC(A_prec,iwork);
  M_new2->rperm = uwork;
  M_new2->perm  = iwork;

  M_new2->perm2 = (int*)malloc(n*sizeof(int)); //for ilutp column pivoting
  if(M_new2->perm2==NULL){
     printf("malloc of M_new->perm2 failed\n");
     exit(0);
  }
  for (j=0; j<A_prec->n; j++) M_new2->perm2[j] = j;

  lfill[5]= lfill[6] = 20;              //amount of fill-in kept in L, U
  droptol[5] = droptol[6] = 0.01; //threshold for dropping  in L, U
  permtol = 0.99;                       //tolerance ratio
  mband = n;                            // pivot is searched in whole column

  temp = zilutpC(A_prec, droptol, lfill, permtol , mband, M_new2);
  printf("RESULT OF ILUTP: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 2γ)

Σημειώνουμε πως για την κατασκευή των προρρυθμιστών ILUT και ILUTP το αρχικό κομμάτι κώδικα που περικλείεται στον έλεγχο #if defined (ILUTD) || defined (ILUTP) είναι το ίδιο και στις δύο περιπτώσεις.

- Για την κατασκευή του ARMS Προρρυθμιστή έχουμε:

```
#ifdef CARMS
  M_new3 = (zarmsMat*)malloc(sizeof(zarmsMat));
  zsetup_arms(M_new3);
  ipar[0] = 20;                    //levels
  ipar[1] = 1;                     //ddPQ
  ipar[2] = 600;                   //last schur size
  ipar[3] = 1;                     //print statistics
  ipar[12] = ipar[13] = 1;         //scaling at intermediate levels
  ipar[14] = 1;                    //permutations at last level
  ipar[15] = 1;                     //ILUTP in last level
  ipar[16] = ipar[17] = 1;         //scaling at last level
  tolind = 0.7;                    //tolerance for excluding a row from B-block
  lfill[0] = lfill[1] = 20;        //amount of fill-in kept in L, U [B]
  lfill[2] = lfill[3] = 20;        //amount of fill-in kept in E L\inv, U \inv F
  lfill[4] = 20;                   //amount of fill-in kept  in S
  lfill[5]= lfill[6] = 20;         //amount of fill-in kept  in L_S, U_S
  droptol[0] = droptol[1] = 0.001;   //threshold for dropping  in L, U [B]
  droptol[2] = droptol[3] = 0.0001;  //threshold for dropping  in L^{-1} F, E U^{-1}
  droptol[4] = 0.001;              //threshold for dropping  in Schur complement
  droptol[5] = droptol[6] = 0.01; //threshold for dropping  in L, U in last block
  for(i=0; i<7; i++){
    lfill[i] = lfill[i] * ((int)(nnz/n));
    if (i<4) droptol[i] = droptol[i] *1.6;
    else droptol[i] = droptol[i] * 0.004;
  }

  temp = zarms2(A_prec, ipar, droptol, lfill, tolind, M_new3, stdout);
  printf("RESULT OF ARMS: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 2δ)

- Τέλος, για την κατασκευή του απλού διαγώνιου προρρυθμιστή έχουμε:

```
#ifdef DIAG
  //form the diagonal preconditioner vector
  M = AC_mallocVector(n);
  for(i=0; i<n; i++){
    int p, found;
    found = -1;
    for(p=As_col->p[i]; p<As_col->p[i+1]; p++){
      if(As_col->i[p] == i) found = p;
    }
    if(found>=0) M[i] = 1/As_col->x[found];
    else M[i] = 1.0 + 0.0*I;
  }
#endif
```

Αρχείο 6: ac.c (μέρος 2στ)

Επίλυση Προρρυθμιστών για την CG

Για την επίλυση του συστήματος $Mz = r$ όπου M ο εκάστοτε προρρυθμιστής έχουμε:

- Για M να είναι ο ILUK προρρυθμιστής:

```
#ifdef ILUK
    zlusolC(r, z, M_new);
#endif
```

Αρχείο 6: ac.c (μέρος 3α)

- Για M να είναι ο ILUT προρρυθμιστής:

```
#ifdef ILUTD
    zLsol(M_new2->L, r, y);
    zUsol(M_new2->U, y, z);
#endif
```

Αρχείο 6: ac.c (μέρος 3β)

- Για M να είναι ο ILUTP προρρυθμιστής:

```
#ifdef ILUTP
    for(i=0; i<n; i++) y[i] = r[i];
    zSchLsol(M_new2, y);
    zSchUsol(M_new2, y);
    for(i=0; i<n; i++) z[i] = y[i];
#endif
```

Αρχείο 6: ac.c (μέρος 3γ)

- Για M να είναι ο ILUTP προρρυθμιστής:

```
#ifdef CARMS
    for(i=0; i<n; i++) y[i] = r[i];
    zarmsol2(y, M_new3);
    for(i=0; i<n; i++) z[i] = y[i];
#endif
```

Αρχείο 6: ac.c (μέρος 3δ)

- Για M να είναι ο απλός προρρυθμιστής Jacobi:

```
#ifdef DIAG
    AC_preconditionerSolve(M, r, z, n);
#endif
```

Αρχείο 6: ac.c (μέρος 3στ)

Η επαναληπτική μέθοδος Bi-CG υλοποιείται από την συνάρτηση AC_methodBiCGSparse, η οποία επίσης βρίσκεται στο αρχείο **ac.c**. Για ευκολότερη ανάγνωση του κώδικα θα δώσουμε τα κομμάτια κώδικα που κατασκευάζουν και επιλύουν τον εκάστοτε προρρυθμιστή ξεχωριστά από το υπόλοιπο κώδικα που υλοποιεί την Bi-CG μέθοδο.

```
/*The iterative preconditionedBi-Conjugate Gradient method for solving complex linear sys-
tems.*/
int AC_methodBiCGSparse(int N, int K, double itol){
  int i, j, iter , temp;
  cs_complex_t rho, rho1, alpha, beta, omega;
  double normB;
  int n=N-1+K;

  init_ac(n);

#ifndef DIAG
  //form A matrix in SparMat format
  A_prec = (zSparMat*)malloc(sizeof(zSparMat));
```

```
  if(A_prec==NULL){
    printf("malloc of A_prec failed\n");
    exit(0);
  }
  if(zsetupCS(A_prec, As_col->n)){
    printf("setupCS error\n");
    exit(0);
  }
  for(i=0; i<As_col->n; i++){
    A_prec->nzcount[i] = As_col->p[i+1] - As_col->p[i];
    A_prec->ja[i] = (int*)malloc((A_prec->nzcount[i])*sizeof(int));
    A_prec->ma[i] = (cs_complex_t*)malloc((A_prec->nzcount[i])*sizeof(cs_complex_t));
    for(j=0; j<A_prec->nzcount[i]; j++){
      A_prec->ja[i][j] = As_col->i[(As_col->p[i])+j];
      A_prec->ma[i][j] = As_col->x[(As_col->p[i])+j];
    }
  }
  //find the transpose of A matrix stored in SparMat format.
  A_prec_T = (zSparMat*)malloc(sizeof(zSparMat));
  if(A_prec_T==NULL){
    printf("malloc of A_prec_T failed\n");
    exit(0);
  }
  if(zsetupCS(A_prec_T, n)){
    printf("setupCS error\n");
    exit(0);
  }
  temp= zSparTran(A_prec, A_prec_T, 1, 0);
  printf("\nresult of sparTran: %d\n", temp); fflush(stdout);
#endif

  /*Form the preconditioner ILU preconditioner matrix and find its transpose.
    See code below "μέρος 5".*/

  //res=A*x
  complexMultMatrixVectorSparse(As_col, c_x, res, n);

  //r=r_tilde=b-A*x
  for(i=0; i<n; i++) r[i] = r_tilde[i] = c_b[i]-res[i];
  iter=0;
  normB=complexNormVector(r, n);
  if(normB==0.0) normB=1.0;

  while((complexNormVector(r, n)/normB>itol) && iter<n){
    iter=iter+1;

  /*Solve M z = r.
    Solve M_transpose z~ =r~
    See code below "μέρος 6".*/

    rho=complexMultVectorVector(z, r_tilde, n); //inner product
    if(!(cabs(rho)-EPS > 0.0)){
      printf("\nBi-CG algorithm failure due to zero rho.\n");
      exit(0);
    }
    if(iter==1){
      for(i=0;i<n;i++){
        p[i]=z[i];
        p_tilde[i] = z_tilde[i];
```

```
      }
    }
    else{
      beta=rho/rho1;
      for(i=0;i<n;i++){
        p[i] = z[i] + beta * p[i];
        p_tilde[i] = z_tilde[i] + beta * p_tilde[i];
      }
    }
    rho1=rho;

    //q=A*p
    complexMultMatrixVectorSparse(As_col, p, q, n);
    complexMultMatrixVectorTransSparse(As_col, p_tilde, q_tilde, n);
    omega = complexMultVectorVector(p_tilde, q, n);
    if(!(cabs(omega)-EPS > 0.0)){
      printf("\nBi-CG algorithm failure due to zero omega.\n");
      exit(0);
    }
    alpha=rho/omega;
    for(i=0;i<n;i++){
      c_x[i] = c_x[i] + alpha * p[i];
      r[i] = r[i] - alpha * q[i];
      r_tilde[i] = r_tilde[i] - alpha * q_tilde[i];
    }
#ifdef PRINT_ITERS
    printf("  iter:%5d", iter); fflush(stdout); printf("\b\b\b\b\b\b\b\b\b\b\b");
#endif
  }
  free_ac();
  return iter;
}
```

Αρχείο 6: ac.c (μέρος 4)

Κατασκευή Προρρυθμιστών για την Bi-CG

- Για την κατασκευή του ILUK Προρρυθμιστή έχουμε:

```
#ifdef ILUK
  M_new = (zILUSpar*)malloc(sizeof(zILUSpar));
  if(M_new==NULL){
    printf("malloc of M_new failed\n");
    exit(0);
  }
  M_new_T = (zILUSpar*)malloc(sizeof(zILUSpar));
  if(M_new_T==NULL){
    printf("malloc of M_new_T failed\n");
    exit(0);
  }
  //find preconditioner M_new
  temp = zilukC(5, A_prec, M_new, stdout);
  printf("RESULT OF ILUK: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
  //find transpose preconditioner M_new_T
  temp = zilukC(5, A_prec_T, M_new_T, stdout);
  printf("RESULT OF ILUK (with transpose A): %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 5α)

- Για την κατασκευή του ILUT Προρρυθμιστή έχουμε:

```
#if defined (ILUTD) || defined (ILUTP)
  M_new2 = (zIluSpar*)malloc(sizeof(zIluSpar));
  if(M_new2==NULL){
     printf("malloc of M_new failed\n");
     exit(0);
  }
  if(zsetupILUT(M_new2, n)){
     printf("setupILUT error\n");
     exit(0);
  }
  M_new2->perm = NULL;
  M_new2->rperm = NULL;
  M_new2->D1 = NULL;
  M_new2->D2 = NULL;
  M_new2->perm2 = NULL;
  M_new2_T= (zIluSpar*)malloc(sizeof(zIluSpar));
  if(M_new2_T==NULL){
     printf("malloc of M_new failed\n");
     exit(0);
  }
  if(zsetupILUT(M_new2_T, n)){
     printf("setupILUT error\n");
     exit(0);
  }
  M_new2_T->perm = NULL;
  M_new2_T->rperm = NULL;
  M_new2_T->D1 = NULL;
  M_new2_T->D2 = NULL;
  M_new_T->wk = NULL;
#endif
#ifdef ILUTD
  lfill[5]= lfill[6] = 20;              //amount of fill-in kept in L, U
  droptol[5] = droptol[6] = 0.01; //threshold for dropping in L, U

  //find preconditioner M_new2
  temp = zilutD(A_prec, droptol, lfill, M_new2);
  printf("RESULT OF ILUTD: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }

  //find transpose preconditioner M_new2_T
  temp = zilutD(A_prec_T, droptol, lfill, M_new2_T);
  printf("RESULT OF ILUTD (with transpose A): %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 5β)

- Για την κατασκευή του ILUTP Προρρυθμιστή έχουμε:

```
#if defined (ILUTD) || defined (ILUTP)
  M_new2 = (zIluSpar*)malloc(sizeof(zIluSpar));
  if(M_new2==NULL){
     printf("malloc of M_new failed\n");
     exit(0);
  }
  if(zsetupILUT(M_new2, n)){
     printf("setupILUT error\n");
     exit(0);
```

```
  }
  M_new2->perm = NULL;
  M_new2->rperm = NULL;
  M_new2->D1 = NULL;
  M_new2->D2 = NULL;
  M_new2->perm2 = NULL;

  M_new2_T= (zIluSpar*)malloc(sizeof(zIluSpar));
  if(M_new2_T==NULL){
    printf("malloc of M_new failed\n");
    exit(0);
  }
  if(zsetupILUT(M_new2_T, n)){
    printf("setupILUT error\n");
    exit(0);
  }
  M_new2_T->perm = NULL;
  M_new2_T->rperm = NULL;
  M_new2_T->D1 = NULL;
  M_new2_T->D2 = NULL;
  M_new_T->perm2 = NULL;
#endif
#ifdef ILUTP
  M_new2->D1 = (double *) Malloc(n*sizeof(double), "ilutp:iluschD1" );
  temp=zroscalC(A_prec, M_new2->D1, 1);              //row diagonal scaling
  if (temp) printf("ERROR in roscalC - one row is zero\n");

  M_new2->D2 = (double *) Malloc(n*sizeof(double), "ilutp:iluschD1" );
  temp=zcoscalC(A_prec, M_new2->D2, 1);           //column diagonal scaling
  if (temp) printf("ERROR in coscalC - one column is a zero\n");

  iwork = (int *) Malloc(n*sizeof(int), "ilutp:3" );
  uwork = (int *) Malloc(n*sizeof(int), "ilutp:3.5" );
  zPQperm(A_prec, n, uwork, iwork, &nB, tolind) ; //permutations
  zrpermC(A_prec,uwork);
  zcpermC(A_prec,iwork);
  M_new2->rperm = uwork;
  M_new2->perm  = iwork;
  M_new2->perm2 = (int*)malloc(n*sizeof(int));    //for ilutp column pivoting
  if(M_new2->perm2==NULL){
    printf("malloc of M_new2->perm2 failed\n");
    exit(0);
  }
  for (j=0; j<A_prec->n; j++) M_new2->perm2[j] = j;

  M_new2_T->D1 = (double *) Malloc(n*sizeof(double), "ilutp:iluschD1" );
  temp=zroscalC(A_prec, M_new2_T->D1, 1);
  if (temp) printf("ERROR in roscalC - one row is zero\n");

  M_new2_T->D2 = (double *) Malloc(n*sizeof(double), "ilutp:iluschD1" );
  temp=zcoscalC(A_prec, M_new2_T->D2, 1);
  if (temp) printf("ERROR in coscalC - one column is a zero\n");
  iwork = (int *) Malloc(n*sizeof(int), "ilutp:3" );
  uwork = (int *) Malloc(n*sizeof(int), "ilutp:3.5" );
  zPQperm(A_prec, n, uwork, iwork, &nB, tolind) ;
  zrpermC(A_prec,uwork);
  zcpermC(A_prec,iwork);
  M_new2_T->rperm = uwork;
  M_new2_T->perm  = iwork;
```

```
  M_new2_T->perm2 = (int*)malloc(n*sizeof(int));
  if(M_new2_T->perm2==NULL){
     printf("malloc of M_new2_T->perm2 failed\n");
     exit(1);
  }
  for (j=0; j<A_prec_T->n; j++) M_new2_T->perm2[j] = j;

  lfill[5]= lfill[6] = 20;              //amount of fill-in kept in L, U
  droptol[5] = droptol[6] = 0.01; //threshold for dropping in L, U
  permtol = 0.99;                              //tolerance ratio
  mband = n;                            // pivot is searched in whole column
 for(i=0; i<7; i++){
     lfill[i] = lfill[i] * ((int)(nnz/n));
     if (i<4) droptol[i] = droptol[i] *1.6;
     else droptol[i] = droptol[i] * 0.004;
 }

 //find preconditioner M_new2
  temp = zilutpC(A_prec, droptol, lfill, permtol, mband, M_new2);
  printf("RESULT OF ILUTP: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }

 //find transpose preconditioner M_new2_T
  temp = zilutpC(A_prec_T, droptol, lfill, permtol, mband, M_new2_T);
  printf("RESULT OF ILUTP (with transpose A): %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 5γ)

Σημειώνουμε πως, όπως και στην υλοποίηση της CG μεθόδου, για την κατασκευή των προρρυθμιστών ILUT και ILUTP το αρχικό κομμάτι κώδικα που περικλείεται στον έλεγχο `#if defined (ILUTD) || defined (ILUTP)` είναι το ίδιο και στις δύο περιπτώσεις.

- Για την κατασκευή του ARMS Προρρυθμιστή έχουμε:

```
#ifdef CARMS
  M_new3 = (zarmsMat*)malloc(sizeof(zarmsMat));
  zsetup_arms(M_new3);
  M_new3_T = (zarmsMat*)malloc(sizeof(zarmsMat));
  zsetup_arms(M_new3_T);
  ipar[0] = 20;                         //levels
  ipar[1] = 1;                          //ddPQ
  ipar[2] = 600;                        //last schur size
  ipar[3] = 1;                          //print statistics
  ipar[12] = ipar[13] = 1;              //scaling at intermediate levels
  ipar[14] = 1;                         //permutations at last level
  ipar[15] = 1;                         //ILUTP in last level
  ipar[16] = ipar[17] = 1;              //scaling at last level
  tolind = 0.7;                         //tolerance for excluding a row from B-block
  lfill[0] = lfill[1] = 20;             //amount of fill-in kept in L, U [B]
  lfill[2] = lfill[3] = 20;             //amount of fill-in kept in E L\inv, U \inv F
  lfill[4] = 20;                        //amount of fill-in kept  in S
  lfill[5]= lfill[6] = 20;              //amount of fill-in kept  in L_S, U_S
  droptol[0] = droptol[1] = 0.001;   //threshold for dropping  in L, U [B]
  droptol[2] = droptol[3] = 0.0001;  //threshold for dropping  in L^{-1} F, E U^{-1}
  droptol[4] = 0.001;                   //threshold for dropping  in Schur complement
  droptol[5] = droptol[6] = 0.01; //threshold for dropping  in L, U in last block
```

```
  for(i=0; i<7; i++){
    lfill[i] = lfill[i] * ((int)(nnz/n));
    if (i<4) droptol[i] = droptol[i] *1.6;
    else droptol[i] = droptol[i] * 0.004;
  }

  //find preconditioner M_new3
  temp = zarms2(A_prec, ipar, droptol, lfill, tolind, M_new3, stdout);
  printf("RESULT OF ARMS: %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }

  //find transpose preconditioner M_new3_T
  temp = zarms2(A_prec_T, ipar, droptol, lfill, tolind, M_new3_T, stdout);
  printf("RESULT OF ARMS (with transpose A): %d\n", temp); fflush(stdout);
  if(temp!=0) { exit(0); }
#endif
```

Αρχείο 6: ac.c (μέρος 5δ)

- Τέλος, για την κατασκευή του απλού διαγώνιου προρρυθμιστή έχουμε:

```
#ifdef DIAG
  //form the diagonal preconditioner matrix
  M = AC_mallocVector(n);
  for(i=0; i<n; i++){
    int p, found;
    found = -1;
    for(p=As_col->p[i]; p<As_col->p[i+1]; p++){
      if(As_col->i[p] == i) found = p;
      }
    if(found>=0) M[i] = 1/As_col->x[found];
    else { M[i] = 1.0 + 0.0*I; }
  }
#endif
```

Αρχείο 6: ac.c (μέρος 5στ)

Επισημαίνουμε πως σε αυτή την περίπτωση δεν χρειάζεται να βρούμε ξεχωριστά τον ανάστροφο προρρυθμιστή M^T αφού αυτός συμπίπτει με τον προρρυθμιστή M. Είναι προφανές πως αυτό ισχύει αφού τα διαγώνια στοιχεία ενός τετραγωνικού πίνακα και του ανάστροφου του, είναι ακριβώς τα ίδια.

Επίλυση Προρρυθμιστών για την Bi-CG

Για την επίλυση των συστημάτων $M\,z = r$ και $M^T\,\underline{\tilde{z}} = \underline{\tilde{r}}$, όπου M ο εκάστοτε προρρυθμιστής και M^T ο ανάστροφός του, έχουμε:

- Για M να είναι ο ILUK προρρυθμιστής και M^T ο ανάστροφός του:

```
#ifdef ILUK
    zlusolC(r, z, M_new);
    zlusolC(r_tilde, z_tilde, M_new_T);
#endif
```

Αρχείο 6: ac.c (μέρος 6α)

- Για M να είναι ο ILUT προρρυθμιστής και M^T ο ανάστροφός του:

```
#ifdef ILUTD
    zLsol(M_new2->L, r, y);
    zUsol(M_new2->U, y, z);
    zLsol(M_new2_T->L, r_tilde, y_tilde);
    zUsol(M_new2_T->U, y_tilde, z_tilde);
#endif
```

Αρχείο 6: ac.c (μέρος 6β)

- Για M να είναι ο ILUTP προρρυθμιστής και M^T ο ανάστροφός του:

```
#ifdef ILUTP
    for(i=0; i<n; i++) y[i] = r[i];
    zSchLsol(M_new2, y);
    zSchUsol(M_new2, y);
    for(i=0; i<n; i++) z[i] = y[i];

    for(i=0; i<n; i++) y_tilde[i] = r_tilde[i];
    zSchLsol(M_new2_T, y_tilde);
    zSchUsol(M_new2_T, y_tilde);
    for(i=0; i<n; i++) z_tilde[i] = y_tilde[i];
#endif
```

Αρχείο 6: ac.c (μέρος 6γ)

- Για M να είναι ο ARMS προρρυθμιστής και M^T ο ανάστροφός του:

```
#ifdef CARMS
    for(i=0; i<n; i++) y[i] = r[i];
    zarmsol2(y, M_new3);
    for(i=0; i<n; i++) z[i] = y[i];

    for(i=0; i<n; i++) y_tilde[i] = r_tilde[i];
    zarmsol2(y_tilde, M_new3_T);
    for(i=0; i<n; i++) z_tilde[i] = y_tilde[i];
#endif
```

Αρχείο 6: ac.c (μέρος 6δ)

- Τέλος, για $M = M^T$ να είναι ο απλός διαγώνιος προρρυθμιστής:

```
#ifdef DIAG
    AC_preconditionerSolve(M, r, z, n);
    AC_preconditionerSolve(M, r_tilde, z_tilde, n);
#endif
```

Αρχείο 6: ac.c (μέρος 6στ)

Δ. Υλοποίηση Βοηθητικών Συναρτήσεων

```
#ifndef ACLINSYS_H
#define ACLINSYS_H

#include "cs.h"

struct polar_complex{
   double magnitude;
   double phase; //given in degrees
};
void AC_setToZero(cs_complex_t *vector, int n);
void RsetToZero(double *vector, int n);
cs_complex_t *AC_mallocVector(int n);
double *RmallocVector(int n);
void init_ac(int n);
void free_ac();

cs_complex_t polarToAlgebraic(double magnitude, double phase);
struct polar_complex *algebraicToPolar(cs_complex_t c);
double complexAbs(cs_complex_t c);
double complexNormVector(cs_complex_t *v, int n);
void complexMultMatrixVectorSparse(cs_ci *A, cs_complex_t *x, cs_complex_t *y, int n);
void complexMultMatrixVectorTransSparse(cs_ci *A, cs_complex_t *x, cs_complex_t *y, int n);
cs_complex_t complexMultVectorVector(cs_complex_t *r, cs_complex_t *z , int n);
void AC_preconditionerSolve(cs_complex_t *M, cs_complex_t *r, cs_complex_t *z , int n);
#endif
```

Αρχείο 7: aclinsys.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include "cs.h"
#include "parser.h"
#include "aclinsys.h"
#include "acmthds.h"
#include "ac.h"
/*This function sets a n size vector to all zeros.
@param vector = vector to zero out
@param n = size of vector*/
void RsetToZero(double *vector, int n){
   int i;
   for(i=0; i<n; i++) vector[i] = 0.0;
}
/*This function sets a vector of complex numbers to all zeros.
@param vector =  complex vector to zero out
@param n = size of the complex vector*/
void AC_setToZero(cs_complex_t *vector, int n){
   int i;
   for(i=0; i<n; i++) vector[i] = 0.0 + 0.0*I;
}
/* This function allocates memory for a n size vector.
@param n = size of vector to allocate
@return = a pointer to the allocated memory.*/
double *RmallocVector(int n){
```

```
  double *vector;
  vector = (double*)malloc(n*sizeof(double));
  if(vector==NULL){
     printf("error in malloc\n");
     exit(0);
  }
  RsetToZero(vector, n);
  return vector;
}
/* This function allocates memory for a n size vector of complex numbers.
@param n = size of complex vector to allocate
@return = a pointer to the allocated memory.*/
cs_complex_t *AC_mallocVector(int n){
  cs_complex_t *vector;
  vector = (cs_complex_t*)malloc(n*sizeof(cs_complex_t));
  MALLOC_RETURN_CHECK(vector);
  AC_setToZero(vector,n);
  return vector;
}
/*Frees the previously allocated memory for iterative solvers
@param n = size of vectors to allocate*/
void free_ac(){
#ifndef DIAG
  zcleanCS(A_prec);
#endif
  cs_ci_free(r);
  cs_ci_free(z);
  cs_ci_free(p);
  cs_ci_free(q);
  cs_ci_free(res);
  cs_ci_free(y);
  free(droptol);
  free(lfill);
  free(ipar);
#ifdef DIAG
  cs_ci_free(M);
#endif
#ifdef ILUK
  zcleanILU(M_new);
#endif
#if ILUT
  zcleanILUT(M_new2, 0);
#endif
#ifdef ARMS
  zcleanARMS(M_new3);
#endif
  if(!optionSPD){
#ifndef DIAG
     zcleanCS(A_prec_T);
#endif
     cs_ci_free(r_tilde);
     cs_ci_free(z_tilde);
     cs_ci_free(p_tilde);
     cs_ci_free(q_tilde);
     cs_ci_free(y_tilde);
#ifdef ILUK
     zcleanILU(M_new_T);
#endif
#if defined (ILUTD) || defined (ILUTP)
```

```
    zcleanILUT(M_new2_T, 0);
#endif
#ifdef ARMS
    zcleanARMS(M_new3_T);
#endif
  }
}
/* Makes the apropriate initializations for iterative solvers
@param n = size of vectors to allocate*/
void init_ac(int n){
  r = (cs_complex_t*)AC_mallocVector(n);
  z = (cs_complex_t*)AC_mallocVector(n);
  p = (cs_complex_t*)AC_mallocVector(n);
  q = (cs_complex_t*)AC_mallocVector(n);
  res = (cs_complex_t*)AC_mallocVector(n);
  y = (cs_complex_t*)AC_mallocVector(n);
  lofM = 0;
  lfill = (int*)RmallocVector(7);
  droptol = (double*)RmallocVector(7);
  tolind = 0.0;
  ipar = (int*)RmallocVector(18);
  permtol = 0.0;
  mband = 0;

  if(!optionSPD){
    r_tilde  = (cs_complex_t*)AC_mallocVector(n);
    z_tilde  = (cs_complex_t*)AC_mallocVector(n);
    p_tilde  = (cs_complex_t*)AC_mallocVector(n);
    q_tilde  = (cs_complex_t*)AC_mallocVector(n);
    y_tilde  = (cs_complex_t*)AC_mallocVector(n);
  }
}
/*Converts a complex number from polar to algebraic form
@param magnitude = magnitude of complex number in polar form
@param phase = phase of complex number in polar form
@return c = complex number in algebraic form*/
cs_complex_t polarToAlgebraic(double magnitude, double phase){
  double real, imag;
  cs_complex_t c;

  real = magnitude * cos(phase*(3.14159265/180));
  imag = magnitude * sin(phase*(3.14159265/180)); //cos, sin need an angle
                                                  expressed inradians
  c = real + imag * I;
  return(c);
}
/* Convert a complex number from algebraic to polar form
@param c = complex number in algebraic form
@return res = struct that contains magnitude and phase of complex number in polar form*/
struct polar_complex *algebraicToPolar(cs_complex_t c){
  struct polar_complex *res;

  res = (struct polar_complex*)malloc(sizeof(struct plar_complex));

  res->magnitude = sqrt(pow(creal(c), 2) + pow(cimag(c), 2));
  res->phase = atan2(cimag(c),creal(c)) * (180/3.14159265);
                                  //atan2 retunrs the value   of  arctan expressed in radians
                                  //radians*(180/π) = degree
return res;
```

```
}
/* Returns a complex number's absolute value
@param c = complex number in algebraic form
@return  = absolute value of complex number*/
double complexAbs(cs_complex_t c){
  int vabs;

  vabs = pow(creal(c),2.0) + pow(cimag(c),2.0);

  return sqrt(vabs);
}
/* Returns the Euclidean norm of a complex vector
@param v = complex vector in algebraic form
@param n = size of complex vector
@return =  norm of complex vector*/
double complexNormVector(cs_complex_t *v, int n){
  int i;
  double sum= 0.0;

  for(i=0;i<n;i++) sum += pow(creal(v[i]),2.0) + pow(cimag(v[i]),2.0);
  return sqrt(sum);
}
/*Multiplies two complex vectors and returns the result (Euclidean inner product)
@param r = the one complex vector multiplicand in algebraic form
@param z = the other complex vector multiplicand in algebraic form
@param n = the size of both complex vector
@return sum = the Euclidean inner product  of the two vectors*/
cs_complex_t complexMultVectorVector(cs_complex_t *r, cs_complex_t *z, int n){
  int i;
  cs_complex_t *temp;
  cs_complex_t sum = 0.0 + 0.0*I;

  temp = (cs_complex_t*)AC_mallocVector(n);

  for(i=0; i<n; i++) {
    temp[i] = creal(z[i]) - cimag(z[i])*I;
  }
  for(i=0; i<n; i++) sum += r[i] * temp[i];

  cs_ci_free(temp);
 return sum;
}
/*Multiplies the size n*n sparse matrix A with the complex vector x and the result is stored at
complex vector y.
@param A = matrix multiplicand in CSC format of CXsparse library  struct type
@param x =  complex vector multiplicand in algebraic form
@param y =  where the result is stored
@param n = the size of complex vector
@On return  y = the result complex vector*/
void complexMultMatrixVectorSparse(cs_ci *A, cs_complex_t *x, cs_complex_t *y, int n){
  int j, p;
  for(j=0; j<n; j++) y[j] = 0.0 + 0.0*I;
  for(j=0; j<n; j++){
    for(p=A->p[j]; p<A->p[j+1]; p++){
      y[A->i[p]] = y[A->i[p]] + A->x[p] * x[j];
    }
  }
}
```

```
/*Multiplies the size n*n transposed sparse matrix A with the complex vector x and the result is
stored at complex vector y.
@param A = matrix multiplicand in CSC format of CXsparse library  struct type
@param x =  complex vector multiplicand in algebraic form
@param y =  where the result is stored
@param n = the size of complex vector
@On return  y = the result complex vector*/
void complexMultMatrixVectorTransSparse(cs_ci *A, cs_complex_t *x, cs_complex_t *y ,
                                                                                int n){
  int j, p;
  for(j=0; j<n; j++) y[j] = 0.0 + 0.0*I;
  for(j=0; j<n; j++){
    for(p=A->p[j]; p<A->p[j+1]; p++){
      y[j] = y[j] + A->x[p] * x[A->i[p]];
    }
  }
}
/*Multiplies the complex vector M with the complex vector r and the result is stored at complex
vector z
@param M = complex vector in algebraic form (diagonal preconditioner)
@param z = where result is stored
@param  n = the size of complex vector
@on return z = the solution complex vector*/
void AC_preconditionerSolve(cs_complex_t *M, cs_complex_t *r, cs_complex_t *z, int n){
  int j;
  for(j=0; j<n; j++) z[j]=M[j]*r[j];
}
```

Αρχείο 8: aclinsys.c

Βιβλιογραφία - Αναφορές

[1] "Circuit Simulation", Farid N.Najm, Wiley & Sons Publications, New Jersey, 2010.

[2] "An Introduction to the Modified Nodal Analysis", Michael Hanke, May 2006, Department of Numerical Analysis and Computer Science, Royal Institute of Technology, Sweden.

[3] Σημειώσεις μαθήματος "Προσομοίωση Κυκλωμάτων", Διδάσκων Ευμορφόπουλος Νέστωρ, Φθινόπωρο 2011, Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων, Πανεπιστήμιο Θεσσαλίας.

[4] "Direct Methods for Sparse Linear Systems", TimothyA.Davis, Society for Industrial and Applied Mathematics (SIAM), University of Florida, 2006 .

[5] "Iterative Methods for Sparse Linear Systems", Yousef Saad, SIAM, Second Edition, 2003.

[6] "Preconditiong Techniques for Large Linear Systems: A Survey", Michele Benzi, Mathematics and Computer Science Department, Emory University, Atlanta, revised July 2002.

[7] "Θέματα Ανάλυσης Πινάκων", Παναγιώτης Ι.Ψαρράκος, Απρίλιος 2012, Σχολή Εφαρμοσμένων Μαθηματικών & Φυσικών Επιστημών, Εθνικό Μετσόβιο Πολυτεχνείο.

[8] "Linear Algebra and its Applications", Gilbert Strang, Masschussetts Institude Of Technology, Harcourt Brace Jovanovich publications, Tenth Edition, 2006.

[9] "Numerical radius and zero pattern matrices", Vladimir Nikiforov, January 2007, Department of Mathematical Sciences, University of Memphia.

[10] "ILUT: A dual threshold incomplete LU factorization", Youcef Saad, Computer Science Department, University of Minnesota.

[11] "Preconditioning Techniques for nonsymmetric indefinite linear systems", Youcef Saad, December 1992, Center of Supercomputing Research and Development, University of Illinois.

[12] "Schur complement preconditioners for distributed general sparse linear systems", Youcef Saad, Department of Computer Science and Engineering, University of Minnesota.

[13] "ARMS: An Algebraic Recursive Multilevel Solver sparse linear systems", Youcef Saad and Brian Suchomel, June 1999, Department of Computer Science and Engineering, University of Minnesota.

[14] "Multilevel ILU with reordering for diagonal dominance", Youcef Saad, November 2004, Department of Computer Science and Engineering, University of Minnesota

[15] "BILUM: Block Versions of Multielimination and Multilevel ILU preconditioner for general sparse linear systems, Youcef Saad and Jun Zhang, 1999, SIAM.

[16] "BILUTM: A domain-based Multilevel Block ILUT preconditioner for general sparse matrices", Youcef Saad and Jun Zhang, 1999, SIAM.

[17] "Greedy coarsening strategies for non-symmetric problems", S.MacLachlan and Youcef Saad, May 2006, Department of Computer Science and Engineering, University of Minnesota.

[18] "A Multilevel block incomplete factorization preconditioning", Y.Notay, 1999, *Université Libre de Bruxelles*, Applied Numerical Mathematics, Elsevier.

[19] "Multilevel preconditioners constructed from inverse-based ILU", Matthias Bollhöfer and Youcef Saad, May 2004, Department of Computer Science and Engineering, University of Minnesota.

[20] "PSpice: Referance Guide", Cadence Design Systems, Second online edition, May 2000.

[21] "The designer's guide to spice and spectre", Kenneth S.Kundert, Cadence Design

Systems, Kluwer Academic Publishers, 1995.
[22] "Frequency Domain Analysis of Switching Noise on Power Supply Network", Shiyou Zhao, Kaushik Roy and ChengKok Koh, School of Electrical and Computer Engineering Purdue University.
[22] "Power Grid Analysis based on a Macro circuit model", Shahar Kvatinsky, Eby G. Friedman , Avinoam Kolodny and Levi Schächter, 2010, IEEE 26-th Convention of Electrical and Electronics Engineers, Israel.
[23] "Power Grid Analysis Benchmarks", Sani R. Nassif, IBM Research - Austin.
[24] "Numerical performance of preconditioning techniques for the solution of complex sparse linear systems", Annamaria Mazzia and Giorgio Pini, Dipartimento di Metodi e Modelli Matematici per le Scienze Applicate, Universita degli Studi di Padova, 2003, Communications in numerical methods in engineering.
[25] "A comparison of preconditioners for complex-valued matrices", Daniel Osei-Kuffuor and Yousef Saad, December 2007, Department of Computer Science and Engineering, University of Minnesota.