ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΒΑΖΑΚΑΣ ΑΝΑΣΤΑΣΙΟΣ

Αλγόριθμοι Προσομοίωσης Μεγάλης Κλίμακας Γραμμικών Κυκλωμάτων

Μάρτιος 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Αλγόριθμοι Προσομοίωσης Μεγάλης Κλίμακας Γραμμικών Κυκλωμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

ΒΑΖΑΚΑ ΑΝΑΣΤΑΣΙΟΥ

Επιβλέποντες :

ΠΡΩΤΕΥΩΝ ΕΠΙΒΛΕΠΩΝ	ΔΕΥΤΕΡΕΥΩΝ ΕΠΙΒΛΕΠΩΝ
Δρ. Νέστορας Ευμορφόπουλος	Δρ. Γεώργιος Σταμούλης
Λέκτορας καθηγητής του	Καθηγητής του
Πανεπιστημίου Θεσσαλίας	Πανεπιστημίου Θεσσαλίας

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την 01/03/2012.

ΠΡΩΤΕΥΩΝ ΕΠΙΒΛΕΠΩΝ	ΔΕΥΤΕΡΟΥΩΝ ΕΠΙΒΛΕΠΩΝ
Δρ. Νέστορας Ευμορφόπουλος	Δρ. Γεώργιος Σταμούλης
Λέκτορας καθηγήτης του	Καθηγητής του
Πανεπιστήμιου Θεσσαλίας	Πανεπιστημίου Θεσσαλίας

(Υπογραφή)

.....

Βαζάκας Αναστάσιος

στους γονείς μου

Περιεχόμενα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω από τα βάθη της καρδιάς μου μια σειρά ανθρώπων που με βοήθησαν και με υποστήριξαν με ποικίλους τρόπους κατά τη διάρκεια της εκπόνησης και της συγγραφής της παρούσας διπλωματικής εργασίας.

Πρώτα ,και κύρια θα ήθελα να ευχαριστήσω τους καθηγητές κ.Ευμορφόπουλο Νέστορα και κ.Σταμούλη Γεώργιο για την εμπιστοσύνη που μου έδειξαν με την ανάθεση του παρόντος θέματος καθώς και για τη καλή συνεργασία που είχαμε, παρά τις δύσκολες συνθήκες υπό τις οποίες εκπονήθηκε η διπλωματική.

Επίσης, θέλω να ευχαριστήσω τους συμφοιτητές μου για τις πολύτιμές συμβουλές τους και για την άριστη συνεργασία που είχαμε, στοιχεία που έπαιξαν μείζονα ρόλο στην ολοκλήρωση της παρούσας εργασίας.

Τέλος ιδιαίτερες ευχαριστίες αξίζουν και στην οικογένειά μου. Τόσο για την αγάπη και τη στήριξη που μου παρείχαν, όσο και για το λαμπρό παράδειγμα θάρρους, καρτερικότητας και αυταπάρνησης που υπήρξαν στις δύσκολες στιγμές που αντιμετωπίσαμε πρόσφατα

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με την προσομοίωση-επίλυση γραμμικών κυκλωμάτων μεγάλης κλίμακας μέσω διαφόρων μεθόδων .Τα σχετικά βιβλία πάνω στα οποία κυρίως βασίστηκε η διπλωματική εργασία είναι το Circuit Simulation του Farid N. Najm και το Direct Methods for Sparse Linear Systems του Timothy A. Davis .

Πιο συγκεκριμένα στο 1^{ο} κεφάλαιο δίνεται μία πρώτη μαθηματική περιγραφή των βασικών εννοιών της ανάλυσης κυκλωμάτων.Ειδικότερα αναπτύσσεται η ανάλυση της τροποιημένης μεθόδου των κόμβων ,τόσο σε απλά γραμμικά δίκτυα όσο και σε δίκτυα τροφοδοσίας μεγάλης κλίμακας.,Μέσα από αυτή την ανάλυση δημιουργείται ένα σύστημα γραμμικών εξισώσεων στο οποίο μέσω συγκεκριμένων μεθόδων γραμμικής άλγεβρας γίνεται η επίλυση του.Πιο συγκεκριμένα στο ίδιο κεφάλαιο περιγράφονται με αναλυτικό τρόπο τόσο οί άμμεσοι μέθοδοι επίλυσης, όπως είναι η LU παραγοντοποληση, η μέθοδος του Gauss, και η μέθοδος Cholesky.Όσο και επαναληπτικοί μέθοδοι οι οποίοι επιτυγχάνουν την επίλυση του συστήματος με προσεγγιστικό τρόπο,αυτές οι μέθοδοι είναι η μέθοδος συζυγών κλίσεων και η μέθοδος των συζυγών κλίσεων με προρρυθμιστή στο ίδιο κεφάλαιο περιγράφονται και τα θέματα σύγκλισης αυτών των μεθόδων.

Στό δεύτερο (2^{ο}) κεφάλαιο περιγράφεται η δημιουργία και αναπαράσταση αυτής της μεθόδου μέσω συγκεκριμένων παραδειγμάτων.Αναλυτικότερα παρουσιάζεται η μεθοδολογία και τα βημάτα τα οποία ακολουθούνται για την ορθή αναπαράσταση του συστήματος .Επιπλέον στο ίδιο κεφάλαιο πραγματοποιείται και η εμπειρική μελέτη του θέματος μέσα από την προσομοίωση συγκεκριμένων κυκλωμάτων μέσω της εξέτασης του χρόνου υπολογισμού των λύσεων του συστήματος.Στο 3^{ο} και 4^{ο} κεφάλαιο πραγματοποιείται μία μεγάλη αναφορά στην μέθοδο των αραίων πινάκων και πώς μπορεί να γίνει η επιτυχή αναπαράσταση-επίλυση ηλεκτρικών κυκλωμάτων μέσω πολύ συγκεκριμένων συναρτήσεων και μεθόδων οι οποίες περιγράφονται με πολύ χαρακτηριστικό τρόπο στο 3^{ο} κεφάλαιο.

Στο 4^{ο} κεφάλαιο γίνεται η αναλυτική περιγραφή του τρόπου με τον οποία μπορούμε να μετασχηματίσουμε τον πίνακα A της τροποποιημένης μεθόδου των κόμβων σε

sparse πίνακα. Αναλυτικότερα περιγράφονται τα βήματα και οι συναρτήσεις οι οποίες χρησιμοποιήθηκαν για την μετατροπή του πίνακα σε triplet form. Επιπρόσθετα στο ίδιο κεφάλαιο περιγράφονται τόσο τα αποτελέσματα της λύσης του συστήματος μέσω της συνάρτησης lusol όσο και οι χρόνοι υπολογισμού αυτών .

Στο 5^{ο} κεφάλαιο παρουσιάζονται τα δίκτυα διανομής ισχύος αρχικά περιγράφονται οι μαθηματικές αποδείξεις της τροποποιημένης μεθόδου των κόμβων για τα δίκτυα αυτά. Επίσης παρουσιάζουμε τις υλοποιήσεις σε Matlab και σε C δίνοντας σιμαντικές διευκρυνήσεις για τη λειτουργία τους.

Στο 6^{ο} κεφάλαιο βλέπουμε τα αποτελέσματα της πρακτικής εκτέλεσης σε ένα μικρό πρόβλημα. Εν συνεχεία στο 7^{ο} κεφάλαιο παρουσιάζονται τα πειραματικά αποτελέσμα του χρόνου επίλυσης μίας σειράς παραδειγμάτων με την βοήθεια τόσο του Matlab όσο και της γλώσσας C μέσω της βιβλιοθήκης time.

Τέλος στο 8^{ο} κεφάλαιο παρουσιάζουμε μία σύνοψη και τα συμπεράσματά μας από αυτή τη πολύ ενδαφέρουσα πτυχιακή εργασία.

Σε όλες τις προσπάθειες μοντελοποίησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού C, και το Matlab.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Αλγόριθμος του Gauss, Πίνακας Τροποποιημένης Μεθόδου Κόμβων ,Sparse πίνακας,Δίκτυα Δομής Πλέγματος,Μέθοδος Domain Decomposition

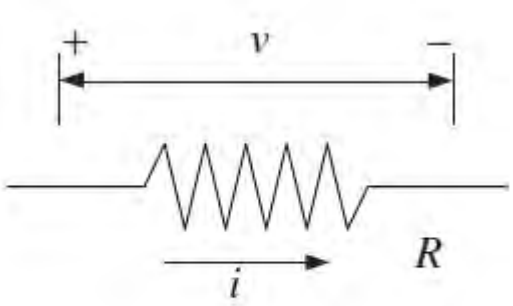
Κεφάλαιο 1

Εισαγωγή

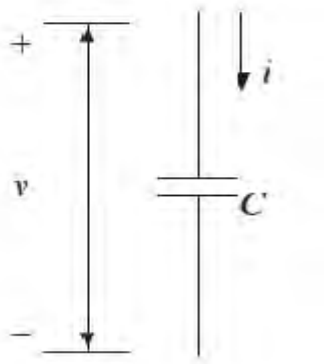
Προσομοίωση Κυκλωμάτων είναι η διαδικασία κατά την οποία ελέγχουμε και επαληθεύουμε ένα ηλεκτρονικό κύκλωμα πριν την τελική κατασκευή του. Χρησιμοποιείται σε ένα ευρύ φάσμα εφαρμογών που κυμαίνονται από τα ολοκληρωμένα κυκλώματα και την μικροηλεκτρονική ως τα δίκτυα διανομής ηλεκτρικής ενέργειας και ηλεκτρικής ισχύς. Η προσομοίωση κυκλωμάτων παραμένει ένας σημαντικός κλάδος των ολοκληρωμένων κυκλωμάτων ο οποίος συνδυάζει γνώσεις ηλεκτρονικής και προγραμματισμού .Η συγκεκριμένη εργασία καλύπτει το θεωρητικό υπόβαθρο (στα πρώτα κεφάλαια) αλλά και το πρακτικό (στα τελευταία) στην προσομοίωση των κυκλωμάτων και εστιάζει περισσότερο στα γραμμικά κυκλώματα κατά την διάρκεια της υλοποίησης. Η προσομοίωση κυκλωμάτων συνδυάζει πρώτον την μαθηματική μοντελοποίηση των ηλεκτρικών κυκλωμάτων, δεύτερον την μαθηματική αναπαράσταση αυτών μέσω γραμμικών και μη γραμμικών συστημάτων και τρίτον την επίλυση αυτών μέσω συγκεκριμένων αλγορίθμων.

1.1.Εξισώσεις βασικών στοιχείων κυκλώματος

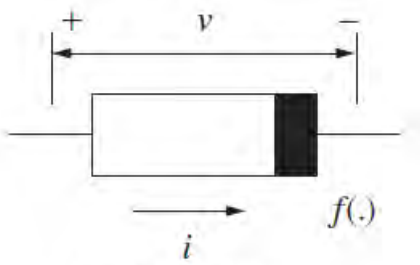
Ηλεκτρικό στοιχείο ονομάζεται οποιοδήποτε στοιχείο το οποίο μπορεί να εκφραστεί μέσα από μία μαθηματική έκφραση πηγών και τάσεων. Για παράδειγμα μία αντίσταση R(Σχ.1) μπορεί να εκφραστεί μέσα από την εξίσωση του Ohm $u = R * i$. Κατά σύμβαση θεωρούμε ότι ,όταν η μαθηματική έκφραση δεν είναι δευτέρου βαθμού η μεγαλύτερο τότε λέμε ότι το στοιχείο αυτό είναι γραμμικό. Ένα δίκτυο από γραμμικά στοιχεία θα ονομάζεται γραμμικό κύκλωμα. Η εξίσωση της αντίστασης είναι μία αλγεβρική συνάρτηση σε αντίθεση με τον πυκνωτή(Σχ.2) του οποίου η εξίσωση είναι $I = C * \frac{dv}{dt}$.Όπως βλέπουμε είναι μία διαφορική εξίσωση πρώτου βαθμού



Σχ.1:Αντίσταση



Σχ.2:Πυκνωτής



Σχ.3:Μη γραμμικό στοιχείο

Είναι πρώτου βαθμού γιατί περιέχει μόνο πρώτου βαθμού παραγώγους και είναι συνήθης διότι δεν περιέχει μερικές παραγώγους. Δεδομένου ότι δεν υπάρχει καμία δύναμη του δύο η εξίσωση αυτή είναι γραμμική .Η κατεύθυνση αναφοράς του ρεύματος βασίζεται στην κατεύθυνση αναφοράς της τάσης όπως φαίνεται στο σχήμα 2. Η αναπαράσταση ενός κυκλώματος γίνεται μέσω των στοιχείων R,L,C, έτσι ώστε αυτά τα στοιχεία να περιγράφονται από ένα σύστημα πρώτης τάξης γραμμικών διαφορικών εξισώσεων .Αντιστάσεις και πυκνωτές είναι γραμμικά στοιχεία με δύο ακροδέκτες .Σε γενικές γραμμές ένα στοιχείο με δύο τερματικά μπορεί να περιγραφεί από μία $i - v$ εξίσωση $i = f(v)$, όπου f μπορεί να είναι οποιαδήποτε συνάρτηση $f: \Re \rightarrow \Re$.Όταν f είναι μη γραμμική εξίσωση ,το στοιχείο λέγεται μη γραμμικό και ο συμβολισμός του φαίνεται στο σχήμα 3

Ιδανικές πηγές τάσεις

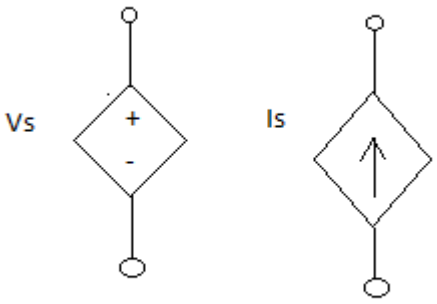
Μια ιδανική πηγή τάσης είναι μία ηλεκτρική συσκευή που παράγει προκαθορισμένη τάση στα άκρα της ανεξάρτητα από το ρεύμα που τη διαρρέει .Η ποσότητα του ρεύματος που περέχει η πηγή καθορίζεται από το κύκλωμα στο οποίο συνδεέται.

Ιδανικές πηγές ρεύματος

Μια ιδανική πηγή ρεύματος παρέχει προκαθορισμένο ρεύμα σε οποιοδήποτε κύκλωμα συνδεθεί.Η τάση που παρέχει η πηγή καθορίζεται από το κύκλωμα στο οποίο συνδέεται.

Εξαρτώμενες ή Ελεγχόμενες Πηγές

Οι πηγές που περιγράψαμε μέχρι τώρα έχουν την ικανότητα να παράγουν μία προκαθορισμένη τάση ή ρεύμα δίχως να επηρεάζονται από οποιοδήποτε άλλο στοιχείο ενός κυκλώματος. Για αυτό ονομάζονται ανεξάρτητες πηγές.Όμως υπάρχει και μία άλλη κατηγορία πηγών των οποίων η έξοδος είναι συνάρτηση κάποιας άλλης τάσης ή ρεύματος σε ένα κύκλωμα.Αυτές ονομάζονται εξαρτώμενες ή ελεγχόμενες πηγές.Ο συμβολισμός των εξαρτώμενων πηγών φαίνεται στο παρακάτω σχήμα:



Η σχέση μεταξύ της εξαρτώμενης πηγής τάσης v_s ή ρεύματος i_s με την τάση v_x ή το ρεύμα i_x ,όπου τα v_x ή i_x μπορεί να είναι οποιοδήποτε τάση ή ρεύμα στο κύκλωμα φαίνεται παρακάτω:

Τύπος πηγής	*Σχέση*
Πηγή τάσης εξαρτώμενη από τάση(VCVS)	$v_s=\mu v_x$
Πηγή τάσης εξαρτώμενη από ρεύμα(CCVS)	$v_s = ri_x$
Πηγή ρεύματος εξαρτώμενη από τάση(VCCS)	$i_s = gv_x$
Πηγή ρεύματος εξαρτώμενη από ρεύμα(CCCS)	$i_s = \beta i_x$

1.2 Κατηγοριοποίηση Δικτύων – Περιορισμοί Τοπολογίας

Όπως είδαμε παραπάνω ,οι πυκνωτές και τα πηνία αναφέρονται ως δυναμικά στοιχεία.Ένα δίκτυο αναφέρεται ως ένα δυναμικό δίκτυο εάν περιέχει δυναμικά στοιχεία, διαφορετικά καλείται δίκτυο αντιστασεών .Ένα δίκτυο με μόνο γραμμικά στοιχεία ονομάζεται γραμμικό δίκτυο, διαφορετικά ονομάζεται μη γραμμικό .Είναι επίσης χρήσιμη η ταξινόμηση των δικτύων ανάλογα με το αν έχουν η όχι πηγές τάσης .Έτσι τα δίκτυα μπορούν να είναι γραμμικά η μη γραμμικά ,δυναμικά , με η χωρίς πηγές τάσης.
Στην πλειοψηφία τους τα στοιχεία για τα οποία μιλήσαμε παραπάνω είναι χρόνο-αμετάβλητα υπό την έννοια ότι η χαρακτηριστική συνάρτηση i-v είναι αμετάβλητη με το πέρασμα του χρόνου .Ένα δίκτυο που αποτελείται μόνο από τέτοια στοιχεία ονομάζεται χρόνο-αμετάβλητο.

Περιορισμοί τοπολογίας :

Όταν ένα στοιχείο έχει σχέση με την τοπολογία ενός δικτύου τότε αναφέρεται ως κλάδος.Ένα δίκτυο μπορεί να μοντελοποιηθεί βάση κάποιων περιορισμών μέσα από ένα σύστημα εξισώσεων.

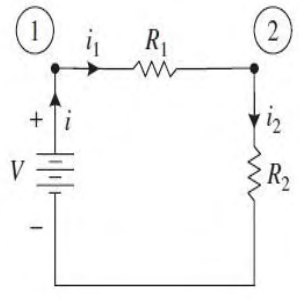
Υπάρχουν δύο τύποι περιορισμών:

Α) Περιορισμοί κλάδων , οι οποίοι ονομάζονται εξισώσεις κλάδων ή εξισώσεις στοιχείων.
Β) Περιορισμοί τοπολογίας οι οποίοι προκύπτουν από τον νόμο του Kirchoff(KCL) και από τον νόμο των τάσεων (KVL).Όλοι αυτοί είναι γραμμικοί αλγεβρικοί περιορισμοί οι οποίοι προκύπτουν από την δομή του ίδιου του δικτύου και όχι από τους περιορισμούς του κάθε στοιχείου ξεχωριστά.

1.3 Εξισώσεις τοπολογίας κυκλωματων

Εισαγωγή:
Η συμπεριφορά ενός κυκλώματος μοντελοποιείται από τον συνδυασμό ενός συνόλου εξισώσεων των ηλεκτρικών στοιχείων και των νόμων των ρευμάτων του Kirchoff και των νόμων των τάσεων .Γενικά αυτό οδηγεί σε ένα σύνολο ταυτόχρονων μη γραμμικών πρώτου βαθμού διαφορικών εξισώσεων οι οποίες προέρχονται από την συνδεσιμότητα και όχι από την φύση των στοιχείων . Για ένα γραμμικό κύκλωμα οι εξισώσεις της μοντελοποίησης είναι γραμμικές. Για παράδειγμα θεωρείστε το κύκλωμα της εικόνας 3.Από τον νόμο των ρευμάτων μπορούμε να γράψουμε

$$i = i_1 + i_2 \qquad (1.1)$$



Σχ3: Ενα γραμμικό κύκλωμα

Οι εξισώσεις των στοιχείων:

$$u_1 = V,\ i_1 = (u_1 - u_2)/R_1, \quad i_2 = u_2/R_2 \tag{1.2}$$

δεδομένου ότι $i_1 = i_2$ έχουμε :

$$(u_1 - u_2)/R_1 = u_2/R_2 \tag{1.3}$$

Εφαρμόζοντας τον νόμο των τάσεων στον βρόχο του σχήματος έχουμε:

$$V = R_1 * i_1 + R_2 * i_2 = (R_1 + R_2) * i = {(R_1 + R_2) * u_2}/{R_2} \tag{1.4}$$

$$u_2 = (R_2/R_1 + R_2) * V \tag{1.5}$$

Έχοντας βρει την τάση u_2 μπορούμε βάση της σχέσης (1.3) να υπολογίσουμε την τάση u_1.

Η ανωτέρω ειδική προσέγγιση της επίλυσης των εξισώσεων μέσω της αντικατάστασης δεν είναι αποδοτική για κυκλώματα μεγάλης κλίμακας . Αντ ’ αυτού χρησιμοποιούμε μια συστηματική και αυτόματη προσέγγιση για την επίλυση των ηλεκτρικών κυκλωμάτων .

Εξισώσεις τοπολογίας κυκλωμάτων [10]

Γενικότερα για ένα κύκλωμα με V={0,1, … , $n-1$}, το σύνολο των n κόμβων του κυκλώματος (κόμβος 0 είναι ο κόμβος αναφοράς η γείωσης), και E = {$e_1, e_2, \dots, e_n$} το σύνολο των n κλάδων του κυκλώματος.Εάν σε κάθε κλάδο προσαρτηθεί μια αυθαίρετη (αλλα πάντα συνδιασμένη) φορά αναφοράς τότε ο ελαττωμένος πίνακας πρόσπτωσης A(ως προς τον κόμβο αναφοράς η γείωσης) του προσανατολισμένου γράφου του κυκλώματος ορίζεται ως εξής:

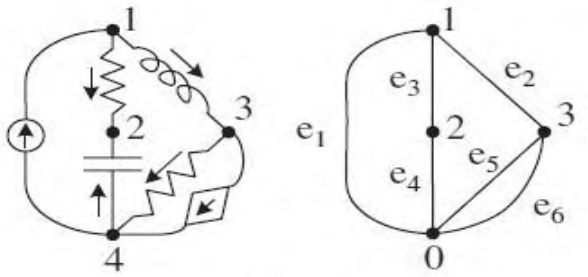
- πίνακας A είναι $(n-1) \times n$ διαστάσεων
- $a_{ij} = \begin{cases} +1\ \textit{εάν ο κλάδος } j \textit{ εξέρχεται απο τον } i \\ -1\ \textit{εάν ο κλάδος } j \textit{ εισέρχεται στον } i \\ 0, \textit{εάν ο κλάδος } j \textit{ δεν συνδέεται με τον κόμβο } \iota \end{cases}$

Εάν $u(t) = [u_1(t), \dots, u_n(t)]^T$ το διάνυσμα των τάσεων κατά μήκος των n κλάδων, $v(t) = [v_1(t), \dots v_{n-1}]^T$ το δίανυσμα των δυναμικών των $n-1$ κόμβων (ως προς τον κόμβο αναφοράς η γείωσης),*και* $i(t) = [i_1(t), \dots, i_n(t)]^T$ το διάνυσμα των ρευμάτων των n κλάδων ,τότε οι εξισώσεις τοπολογίας του κυκλώματος εκφράζονται από τους :

1. Νόμος του Kirchoff με την μορφή $Ai = 0$ όπου A ελαττωμένος πίνακας πρόσπτωσης i διάνυσμα με όλα τα ρεύματα κλάδων
2. Νόμος των τάσεων $u(t) = A^T * v(t)$ όπου u είναι διάνυσμα με όλες τις τάσεις κλάδων , v διάνυσμα με όλες τις τάσεις κόμβων .

Παράδειγμα

Έστω το κύκλωμα της εικόνας 4



ΣΧ.4

Για το κύκλωμα αυτό ο ελαττωμένος πίνακας πρόσπτωσης είναι:

$$A = \begin{matrix} -1 & +1 & +1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & +1 & +1 \end{matrix}$$

Οι νόμοι των τάσεων και των ρευμάτων του Kirchoff είναι οι εξής:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ +1 & 0 & -1 \\ +1 & -1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & +1 \\ 0 & 0 & +1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \qquad \begin{bmatrix} -1 & +1 & +1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & +1 & + \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

1.3.1. Σχέση ρεύματος – τάσης (i-v) μεταξύ των κλάδων του κυκλώματος

Η εξίσωση κάθε κλάδου είναι:

$$i_{bk}(t)=Z_k(D)v_{bk}(t)$$

Όπου k=1,2,...,β το πλήθος των κλάδων του κυκλώματος

$$Z_k(D) = \begin{cases} \frac{1}{R_k}, \text{άν ο κλάδος έχει αντίσταση} \\ \frac{1}{L_k}\int_0^t * \, dt = \frac{1}{L_k}D^{-1}, \text{άν ο κλάδος έχει αυτεπαγωγή} \\ C_k \frac{d}{dt} *= C_k D, \text{αν ο κλάδος έχει χωρητικότητα} \end{cases}$$

1.3.2.Τροποποιημένη ανάλυση κόμβων

Η τροποποιημένη ανάλυση κόμβων (MNA) διατυπώθηκε το 1975 ,(έχει κάποια χαρακτηριστικά δεδομένου ότι μπορει να εφαρμοστεί σε οποιοδήποτε κύκλωμα) υλοποιείται ως εξής :

Έστω ότι τα n στοιχεία ενός κυκλώματος χωρίζονται σε δύο ομάδες:

- Στοιχεία των οποίων οι εξισώσεις μπορούν να γραφούν στην μορφή

$$i_k(t) = g_k * u_k(t) + C_k * \frac{du_k}{dt} + S_k(t)$$

(όπου $S_k(t)$γνωστή συνάρτηση του t που εκφράζει την πηγή ρεύματος)

- Στοιχεία των οποίων οι εξισώσεις δεν μπορούν να γραφούν υπο αυτή την μορφή.Επιζητείται το ρεύμα τους κατά την προσομοίωση περιλαμβάνονται αυτεπαγωγές ,πηγές τάσης καθώς και αντιστάσεις(και ίσως χωρητικότητες) των οποίων το ρεύμα είναι ζητούμενο

Έστω n_1 ο αριθμός των στοιχείων της πρώτης ομάδας($group\ 1$) και n_2 ο αριθμός των στοιχείων της δεύτερης ομαδάς($group\ 2$) όπου $n = n_1 + n_2$
Χωρίζουμε τον ελαττωμένο πίνακα πρόσπτωσης A και τα διανύσματα $u(t)$ και $i(t)$ σε υποπίνακες και υποδιανύσματα ως εξής:

$A = [A_1\ A_2]$ $όπου\ A_1 = (n-1) \times n_1 και\ A_2 = (n-1) \times n_2$

$u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$ $όπου\ u_1(t) = n_1 \times 1\ και\ u_2(t) = n_2 \times 1$

$i(t) = \begin{bmatrix} i_1(t) \\ i_2(t) \end{bmatrix}$ $όπου\ i_1(t) = n_1 \times 1\ και\ i_2(t) = n_2 \times 1$

Βάση του κανόνα του Kirchoff έχουμε:

$$Ai = 0 \overset{1.2.1,1.2.3}{\Longleftrightarrow} A_1 * i_1(t) + A_2 * i_2(t) = 0 \ (1.2.1)$$

Βάση του κανόνα των τάσεων έχουμε:

$$u(t) = A^T * v(t) \overset{12.1,1.2.2}{\Longleftrightarrow} \begin{cases} u_1(t) = A_1^T * v(t)(1.2.2a) \\ u_2(t) = A_2^T * v(t)(1.2.2b) \end{cases} \kappa\alpha\theta\acute{\omega}\varsigma\ A^T = \begin{bmatrix} A_1^T \\ A_2^T \end{bmatrix}$$

Οί εξισώσεις των στοιχείων της ομάδας G1 γράφονται υπο μορφή πινάκων :

$\iota_1(t) = G * u_1(t) + C * \frac{du_1(t)}{dt} + S_1(t)(1.2.3)$, όπου G διαγώνιος πίνακας $n_1 \times n_1$ με μη μηδενικές τιμές στην διαγώνιο στην θέση των αντιστάσεων και μηδενικά στην θέση χωρητικοτήτων και πηγών ρεύματος.
C πίνακας διαγώνιος $n_1 \times n_1$ με μη μηδενικές τιμές στην θέση των χωρητικοτήτων και $S_1(t)$ $διάνυσμα$ $n_1 \times 1$με μη μηδενικές τιμές στην θέση των πηγών ρεύματος.

Οί εξισώσεις των στοιχείων της ομάδας G_2 γράφονται υπο την μορφή πίνακα:

$$u_2(t) = R * i_2(t) + L * \frac{di_2(t)}{dt} + S_2(t)(1.2.4)$$

Όπου R διαγώνιος $n_2 \times n_2$ πίνακας με μη μηδενικές τιμές στην διαγώνιο στην θέση των αντιστάσεων των οποίων το ρεύμα αναζητείται.
L διαγώνιος $n_2 \times n_2$ πίνακας με μη μηδενικές τιμές στην διαγώνιο στην θέση των αυτεπαγωγών. $S_2(t)$ διάνυσμα $n_2 \times 1$ με μη μηδενικές τιμές στις τιμές των πηγών τάσεων.

Εάν αντικαταστήσουμε την (1.2.2.α) στην (1.2.3) και εν συνεχεία την τελευταία στην(1.2.1) θα έχουμε :

$$A_1 G A_1^T v(t) + A_1 C A_1^T \frac{dv(t)}{dt} + A_2 i_2(t) = -A_1 S_1(t)(1.2.5)$$

Επίσης αντικαθιστώντας την (1.2.2b) στην (1.2.4) προκύπτει :

$$A_2^T v(t) - R i_2(t) - L \frac{di_2(t)}{dt} = S_2(t)(1.2.6)$$

Η (1.2.5) αποτελεί ένα σύστημα (n-1) εξισώσεων ως προς (n-1)+n_2 αγνώστους συναρτήσεις (τις $v(t)$ και i_2(t)).Αντίστοιχα η b αποτελεί ένα σύστημα n_2 εξισώσεων ως προς τους ίδιους αγνώστους .Ο συνδυασμός των (1.2.5) και (1.2.6) θα δώσει ένα μεγαλύτερο σύστημα $[(n-1)+n_2] \times [(n-1)+n_2]$:

$$\begin{bmatrix} A_1 G A_1^T & A_2 \\ A_2^T & -R \end{bmatrix} \begin{bmatrix} v(t) \\ i_2(t) \end{bmatrix} + \begin{bmatrix} A_1 C A_1^T & 0 \\ 0 & -L \end{bmatrix} \begin{bmatrix} \frac{dv(t)}{dt} \\ \frac{di_2(t)}{dt} \end{bmatrix} = \begin{bmatrix} -A_1 S_1(t) \\ S_2(t) \end{bmatrix} \qquad \text{(Σύστημα MNA)}$$

Στην DC ανάλυση δεν υπάρχει χρονική μετάδοση και το σύστημα γίνεται:

$$\begin{bmatrix} A_1 G A_1^T & A_2 \\ A_2^T & -R \end{bmatrix} \begin{bmatrix} v(t) \\ i_2(t) \end{bmatrix} = \begin{bmatrix} -A_1 S_1(t) \\ S_2(t) \end{bmatrix}$$

Είναι της μορφής Ax=b

$A_1 G A_1^T = G_n$ όπου
A_1 πίνακας πρόσπτωσης: $(n-1) \times n_1$,
G διαγώνιος πίνακας αγωγιμοτήτων κλάδων :$n_1 \times n_1$
A_1^T: $(n-1) \times n_1$

G_n πίνακας αγωγιμοτήτων κόμβων: $(n-1) \times (n-1)$

$\begin{cases} g_{ii} \text{ άθροισμα αγωγιμοτήτων που προσπίπτουν στον κόμβο } i \; i = 1, \dots, n-1 \\ g_{ij} \text{ αρνητική αγωγιμότητα που συδέει τους δύο κόμβους } i \text{ και } j \end{cases}$

Στην πράξη το σύστημα αυτό διαμορφώνεται με βάση τις ακόλουθες συνεισφορές των ακολουθιακών στοιχείων.

Αν το στοχείο συνδέεται με τους κόμβους i και j και η j είναι η γείωση τότε στον πίνακα εμφανίζεται μόνο το στοιχείο $g_{ii.}$

Γία παράδειγμα για μία αντίσταση της ομάδας 1 ο πίνακας θα έχει την ακόλουθη μορφή

$$\begin{matrix} & v+ & v- & \\ v+ & 1/R & -1/R & \cdots \\ v- & -1/R & 1/R & \cdots \\ & \cdots & \cdots & \cdots \end{matrix}\begin{bmatrix} v+ \\ v- \\ \cdots \end{bmatrix} = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix}$$

Ένω για μία αντίσταση της ομάδας 2 ο πίνακας είναι:

$$\begin{matrix} & v+ & v- & k \\ v+ & & & +1 \\ v- & & & -1 \\ k & +1 & -1 & -R \end{matrix}\begin{bmatrix} v+ \\ v- \\ i_k \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

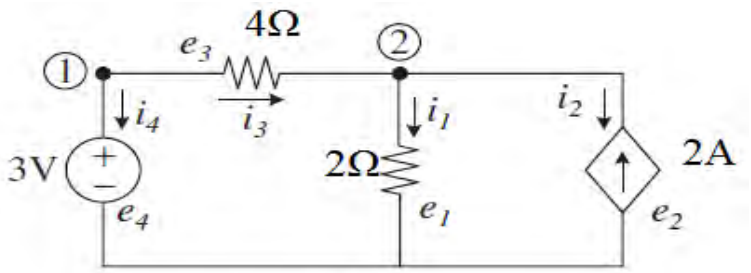
Η ανεξάρτητη πηγή τάσης έχει συνεισφορές στα A_2, A_2^T, S_2:

$$\begin{matrix} & v+ & v- & i_K \\ v+ & & & +1 \\ v- & & & -1 \\ i_K & +1 & -1 & \end{matrix}\begin{bmatrix} v+ \\ v- \\ i_K \end{bmatrix} = \begin{bmatrix} \\ \\ S_K \end{bmatrix}$$

Ανεξάρτητη πηγή ρεύματος:

$$\begin{matrix} v+ \\ v- \end{matrix}\begin{bmatrix} & \end{bmatrix}\begin{bmatrix} v+ \\ v- \end{bmatrix} = \begin{bmatrix} -S_K \\ S_K \end{bmatrix}$$

Ένα παράδειγμα της τροποποιημένης μεθόδου των κόμβων, θεωρείστε το κύκλωμα της εικόνας 4.



ΣΧ.4

Για το παραπάνω κύκλωμα επιζητείται το ρεύμα από την αντίσταση στον κλάδο e_3 ο αριθμός των κόμβων είναι n-1=2 ενώ ο αριθμός των στοιχείων group 2 είναι 2(αντίσταση,πηγή τάσης).

Για τον κλάδο e_1 έχουμε μία αντίσταση της ομάδας 1 άρα η συνεισφορά στον πίνακα θα είναι:

$$< v+>< v+>: +g_k$$
$$< v+>< v->: -g_k$$
$$< v->< v+>: -g_k$$
$$< v->< v->: +g_k$$

Για τις αντιστάσεις της ομάδας 2 η συνεισφορά τους στον πίνακα είναι:

$$< v+>< (n-1)+k >: +1$$
$$< v->< (n-1)+k >: -1$$
$$< (n-1)+k >< v+>: +1$$
$$< (n-1)+k >< v->: -1$$
$$< (n-1)+k >< (n-1)+k >: -r_k$$

Για τις ανεξάρτητες πηγές τάσης η συνεισφορά τους είναι ως εξής:

$$< v+>< (n-1)+k >: +1$$
$$< v->< (n-1)+k >: -1$$
$$< (n-1)+k >< v+>: +1$$
$$< (n-1)+k >< v->: -1$$

Δεξί μέλος:

$$< (n-1)+k > 1: +S_K$$

Η ανεξάρτητη πηγή ρεύματος έχει συνεισφορά μόνο στο δεξί μέλος του συστήματος :

$$< +1 >< 1 >: -S_K$$

$$< -1 >< 1 >: +S_K$$

Ποίο συγκεκριμένα για τον κλάδο e_1 το αντίστοιχο στοιχείο θα είναι στην θέση (2,2) και θα έχει τιμή $+1/2$ δεδομένου ότι ο άλλος κόμβος είναι στη γή η συνεισφορά του συγκεκριμένου στοιχείου θα είναι μόνο σε αυτή την θέση .

Για τον κλάδο e_2 έχουμε την πηγή ρεύματος για αυτή την πηγή ο θετικός πόλος είναι στον κόμβο 0 ενώ ο αρνητικός στον κόμβο 2 .Η συνεισφορά αυτού του στοιχείου θα είναι μόνο στο δεξί μέλος του συστήματος και στην θέση (2,1) με τιμή +2

Για τον κλάδο e_3 έχουμε μια αντίσταση την ομάδας 2 .Οι κόμβοι στους οποίους προσπίπτει αυτή η αντίσταση είναι ο 1 και ο 2 για αυτή την αντίσταση η συνεισφορά θα είναι :

στοιχείο (1,3):+1
στοιχείο (2,3):-1
στοιχείο (3,1):+1
στοιχείο (3,2):-1
στοιχείο (3,3):-4

Για τον κλάδο e_4 έχουμε μια πηγή τάσης της ομάδας 2 η συνεισφορά αυτού του στοιχείου στον πίνακα είναι η ακόλουθη :

στοιχείο (1,4):+1
στοιχείο (4,1):+1 ενώ η συνεισφορά στο δεξί μέλος θα είναι +3 στο στοιχείο (4,1)

Ύστερα από τα παραπάνω ο πίνακας MNA σχηματίζεται ως εξής:

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1/2 & -1 & 0 \\ 1 & -1 & -4 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ i_3 \\ i_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 3 \end{bmatrix}$$

Έαν δεν θέλουμε να βρούμε το ρεύμα από τον κλάδο e_3 τότε θα επιλύσουμε ως προς τις τάσεις u_1, u_2 και i_4.Οι κλάδοι e_1 και e_2 παραμένουν οι ίδιοι ενώ οι κλάδοι e_3και e_4 αλλάζουν και γίνονται

Κλάδος e3(αντίσταση ομάδας 1)

Πίνακας:

Στοιχείο (1,1): $+1/4$
Στοιχείο (1,2): $-1/4$

Στοιχείο (2,1): $-1/4$
Στοιχείο (2,2): $+1/4$

Τα στοιχεία της ομάδας 2 είναι 1 (η πηγή τάσης) έτσι

Για τον κλάδο e4 (πηγή τάσης)

Στοιχείο (1,3):+1

Στοιχείο (3,1):+1 ενώ στο δεξί μέλος έχουμε στο στοιχείο (3,1):+3

Ύστερα από αυτές τις τροποποιήσεις ο καινούργιος πίνακας MNA είναι ο ακόλουθος:

$$\begin{bmatrix} 1/4 & -1/4 & 1 \\ -1/4 & 3/4 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ i_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}$$

1.4 Τεχνικές Επίλυσης Γραμμικών Συστημάτων

Σύστημα Γραμμικών Εξισώσεων:

Ax= b , $A \in \mathcal{R}^{n*n}$ όπου A πίνακας τετραγωνικός και αντιστρέψιμος x , $b \in \mathcal{R}^{n}$

Άμεσες μέθοδοι επίλυσης οι οποίες τερματίζουν σε προκαθορισμένο αριθμό βημάτων

Παραγοντοποίηση LU:
Εάν $A \in \mathcal{R}^{n\times n}$ είναι αντιστρέψιμος , τότε υπάρχουν μοναδικοί πίνακες

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Τέτοιοι ώστε PA=LU, όπου $P \in \Re^{n \times n}$ είναι πίνακας μετάθεσης (permutation) ο οποίος προκύπτει από τον μοναδιαίο πίνακα I με εναλαγές γραμμών

Π.χ

$$I = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \qquad P = \begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Ο πίνακας PA προκύπτει από τον A με τις ίδιες εναλλαγές γραμμών και υλοποιεί τη λειτουργία της μερικής οδήγησης η οδήγησης γραμμών ,κατά την οποία επιλέγεται γραμμή εξίσωση με το μεγαλύτερο διαγώνιο στοιχείο κατά το τρέχον βήμα ώστε να αποφευχθεί η απώλεια αριθμητικής ακρίβειας και η πιθανότητα διαίρεσης με το 0.Η ολική οδήγηση χρησιμοποιείται για αραιούς πίνακες .

1.5.1 Αλγόριθμος Απαλοιφής Gauss

$$a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=b_1$$
$$a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n=b_2$$
$$\vdots \quad \vdots \quad \vdots \quad \vdots$$
$$a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n=b_n$$

$$\downarrow$$

$$a_{11}x_1+\ a_{12}\,x_2+\ \ldots+a_{1n}\,x_n=\ b_1$$
$$0x_1+\left[a_{22}-a_{12}\left(\frac{a_{21}}{a_{11}}\right)\right]x_2+\ldots+\left[a_{2n}-a_{1n}\left(\frac{a_{21}}{a_{11}}\right)\right]x_n=b_2-b_1\left(\frac{a_{21}}{a_{11}}\right)$$
$$0x_1+\left[a_{32}-a_{12}\left(\frac{a_{31}}{a_{11}}\right)\right]x_2+\ldots+\left[a_{3n}-a_{1n}\left(\frac{a_{31}}{a_{11}}\right)\right]x_n=b_3-b_1\left(\frac{a_{31}}{a_{11}}\right)$$
$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad = \quad \vdots$$
$$0x_1+\left[a_{n2}-a_{12}\left(\frac{a_{n1}}{a_{11}}\right)\right]x_2+\ldots+\left[a_{nn}-a_{1n}\left(\frac{a_{n1}}{a_{11}}\right)\right]x_n=b_n-b_1\left(\frac{a_{n1}}{a_{11}}\right)$$

Σε μορφή πινάκων:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \ldots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & \ldots & a_{2n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & a_{n3}^{(1)} & \ldots & a_{nn}^{(1)} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \quad \rightarrow$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix} \quad (1)$$

Ο αλγόριθμος επαναλαμβάνεται n φορές με πρώτο βήμα το αρχικό σύστημα , μετά από n βήματα έχουμε:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \dots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \dots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

Η ακολουθία αλλαγών στα στοιχεία του b είναι :

Βήμα	Στοιχείο b_i
0	$b_i^{(1)} = b_i$(αρχική τιμή)
1	$b_i^{(2)} = b_i^{(1)} - \left(\frac{a_{i1}^{(1)}}{a_{11}^{(1)}}\right) * b_1^{(1)}$
2	$b_i^{(3)} = b_i^{(2)} - \left(\frac{a_{i2}^{(2)}}{a_{22}^{(2)}}\right) * b_2^{(2)}$
	⋮
k	$b_i^{(k+1)} = b_i^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) * b_k^{(k)}$
	⋮
i-1	$b_i^{(i)} = b_i^{(i-1)} - \left(\frac{a_{ii-1}^{(i-1)}}{a_{i-1i-1}^{(i-1)}}\right) * b_{i-1}^{(i-1)}$

Δηλαδή το στοιχείο i αλλάζει μέχρι το βήμα i-1

Αναπτύσοντας την αναδρομή έχουμε :

Για το $b_i^{(i)}$ μετά από i βήματα :

$$b_i^{(i)} = b_i - \sum_{j=1}^{i-1} \left(\frac{a_{ij}^{(j)}}{a_{jj}^{(j)}}\right) * b_j^{(j)} \rightarrow b_i^{(i)} + \sum_{j=1}^{i-1} \left(\frac{a_{ij}^{(j)}}{a_{jj}^{(j)}}\right) * b_j^{(j)} = b_i$$

Το οποίο γράφεται υπο μορφή πίνακα:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ (a_{21}^{(1)}/a_{11}^{(1)}) & 1 & 0 & \cdots & 0 \\ (a_{31}^{(1)}/a_{11}^{(1)}) & (a_{32}^{(2)}/a_{22}^{(2)}) & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (a_{n1}^{(1)}/a_{11}^{(1)}) & (a_{n2}^{(2)}/a_{22}^{(2)} & (a_{n3}^{(3)}/a_{33}^{(3)}) & \cdots & 1 \end{bmatrix} \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Πολλαπλασιάζοντας από αριστερά και τα δύο μέλη της (1) με L έχουμε την τελική μορφή $LUx=b$

Παράδειγμα :

$$y + z = 2$$

$$2x + 3z = 5$$

$$x + y + z = 3$$

$$A=\begin{bmatrix} 0 & 1 & 1 & 2 \\ 2 & 0 & 3 & 5 \\ 1 & 1 & 1 & 3 \end{bmatrix}$$

Βήμα 1

Εναλλαγή της $1^{ης}$ με την $2^{η}$ γραμμή

$$\begin{bmatrix} 2 & 0 & 3 & 5 \\ 0 & 1 & 1 & 2 \\ 1 & 1 & 1 & 3 \end{bmatrix}$$

Βήμα 2

Διαίρεση της $1^{ης}$ γραμμής με 2

$$\begin{bmatrix} 1 & 0 & 3/2 & 5/2 \\ 0 & 1 & 1 & 2 \\ 1 & 1 & 1 & 3 \end{bmatrix}$$

Βήμα 3

Αφαίρεσε την 1η γραμμή από την 3η

$$\begin{bmatrix} 1 & 0 & 3/2 & 5/2 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & -1/2 & 1/2 \end{bmatrix}$$

Βήμα 4

Αφαίρεσε την 3η γραμμή από την 2η

$$\begin{bmatrix} 1 & 0 & 3/2 & 5/2 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & -3/2 & -3/2 \end{bmatrix}$$

Βήμα 5

Διαίρεσε την 3η γραμμή με $-3/2$

$$\begin{bmatrix} 1 & 0 & 3/2 & 5/2 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Η λύση που έχουμε είναι x=1 y=1 z=1

1.5.2 Αλγόριθμος παραγοντοποίησης LU

Για $k = 1 : n$
  $x = |a_{kk}|$
  Για $i = k : n$
    Εάν $|a_{ik}| \geq x$ *τότε* $m = i$
  Εναλλαγή της γραμμής k με την γραμμή m
  Τέλος
  Για $i = k + 1 : n$
    $a_{ik} = l_{ik} = \frac{a_{ik}}{a_{kk}}$
  Τέλος
  Για $i = k + 1 : n$
    Για $j = k + 1 : n$
      $a_{ij} = a_{ij} - l_{ik} * a_{kj}$
    Τέλος
  Τέλος
Τέλος

Η διαγώνιος του L δεν χρειάζεται αποθήκευση .Τα στοιχεία l_{ik} με i≥k μπορούν να αποθηκευτούν στο κάτω τριγωνικό τμήμα (του A→LU) μόλις υπολογίζονται .
Η πολυπλοκότητα του αλγορίθμου είναι O(n^3).

Παράδειγμα:

$$A = \begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Για να βρούμε τα στοιχεία l_{21}, l_{31} χρείαζεται να βρούμε τον αριθμό που χρησιμοποιούμε για να κάνουμε τα στοιχεία α_{21} και α_{31} ίσα με μηδέν

$l_{21} = \frac{64}{25} = 2{,}56$, $l_{31} = \frac{144}{25} = 5{,}76$

Για να βρούμε το στοιχείο l_{32} θα χρειαστεί να βρούμε τον αριθμό που μηδενίζει το στοιχείο α_{32} στην δεύτερη επανάληψη . Ο πίνακας Α στην αρχή της δεύτερης επανάληψης είναι:

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & -16.8 & -4.76 \end{bmatrix}$$

Έτσι $l_{32} = \frac{-16.8}{-4.8} = 3.5$ άρα ,

$$\text{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \quad \text{και} \quad \text{U} = \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

1.5.3 Αλγόριθμος Cholesky [12]

Εάν ο $A \in \Re^{n \times n}$ είναι πίνακας συμμετρικός (δηλαδή $A=A^T$) και θετικά ορισμένος (δηλαδή $x^T Ax>0$, $\forall x \in \Re^n, x \neq 0$) τότε υπάρχει μοναδικός κάτω τριγωνικός πίνακας L τέτοιος ώστε $A=LL^T$ (χωρίς πίνακα μεταθέσεων P).Άρα δεν χρειάζεται και οδήγηση

Μπορούμε να σπάσουμε τον πίνακα Α στην εξής μορφή:

$$\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} = \begin{bmatrix} l_{11}^2 & l_{11}L_{21}^T \\ l_{11}L_{21} & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix}$$

$$l_{11} = \sqrt{a_{11}} \ , L_{21} = \frac{1}{l_{11}} A_{21}$$

$$A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T \quad ,$$

Βήμα 1 : Εφόσον ο πίνακας είναι θετικά ορισμένος τότε $\alpha_{11} > 0$

Βήμα 2 : Εφόσον ο πίνακας είναι θετικά ορισμένος τότε

$$A_{22} - L_{21}L_{21}^T = A_{22} - \frac{1}{a_{11}} A_{21}A_{21}^T \text{ θετικά ορισμένος}$$

1.5.4 Ψευδοκώδικας Αλγορίθμου Cholesky

Για k = 1 : n

$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$

Για i = k + 1 : n

$a_{ik} = l_{ik} = \frac{1}{l_{kk}} \left(a_{ki} - \sum_{j=1}^{k-1} l_{ij} * l_{kj}\right)$

Τέλος

Τέλος

Ο αλγόριθμος αποθηκεύει τα στοιχεία l_{ik}(i>k) στο κάτω τριγωνικό μέρος του συμμετρικού A εκτός από την διαγώνιο l_{kk}, την οποία αποθηκεύει σε ξεχωριστό διάνυσμα .Η παραγοντοποίηση Cholesky δεν απαιτεί τη χρήση οδήγησης ,ο αλγόριθμος αποτυγχάνει αν ο πίνακας δεν είναι θετικά ορισμένος γιατί τότε:

$a_{kk} < \sum_{j=1}^{k-1} l_{kj}^2$ για κάποιο k δηλαδή έχουμε αρνητική ρίζα

Παράδειγμα :

$$A = \begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

Πρώτη στήλη του L

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & l_{22} & 0 \\ -1 & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 5 & 3 & -1 \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

Δεύτερη στήλη του L

$$\begin{bmatrix}18 & 0\\ 0 & 11\end{bmatrix} - \begin{bmatrix}3\\ -1\end{bmatrix}[3 \quad -1\,] = \begin{bmatrix}l_{22} & 0\\ l_{32} & l_{33}\end{bmatrix}\begin{bmatrix}l_{22} & l_{32}\\ 0 & l_{33}\end{bmatrix}$$

$$\begin{bmatrix}9 & 3\\ 3 & 10\end{bmatrix} = \begin{bmatrix}3 & 0\\ 1 & l_{33}\end{bmatrix}\begin{bmatrix}3 & 1\\ 0 & l_{33}\end{bmatrix}$$

Τρίτη στήλη του L: 10-1=$l_{33}^2 \Rightarrow l_{33} = 3$

Τελικά

$$\begin{bmatrix}25 & 15 & -5\\ 15 & 18 & 0\\ -5 & 0 & 11\end{bmatrix} = \begin{bmatrix}5 & 0 & 0\\ 3 & 3 & 0\\ -1 & 1 & 3\end{bmatrix}\begin{bmatrix}5 & 3 & -1\\ 0 & 3 & 1\\ 0 & 0 & 3\end{bmatrix}$$

1.5.4 Επίλυση τριγωνικών συστημάτων

Το σύστημα $LUx=b$ μετά την παραγοντοποίηση ανάγεται στην επίλυση των τριγωνικών συστημάτων:

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Το κάτω τριγωνικό σύστημα $Ly = b$ λύνεται με τον αλγόριθμο forward substitution:

Για k = 1 : n
  Για j = 1 : k-1
    $b_k = b_k - l_{kj} * y_j$
  Τέλος
  $y_k = \frac{b_k}{l_{kk}}$
Τέλος

Ο αλγόριθμος θα πρέπει να πραγματοποιεί την τυχόν μετάθεση των στοιχείων του b.

Το άνω τριγωνικό σύστημα $Ux = y$ λύνεται με τον αλγόριθμο backward substitution και δίνει την τελική λύση του *x.*

Για k = 1 : n
  Για j = k+1 : n

    $y_k = y_k - u_{kj} * x_j$
  Τέλος
  $x_k = \frac{y_k}{u_{kk}}$
Τέλος

Η πολυπλοκότητα των δύο αλγορίθμων είναι O(n^2).

2. MNA

Στα πλαίσια αυτού του κεφαλαίου αναπτύχθηκε ένας parser σε κώδικα C ο οποίος διαβάζει ένα αρχείο της μορφής .txt το οποίο κάθε φορά δίνει ο χρήστης και στο οποίο περιγράφεται η τοπολογία ενός γραμμικού κυκλώματος .

2.1. Τοπολογία

Μέσα από αυτό το αρχείο .txt γίνεται η περιγραφή του κυκλώματος .Η σύμβαση που θα πρέπει να δίνουμε για την σωστή εκχώρηση των στοιχείων του κυκλώματος είναι η εξής:

Για τις πηγές τάσης :

- V<ακέραιος> <κόμβος.+> <κόμβος.-> <τιμή>

Για τις πηγές ρεύματος :

- I<ακέραιος> <κόμβος.+> <κόμβος.-> <τιμή>

Για τις αντιστάσεις :

- R<ακέραιος> <κόμβος.+> <κόμβος.-> <τιμή>

Για τις αντιστάσεις group 2 :

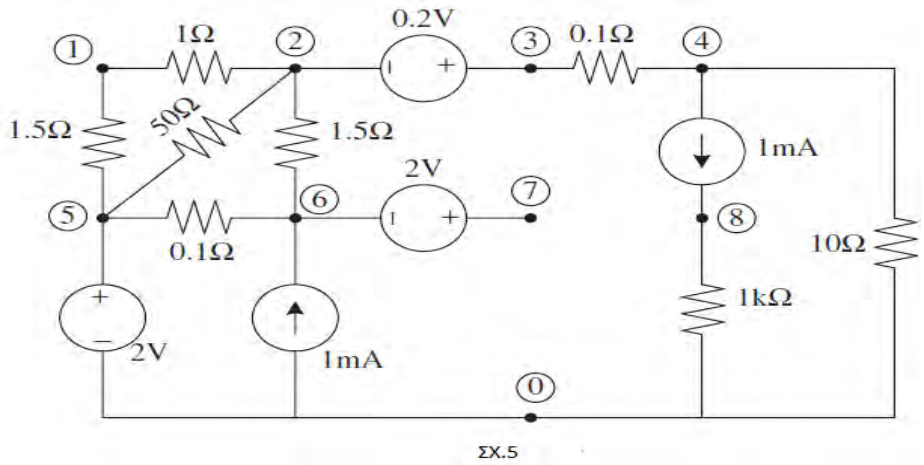
- R<ακέραιος> <κόμβος.+> <κόμβος.-> <τιμή> G

σαν 0 κόμβο θα θεωρούμε την γείωση .

V1 5 0 2
V2 3 2 0.2
I1 4 8 0.001
R1 1 5 1.5
R3 5 2 50 G
R5 2 6 1.5
R7 8 0 1000
R8 4 0 10 G
V3 7 6 2
I2 0 6 0.001
R2 1 2 1
R4 5 6 0.1
R6 3 4 0.1

Για τις πηγές τάσης η σύμβαση πού έχουμε πάρει είναι ότι ο πρώτος κόμβος που δίνουμε στο αρχείο θα είναι ο θετικός πόλος της πηγής ενώ ο δεύτερος ο αρνητικος. Την ίδια σύμβαση έχουμε και για τις πηγές ρεύματος έτσι ώστε η θετική φορά του ρεύματος να δίνεται από τον πρώτο κόμβο του αρχείου.
Στίς αντιστάσεις οι οποίες ανήκουν στο group 2 των ηλεκτρικών στοιχείων θα δίνεται ο συμβολισμός G αμέσως μετά την εκχώρηση της τιμής της

Το σχήμα 5 μοντελοποιείται βάση των στοιχείων που δώσαμε στο αρχείο .txt

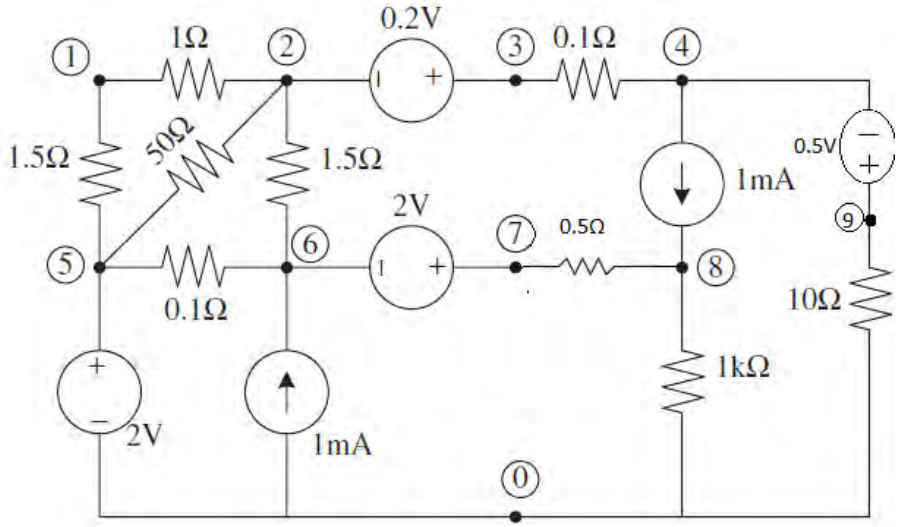


ΣΧ.5

..

Ένα άλλο κύκλωμα που μπορούμε να δώσουμε για προσομοίωση είναι το ακόλουθο στο οποίο έχουμε προσθέσει μία πηγή τάσης και μία αντίσταση :

V1 5 0 2
V3 7 6 2
I2 0 6 0.001
R2 1 2 1
R4 5 6 0.1
R6 3 4 0.1
R8 9 0 10 G
V4 9 4 0.5
V2 3 2 0.2
I1 4 8 0.001
R1 1 5 1.5
R3 5 2 50 G
R5 2 6 1.5
R7 8 0 1000
R9 7 8 0.5



ΣΧ.6

Ο λόγος για τον οποίο δώσαμε δύο παραδείγματα είναι για να μπορέσουμε να εξετάσουμε τον χρόνο εκτέλεσης των δύο κυκλωμάτων .Ο χρόνος αυτός εξετάζεται στήν 2.2.2 υποενότητα του ίδιου κεφαλαίου στην οποία παρουσιάζονται επιπλέον οι λύσεις των δύο παραδειγμάτων.

2.2. Κατασκευή και Επίλυση Συστήματος MNA[5]

Ο στόχος μας είναι να κάνουμε επίλυση του συστήματος $[A] * [x] = [b]$

Για ένα κύκλωμα με N κόμβους και M ανεξάρτητες πηγές τάσης:

- Ο πίνακας A:
  - Είναι $(M + N) \times (M + N)$ όπου N είναι ο αριθμός των κόμβων και M είναι ο αριθμός των ανεξάρτητων πηγών τάσης
  - Το $(N \times N)$ μέρος του πίνακα είναι το πάνω αριστερά:
    - Έχει μόνο παθητικά στοιχεία(αγωγιμότητες)
    - Στοιχεία που συνδέονται με την γείωση αναπραριστόνται στην διαγώνιο
    - Στοιχεια που δεν συνδέονται στην γείωση βρίσκονται στην διαγώνιο αλλα και εκατέροθεν αυτής .
  - Το υπόλοιπο του πίνακα περιέχει(εκτός από τον πάνω αριστερά υποπίνακα N×N) μόνο 1,-1, 0(άλλες τιμές είναι πιθανές εάν υπάρχουν εξαρτημένες πηγές ρεύματος και τάσης)

- Το διάνυσμα x :
  - Είναι $(N + M) \times 1$ το οποίο περιέχει τις μεταβλητές των άγνωστων στοιχείων
  - Εάν έχουμε και K στοιχεία group 2 τότε το δίανυσμα είναι $(N + K + M) \times 1$
  - Τα N πρώτα στοιχεία του διανύσματος είναι οι n κορυφές των τάσεων
  - Για το δίανυσμα $(N + K + M) \times 1$ τα επόμενα K στοιχεία αντιπροσωπεύουν τα ρεύματα κατά μήκος των αντιστατών του group 2
  - Τα υπόλοιπα M στοιχεία αντιπροσωπεύουν τα ρεύματα κατά μήκος των M ανεξάρτητων πηγών τάσης

- Το διάνυσμα z:
    - Είναι $(N + M) \times 1$ το οποίο περιέχει γνωστές τιμές
    - Τα N πρώτα στοιχεία του διανύσματος είναι είτε 0 ή η θετική είτε η αρνητική τιμή των ανεξάρτητων πηγών ρεύματος του κυκλώματος
    - Τα υπόλοιπα M στοιχεία αντιπροσωπεύουν τις M ανεξάρτητες πηγές τάσης του κυκλώματος

Κατασκευή του πίνακα A:

Ο πίνακας A σχηματίζεται από τον συνδυασμό 4 υποπινάκων και έχει την ακόλουθη μορφή:

$$A = \begin{bmatrix} G & B \\ C & D \end{bmatrix}$$

Ο πίνακας A είναι $(N + M) \times (N + M)$(όπου N είναι ο αριθμός των κόμβων και M ο αριθμός των ανεξάρτητων πηγών τάσης):

- G πίνακας N×N ο οποίος καθορίζεται από τις διασυνδέσεις μεταξύ των στοιχείων του κυκλώματος
- B πίνακας N×M ο οποίος καθορίζεται από τις διασυνδέσεις μεταξύ των πηγών τάσης
- $C = B^T$
- D μηδενικός πίνακας M×M

Κανόνες κατασκευής πίνακα G:

Ο πίνακας G είναι ένας πίνακας N×N όταν δεν υπάρχουν στοιχεία group 2 αντιστατών στο κύκλωμα μας ενώ όταν υπάρχουν και το πλήθος αυτών είναι K τότε ο πίνακας αυτός είναι $(N + K) \times (N + K)$.

1. Το κάθε στοιχείο της διαγωνίου είναι ίσο με το άθροισμα της αγωγιμότητας του κάθε στοιχείου που συνδέεται με τον συγκεκριμένο κόμβο .Έτσι το πρώτο στοιχείο της διαγωνίου είναι το άθροισμα των αγωγιμοτήτων οι οποίες συνδέονται με τον κόμβο 1, το δεύτερο στοιχείο είναι το άθροισμα των αγωγιμοτήτων στον κόμβο 2

2. Για τα στοιχεία που ανήκουν στο group 2 παρουσιάζονται από το N+1 μέχρι το K-στοιχείο της διαγωνίου και στο οποίο αποθηκεύεται η αρνητική τιμή της αντίστασης
3. Τα στοιχεία εκτός διαγωνίου περιέχουν την αρνητική αγωγιμότητα του κάθε στοιχείου που συνδέεται με το ζεύγος των αντίστοιχων κόμβων .Κατά συνέπεια μια αντίσταση μεταξύ των κόμβων 1 και 2 πηγαίνει στον πίνακα G στην θέση (1,2) και (2,1). Για τα στοιχεία εκτός διαγωνίου των group 2 οι τιμές θα είναι η +1 η -1 .Ποίο συγκεκριμένα εάν ο πρώτος κόμβος με τον οποίο συνδέεται η αντίσταση είναι θετικός τότε στην θέση G[N+1][κόμβος1-1]+=1 και αντίστοιχα και το συμμετρικό του .Αν ο δεύτερος κόμβος με τον οποίο συνδέεται η αντίσταση group 2 είναι θετικός τότε G[N+1][κόμβος2-1]-=1 και αντίστοιχα και ο συμμετρικός του.

Αν το στοιχείο είναι γειωμένο τότε καταχωρείται μόνο στην διαγώνιο του πίνακα

Κανόνες κατασκευής πίνακα B:

Ο πίνακας B είναι N×M στοιχείων ενώ αν υπάρχουν στοιχεία group 2 είναι (N+K) × M ο πίνακας B περιέχει μόνο 0, 1, -1 τιμές .Κάθε θέση στον πίνακα αντιστοιχεί σε μια συγκεκριμένη πηγή τάσης (στήλες) ή έναν κόμβο (γραμμές).Αν ο θετικός πόλος της i-πηγής τάσης συνδέεται με τον κόμβο k,τότε το στοιχείο (k, i) στον πίνακα B είναι 1.Εάν ο αρνητικός πόλος της i-πηγής τάσης συνδέεται με τον k κόμβο ,τότε το στοιχείο (k,i) στον πίνακα B θα είναι -1 .Σε αντίθετη περίπτωση τα στοιχεία του πίνακα B θα είναι 0

Κανόνες κατασκευής πίνακα C:

Ο πίνακας B είναι M×N στοιχείων ενώ αν υπάρχουν στοιχεία group 2 είναι M ×(N+K) ο πίνακας B περιέχει μόνο 0, 1, -1 τιμές .Κάθε θέση στον πίνακα αντιστοιχεί σε μια συγκεκριμένη πηγή τάσης (γραμμές) ή έναν κόμβο (στήλες).Αν ο θετικός πόλος της i-πηγής τάσης συνδέεται με τον κόμβο k,τότε το στοιχείο (i, k) στον πίνακα B είναι 1.Εάν ο αρνητικός πόλος της i-πηγής τάσης συνδέεται με τον k κόμβο ,τότε το στοιχείο (i,k) στον πίνακα B θα είναι -1 .Σε αντίθετη περίπτωση τα στοιχεία του πίνακα B θα είναι 0.

Με άλλα λόγια ο πίνακας C είναι ο ανάστροφος πίνακας του B

Κανόνες κατασκευής πίνακα D:

Ο πίνακας D είναι ένας πίνακας M×M με όλα του τα στοιχεία να είναι 0

2.2.1 Περιγραφή αλγορίθμου

Η τοπολογία αυτή αποτελείται από αντιστάσεις,πηγές τάσης και και πηγές ρεύματος.Εν συνεχεία η κάθε γραμμή του αρχείου μέσω της οποίας γίνεται η αναπαράσταση του κάθε στοιχείο του κυκλώματος (η τιμής του και οι δύο κόμβοι με τους οποίους γίνεται η σύνδεση) αποθηκεύεται σε μία λίστα(η δομή της λίστας περιγράφεται στο παράρτημα Α.Ακολούθως μέσω της βιβλιοθήκης mna.h γίνεται η κατασκευή του πίνακα Α και του διανύσματος b . Ποίο αναλυτικά μέσω της βιβλιοθήκης αυτής ο πίνακας Α αρχικοποιείται με 0 σε όλα τα στοιχεία του στην συνέχεια διαβάζεται το κάθε στοιχείο από την λίστα και ανάλογα από τον τύπο του στοιχείου αποθηκεύεται στην αντίστοιχη θέση στον πίνακα .Συγκεκριμένα αρχικά αποθηκεύονται οι αντιστάσεις που έχουν δηλωθεί σαν group 1. Στην κύρια διαγώνιο αποθηκεύονται το άθροισμα των αγωγιμοτήτων που προσπίπτουν στον αντίστοιχο κόμβο.Τα στοιχεία εκτός διαγωνίου είναι συμμετρικά .

Η λύση αυτού του συστήματος γίνεται μέσω της μεθόδου του Gauss και της παραγοντοποίησης Cholesky η δεύτερη επιτυγχάνεται εάν ο πίνακας είναι συμμετρικός και θετικά ορισμένος θετικά ορισμένος.

Στην επόμενη ενότητα θα μιλήσουμε για τους αραιούς πίνακες και την επίλυση αυτών των συστημάτων μέσω αντίστοιχων μεθόδων.

2.2.2. Παράδειγμα εκτέλεσης

Μία ενδεικτική εκτέλεση του πρώτου παραδείγματος της προηγούμενης ενότητας παρουσιάζεται παρακάτω :

V (1) = 1.88527V
V (2) = 1.80879V
V (3) = 2.00879V
V (4) = 1.9888V
V (5) = 2V
V (6) = 1.98814V
V (7) = 3.98814V
V (8) = 1V

I (R3) = 3.82mA
I (R8) = 198.88mA
I (V1) = -198.88mA
I (V2) = -199.88mA
I (V3) = 0A

Ο κώδικας την υλοποίησης αναπτύσσεται στο Παράρτημα Α στο οποίο φαίνεται και ο τρόπος με τον οποία βρίσκουμε το χρόνο υπολογισμού των λύσεων του συστήματος.Ακόμη πρέπει να σημειωθεί ότι οι λύσεις των αποτελεσμάτων τυπώνονται σε στρογγυλοποιημένη μορφή και σε αναπαράσταση τριών δεκαδικών ψηφίων μετά την υποδιαστολή .

Ενώ εδώ έχουμε τις λύσεις των αποτελεσμάτων για το δεύτερο παράδειγμα:

V(1) = 1.885V
V(2) = 1.809V
V(3) = 2.009V
V(4) = 1.989V
V(5)= 2V
V(6)= 1.988V
V(7)= 3.988V
V(9)= 1.989V

I(R3)= 4mA
I(R8)= 199mA
I(V1)= -202mA
I(V2)= -200mA
I(V3)= -3mA
I(V4)= -199mA
V(8)= 3.986V

Μέσος όρος χρόνου επίλυσης πρώτου κυκλώματος σχήμα 5 : 15 milliseconds

Μέσος όρος χρόνου επίλυσης δεύτερου κυκλώματος σχήμα 6: 31 milliseconds

3. Προσομοίωση με αραιούς πίνακες

3.1 Εισαγωγή

Ο αριθμός των στοιχείων σε ένα μεγάλο ηλεκτρικό δίκτυο είναι συνήθως μόνο 2-4 φορές ο αριθμός των κόμβων.Ακομή έχουμε ότι ο αριθμός των ακμών σε ένα δίκτυο είναι περίπου O(n^2) όπου n είναι ο αριθμός των κορυφών σε ένα δίκτυο στην πραγματικότητα είναι O(n) για κύκλωμα μεγάλης κλίμακας .Σαν αποτέλεσμα οι πίνακες αυτοί είναι εξαιρετικά αραιοί ,τα περισσότερα στοιχεία είναι 0, με την μέθοδο των αραιών πινάκων μπορεί να γίνει η επίλυση των συστημάτων με μεγάλη ταχύτητα .

3.2 Κατασκευή αραιών πινάκων

3.2.1 Αναπαράσταση σε μορφή triplet

Η αναπαράσταση triplet ,δεδομένου ενός πίνακα A m×n με nz το πλήθος των μη μηδενικών στοιχείων, αποτελείται από δύο διανύσματα ακέραιων αριθμών και ένα διάνυσμα πραγματικών

- Διάνυσμα ακεραίων : r[1] , r[2] , . . . , r[nz] ∈ {1,2, . . . ,m}
- Διάνυσμα ακεραίων : c[1] , c[2] , . . . , c[nz] ∈ {1,2, . . . , n}
- Διάνυσμα πραγματικών : x[1] , x[2] , . . . ,x[nz] ∈ R όπου x[i] = A(r[i] ,c[i]).

Για παράδειγμα ο πίνακας

$$A = \begin{bmatrix} 5.6 & 0 & 1.8 & 0 \\ 2.4 & 1.5 & 0 & -0.2 \\ 0 & 1.6 & 4 & 0 \\ 2.8 & -3 & 0 & 1.7 \end{bmatrix}$$

Έχει την ακόλουθη triplet μορφή:

r = [3 2 4 1 2 4 4 2 1 3]

c = [3 1 4 3 2 1 2 4 1 2]

x = [4.0 2.4 1.7 1.8 1.5 2.8 -3.0 -0.2 5.6 1.6]

3.2.2 Αναπαράσταση σε μορφή compress-column

Η μορφή compress-column για έναν πίνακα A m × n ο οποίος μπορεί να περιέχει περισσότερα από nzmax μη μηδενικά στοιχεία αποτελείται από 3 διανύσματα:

- Διάνυσμα ακεραίων : p[1] , p[2] , . . . , p[n+1] ∈ { 1, 2 , . . . , nzmax+1 }
- Διάνυσμα ακεραίων : r[1] , r[2] , . . . , r[nzmax] ∈ { 1 , 2 , . . . , m }
- Πίνακας πραγματικών : x[1] , x[2] , . . . , x[nzmax] ∈ R, όπου p[0] = 0 ,p[n+1]=nz ,

Όπου nz < nzmax είναι το ακριβές πλήθος των μη μηδενικών στοιχείων , το διάνυσμα p είναι τέτοιο ώστε οι γραμμές με τα μη μηδενικά στοιχεία της στήλης j να αποθηκεύονται στα i [p [j]] , i [p[j]+1], . . ., i[p[j+1]-1] ενώ οι αντίστοιχες τιμές των nonzeros στα ίδια ως προς x x[p[j]] , x[p[j]+1] , . . . , x[p[j+1]-1].Εάν η στήλη j δεν έχει μη μηδενικά στοιχεία τότε p[j] = p[j+1]

Έτσι για τον πίνακα A η αναπαράσταση σε μορφή compress-column είναι η εξής:

p = [1 4 7 9 11]

r = [1 2 4 2 3 4]

x = [5.6 2.4 2.8 1.5 1.6 -3.0 1.8 4.0 -0.2 1.7]

3.3 Βασικοί αλγόριθμοι χειρισμού αραιών πινάκων

Η δήλωση ενός πίνακα A σε μορφή triplet γίνεται μέσω μίας δομής struct cs*A (3.2.1) η οποία περιέχει τα διανύσματα i , j και x για τα οποία έγινε αναφορά στο προηγούμενο κεφάλαιο και τα οποία περιέχουν τις διαστάσεις του πίνακα και το πλήθος των μη μηδενικών στοιχείων :

```
typedef struct cs_sparse    /* matrix in compressed-column or triplet form */
{
   csi nzmax ;     /* maximum number of entries */
   csi m ;         /* number of rows */
   csi n ;         /* number of columns */
   csi *p ;        /* column pointers (size n+1) or col indices (size nzmax) */     (3.2.1)
   csi *i ;        /* row indices, size nzmax */
   double *x ;     /* numerical values, size nzmax */
   csi nz ;        /* # of entries in triplet matrix, -1 for compressed-col */
} cs ;
```

Ενώ η εκχώρηση μνήμης γίνεται μέσω της συνάρτησης *cs_spalloc(int m, int n, int nzmax,int values,int triplet) (3.2.2).Η εντολή που δίνουμε για την δέσμευση μνήμης για έναν πίνακα A n× n με nz μη μηδενικά στοιχεία είναι:

A=cs_spalloc(n,n,nz,1,1)

Συνάρτηση cs_spalloc :

```
#include "cs.h"
 cs *cs_spalloc(int m,int n,int nzmax,int values,int triplet)
 {
    cs *A = cs_calloc(1,sizeof(cs));      /*allocate the cs struct*
    if (!A) return (NULL);                /* out of memory*/
    A->m = m;                             /* define dimensions and nzmax*/
    A->n = n;

    A->nzmax = nzmax = CS_MAX (nzmax,1);                                    (3.2.2)
    A->nz = triplet ? 0 : -1;             /* allocate triplet or comp.col*/
```

```
    A->p = cs_malloc (triplet ? nzmax : n+!,sizeof (int)) ;
    A->i = cs_malloc(nzmax,sizeof(int)) ;
    A->x = values ? cs_malloc (nzmax,sizeof (double)) -: NULL ;
    return ((!A->p || !A->i || (values && !A->x)) ? cs_spfree (A) : A) ;
  }
```

Εν συνεχεία η μετατροπή ενός πίνακα από μορφή triplet σε μορφή compress-column γίνεται μέσω της συνάρτησης cs_compress. C=cs_compress(A)(3.2.3) , ο πίνακας C πρέπει να έχει δηλωθεί ως cs *C, ενώ ο A πρέπει κατόπιν να ελευθερωθεί από την μνήμη με την συνάρτηση cs_spfree(A)(3.2.4).Διαφορετικά nonzeros x στην ίδια θέση i,j του πίνακα συγχωνεύονται με την συνάρτηση cs_dupl(C)(3.2.5).

```
#include "cs.h"

/* C = compressed-column form of a triplet matrix T */
cs *cs_compress (const cs *T)
{
   csi m, n, nz, p, k, *Cp, *Ci, *w, *Ti, *Tj ;
   double *Cx, *Tx ;
   cs *C ;
   if (!CS_TRIPLET (T)) return (NULL) ;                /* check inputs */
   m = T->m ; n = T->n ; Ti = T->i ; Tj = T->p ; Tx = T->x ; nz = T->nz ;
   C = cs_spalloc (m, n, nz, Tx != NULL, 0) ;          /* allocate result */
   w = cs_calloc (n, sizeof (csi)) ;                   /* get workspace */
   if (!C || !w) return (cs_done (C, w, NULL, 0)) ;    /* εκτός μνήμης */          (3.2.3)
   Cp = C->p ; Ci = C->i ; Cx = C->x ;
   for (k = 0 ; k < nz ; k++) w [Tj [k]]++ ;           /* μετρητής στηλών */
   cs_cumsum (Cp, w, n) ;                              /*δείκτης στηλών */
   for (k = 0 ; k < nz ; k++)
   {
      Ci [p = w [Tj [k]]++] = Ti [k] ;    /* A(i,j) is the pth entry in C */
      if (Cx) Cx [p] = Tx [k] ;
   }
   return (cs_done (C, w, NULL, 1)) ;      /* success; free w and return C */
}
```

```
#include "cs.h"

cs *cs_spfree(cs *A)
{
   if(!A) return(NULL); /* μην κάνεις τίποτα αν ο A είναι NULL*/
   cs_free(A->p);
   cs_free(A->i);                                                    (3.2.4)
   cs_free(A->x);
   return (cs_free(A));       /*απελευθέρωσε cs struct και επέστρεψε  NULL*/
   }
```

```
#include "cs.h"

/* αφαίρεσε διπλές εισαγωγές από τον A */
csi cs_dupl (cs *A)
{
   csi i, j, p, q, nz = 0, n, m, *Ap, *Ai, *w ;
   double *Ax ;
   if (!CS_CSC (A)) return (0) ;             /* ξέκαρε την εισαγωγή του πίνακα*/
   m = A->m ; n = A->n ; Ap = A->p ; Ai = A->i ; Ax = A->x ;
   w = cs_malloc (m, sizeof (csi)) ;         /*δέσμευσε μνήμη */
   if (!w) return (0) ;                      /* εκτός χώρου μνήμης */
   for (i = 0 ; i < m ; i++) w [i] = -1 ;    /* η γραμμή  i δεν έχει ακόμα οριστεί  */
   for (j = 0 ; j < n ; j++)
   {
      q = nz ;                               /* στήλη j θα ξεκινάει από το q */
      for (p = Ap [j] ; p < Ap [j+1] ; p++)
      {
         i = Ai [p] ;                        /* A(i,j) είναι μη μηδενικά */                (3.2.5)
         if (w [i] >= q)
         {
            Ax [w [i]] += Ax [p] ;           /* A(i,j) είναι διπλό */
         }
         else
         {
            w [i] = nz ;                     /* καταχώρησε εκέι που εμφανίζεται η στήλη  i  */
            Ai [nz] = i ;                    /* κράτησε το A(i,j) */
            Ax [nz++] = Ax [p] ;
         }
      }
```

```
    Ap [j] = q ;                    /* έναρξη καταχώρησης στήλης j */
  }
  Ap [n] = nz ;                     /* οριστικοποίηση του A */
  cs_free (w) ;                     /* απελευθέρωσε μνήμη */
  return (cs_sprealloc (A, 0)) ;          /* αφαίρεσε επιπλέον χώρο από τον A */
}
```

Μέσω της συνάρτησης cs_load (3.2.6) μπορούμε να κατανοήσουμε πλήρως τον τρόπο αναπαράστασης ενός πίνακα σε sparse μορφή .Η συνάρτηση αυτή αρχικοποιεί έναν άδειο αραιό πίνακα triplet μορφής με 0.Οι διαστάσεις του πίνακα T καθορίζονται από τον μέγιστο δείκτη γραμμής και στήλης οι οποίες διαβάζονται από το αρχείο. Εν συνεχεία καλείται η συνάρτηση cs_entry η οποία τοποθετεί στοιχεία στην θέση i,j

```
#include "cs.h"
/* load a triplet matrix from a file */
cs *cs_load (FILE *f)
{
  double i, j ;   /* use double for integers to avoid csi conflicts */
  double x ;
  cs *T ;
  if (!f) return (NULL) ;                    /* check inputs */
  T = cs_spalloc (0, 0, 1, 1, 1) ;           /* allocate result */          (3.2.6)
  while (fscanf (f, "%lg %lg %lg\n", &i, &j, &x) == 3)
  {
    if (!cs_entry (T, (csi) i, (csi) j, x)) return (cs_spfree (T)) ;
  }
  return (T) ;
}
```

Συνάρτηση επίλυσης lusol(3.2.7):

```
#include "cs.h"
/* x=A\b όπου A είναι μη συμετρικός , b αντικαθίσταται με την λύση  */
csi cs_lusol (csi order, const cs *A, double *b, double tol)
{
   double *x ;
   css *S ;
   csn *N ;
   csi n, ok ;
   if (!CS_CSC (A) || !b) return (0) ;     /* τσεκάρισμα των εισόδων */
   n = A->n ;
   S = cs_sqr (order, A, 0) ;              /* ordering and symbolic analysis */
   N = cs_lu (A, S, tol) ;                 /* numeric LU factorization */
   x = cs_malloc (n, sizeof (double)) ;    /* δέσμευση μνήμης */                    (3.2.7)
   ok = (S && N && x) ;
   if (ok)
   {
      cs_ipvec (N->pinv, b, x, n) ;       /* x = b(p) */
      cs_lsolve (N->L, x) ;               /* x = L\x */
      cs_usolve (N->U, x) ;               /* x = U\x */
      cs_ipvec (S->q, x, b, n) ;          /* b(q) = x */
   }
   cs_free (x) ;
   cs_sfree (S) ;
   cs_nfree (N) ;
   return (ok) ;
}
```

```
#include "cs.h"
/* y = A*x+y */
csi cs_gaxpy (const cs *A, const double *x, double *y)
{
   csi p, j, n, *Ap, *Ai ;
   double *Ax ;
   if (!CS_CSC (A) || !x || !y) return (0) ;       /* τσεκάρισμα των εισόδων */
   n = A->n ; Ap = A->p ; Ai = A->i ; Ax = A->x ;
   for (j = 0 ; j < n ; j++)                                                    (3.2.8)
   {
      for (p = Ap [j] ; p < Ap [j+1] ; p++)
      {
         y [Ai [p]] += Ax [p] * x [j] ;
      }
   }
   return (1) ;
}
```

Ο υπολογισμός της 1-νόρμας ενός πίνακα μπορεί να γίνει με την συνάρτηση cs_norm (3.2.9)

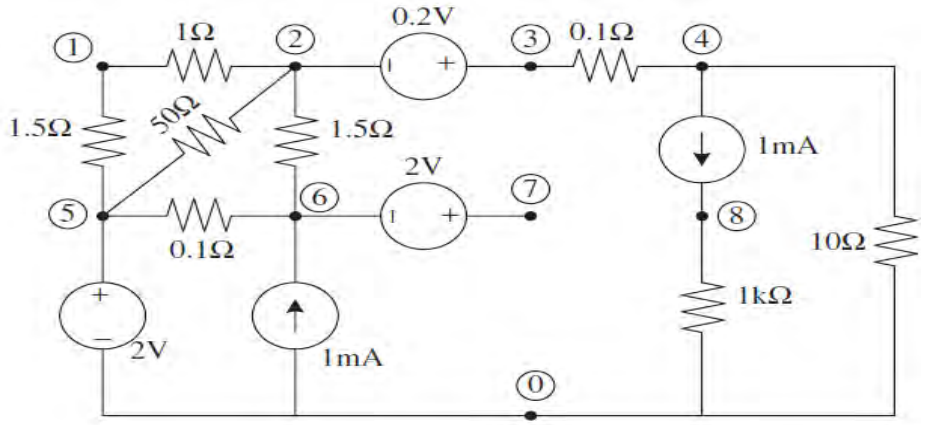
```
#include "cs.h"
/* 1-norm of a sparse matrix = max (sum (abs (A))), largest column sum */
double cs_norm (const cs *A)
{
   csi p, j, n, *Ap ;
   double *Ax,  norm = 0, s ;
   if (!CS_CSC (A) || !A->x) return (-1) ;          /* check inputs */          (3.2.9)
   n = A->n ; Ap = A->p ; Ax = A->x ;
   for (j = 0 ; j < n ; j++)
   {
      for (s = 0, p = Ap [j] ; p < Ap [j+1] ; p++) s += fabs (Ax [p]) ;
      norm = CS_MAX (norm, s) ;
   }
return(norm);
}
```

Με την συνάρτηση cs_entry (3.2.10) αρχικά εισάγουμε και εν συνεχεία προσθέτουμε ένα στοιχείο στον πίνακα triplet η συνάρτηση που το περιγράφει φαίνεται παρακάτω:

```
#include "cs.h"
/*πρόσθεσε ένα στοιχείο σε ένα πίνακα triplet; επέστρεψε 1 εάν είναι οκ, 0 αλλιώς */
csi cs_entry (cs *T, csi i, csi j, double x)
{
   if (!CS_TRIPLET (T) || i < 0 || j < 0) return (0) ;      /* τσέκαρε τις εισόδους */
   if (T->nz >= T->nzmax && !cs_sprealloc (T,2*(T->nzmax))) return (0) ;
   if (T->x) T->x [T->nz] = x ;                                                    (3.2.10)
   T->i [T->nz] = i ;
   T->p [T->nz++] = j ;
   T->m = CS_MAX (T->m, i+1) ;
   T->n = CS_MAX (T->n, j+1) ;
   return (1) ;
}
```

4. Sparse τοπολογία

Η τοπολογία του ηλεκτρικού κυκλώματος είναι η ίδια με την τοπολογία της προηγούμενης ενότητας



ΣΧ.5

4.1 Κατασκευή και επίλυση

Η κατασκευή του συστήματος χρησιμοποιόντας sparse πίνακες γίνεται ως εξής :

Λαμβάνοντας τον πίνακα A από το δεύτερο κεφάλαιο ο οποίος βρίσκεται σε μη αραιή απεικόνηση θα χρειαστεί να τον μετατρέψουμε σε πίνακα sparse στην μορφή triplet οι αλλαγές που θα χρειαστεί να κάνουμε θα είναι στην βιβλιοθήκη mna.h ενώ η βιβλιοθήκη gauss.h δεν μας χρειάζεται γιατί αυτή την φορά θα χρησιμοποιήσουμε την συνάρτηση lusol η οποία βρίσκεται στο αρχείο CSparse ο κώδικας της οποίας παρουσιάστηκε στο προηγούμενο κεφάλαιο.Ως εκ τούτου αφού διαβάσουμε το αρχείο και αποθηκεύσουμε τα στοιχεία σε μία λίστα καλούμε την συνάρτηση getMNA().Μέσω αυτής της συνάρτησης δεσμεύουμε μνήμη μέσω της συνάρτησης cs_spalloc με παραμέτρους (0,0,1,1,1) εν συνεχεία καλούμενη την συνάρτηση setMatrixA(Qt) με την βοήθεια της οποίας θα γίνει η αποθήκευση των στοιχείων στο πίνακα triplet Qt κατόπιν θα χρειαστεί να καλέσουμε την συνάρτηση compress(Qt) έτσι ώστε να μετατρέψει τον πίνακα triplet σε πίνακα compress-column. Τέλος θα χρειαστεί να καλέσουμε την σύναρτηση free(Qt) για να αποδεσμεύσουμε την μνήμη.Στην συνάρτηση

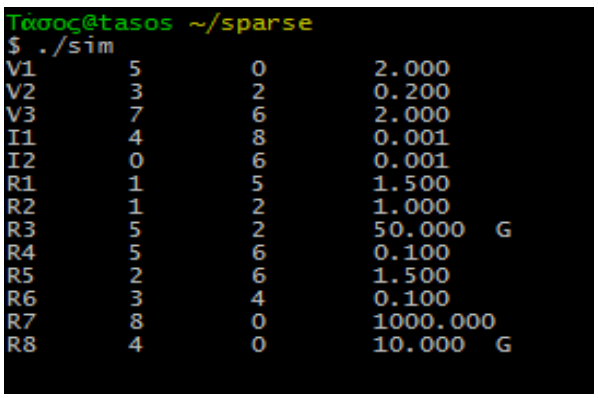
setMatrixA(Qt) ορίζουμε την συνάρτηση addQt.Σε αυτή την συνάρτηση κάθε φορά εξετάζουμε τα στοιχεία που δίνουμε με τις συγκεκριμένες συντεταγμένες κατόπιν η συνάρτηση αυτή κάνει τον έλεγχο έτσι ώστε αν δεί ότι ήδη υπάρχει κάποιο στοιχείο με αυτές τις συντεταγμένες τότε προσθέτει στην παλία τιμή την καινούργια τιμή και τερματίζει , αν δεν βρεί κάποιο στοιχείο που να υπάρχει ήδη σε εκείνη την θέση τότε το τοποθετεί μέσω της συνάρτησης cs_entry. Ο τρόπος υλοποίησης του b μέλους του συστήματος παραμένει ο ίδιος με του κεφαλαίου 2. Ποίο αναλυτικά αρχικοποιούμε ολά τα στοιχεία του διανύσματος με 0 .Για κάθε πηγή ρεύματος κοιτάμε σε ποίους κόμβους προσπύπτει δηλαδή αν προσπύπτει στον αρνητικό κόμβο τότε στην αντί-στοιχη θέση βάζουμε την αρνητική τιμή της πηγής ρεύματος αντίστοιχα αν προσπύ-πτει στην θετικό πόλο τότε στην αντίστοιχη θέση μπαίνει η θετική τιμή της πηγής .Στή συνέχεια αν έχουμε αντιστάσεις του group 2 τότε τοποθετούνται μηδενικά το πλήθος των οποίων είναι ίσο με το άθροισμα αυτών των αντιστατών.Τέλος τοποθε-τούνται οι απόλυτες τιμές των πηγών τάσεων κατά αύξουσα σειρά όπως τις έχουμε ονομάσει.

4.1.1 Περιγραφή αλγορίθμου

Τρόπος αναπαράστασης του πίνακα MNA σε sparse μορφή μέσω των συναρτήσεων cs_spalloc,cs_compress,cs_entry,addQt,

4.1.2 Παράδειγμα εκτέλεσης

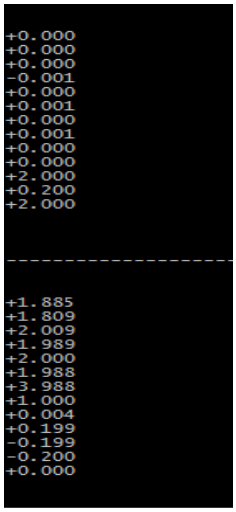
Πρώτο παράδειγμα :



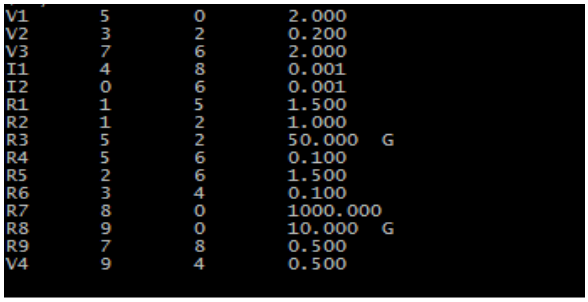
Αναπαράσταση σε μορφή triplet:

```
list contains 13 lines.
CSparse Version 3.0.1, Jan 19, 2010.  Copyright (c) Timothy A. Davis, 2006-2010
13-by-13, nzmax: 35 nnz: 35, 1-norm: 52
    col 0 : locations 0 to 2
      0 : 1.66667
      4 : -0.666667
      1 : -1
    col 1 : locations 3 to 7
      1 : 1.66667
      0 : -1
      5 : -0.666667
      8 : -1
      11 : -1
    col 2 : locations 8 to 10
      2 : 10
      3 : -10
      11 : 1
    col 3 : locations 11 to 13
      3 : 10
      2 : -10
      9 : 1
    col 4 : locations 14 to 18
      4 : 10.6667
      0 : -0.666667
      5 : -10
      8 : 1
      10 : 1
    col 5 : locations 19 to 22
      5 : 10.6667
      4 : -10
      1 : -0.666667
      12 : -1
    col 6 : locations 23 to 23
      12 : 1
    col 7 : locations 24 to 24
      7 : 0.001
    col 8 : locations 25 to 27
      8 : -50
      4 : 1
      1 : -1
    col 9 : locations 28 to 29
      9 : -10
      3 : 1
    col 10 : locations 30 to 30
      4 : 1
    col 11 : locations 31 to 32
      2 : 1
      1 : -1
    col 12 : locations 33 to 34
      6 : 1
      5 : -1
```

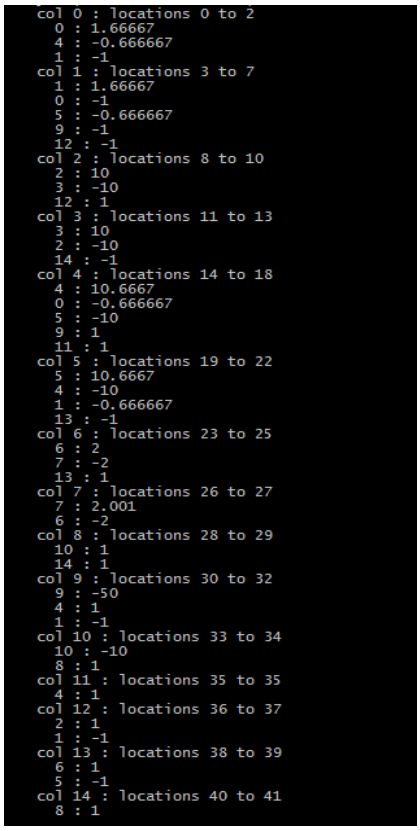
Ενώ τα αποτελέσματα μετά την επίλυση του συστήματος μέσω της συνάρτησης lusol φαίνονται στον παρακάτω πίνακα όπου στο πρώτο διάνυσμα φαίνονται οι τιμές του b ενώ στο δεύτερο οι λύσεις του συστήματος



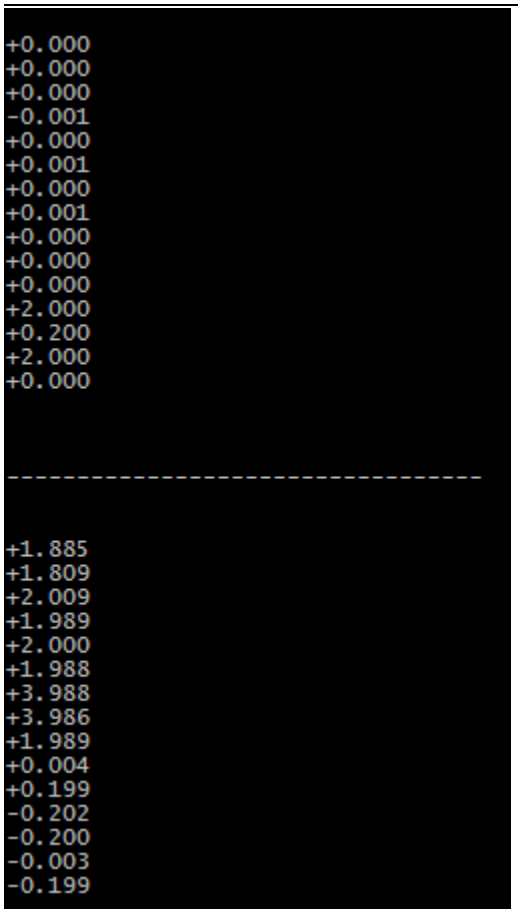
Για το δεύτερο παράδειγμα το netlist το οποίο δίνουμε σαν είσοδο στο πρόγραμμα μας είναι :



Μορφή triplet:



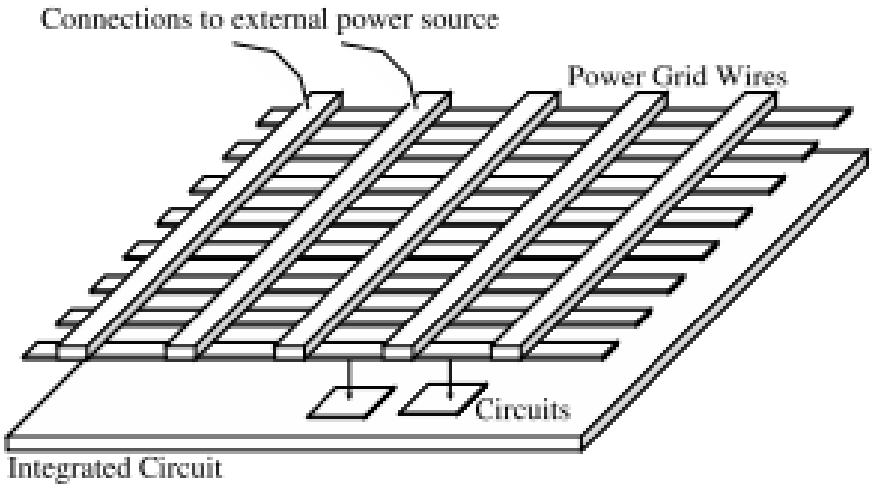
Οι λύσεις του δεύτερου κυκλώματος παρουσιάζονται σε αυτή την εικόνα στην οποία το πρώτο διάνυσμα αναπαριστά το διάνυσμα b ενώ τα αποτελέσματα φαίνονται στο δεύτερο διάνυσμα:



4.2.Πλεονεκτήματα της μεθόδου έναντι MNA και υπολογισμός του χρόνου επίλυσης / χώρου αποθήκευσης με τις δύο τεχνικές

Ποίο αναλυτικά η αναπαράσταση των μηδενικών στοιχείων δεν γίνεται αυτό έχει σαν αποτέλεσμα μεγαλύτερη ταχύτητα στον χρόνο εκτέλεσης για παράδειγμα η παραγοντοποίηση LU ενώ θεωρητικά υπολογίζεται σε O(n^3) στην πραγματικότητα με την μέθοδο των αραιών πινάκων γίνεται O($n^{1.5}$). Για μεγάλο μέγεθος πινάκων γίνεται περίπου O($n^{1.1}$) .Ο συνολικός χρόνος προσομοίωσης παρατηρείται να είναι O(n^a) όπου a να είναι ανάμεσα σε 1.2-1.5

5. Δίκτυα διανομής ισχύος δόμης πλέγματος-Τοπολογία



Με τον όρο power grid αναφερόμαστε σε ένα δίκτυο διανομής ισχύος, το οποίο είναι γραμμικό και συνήθως έχει την μορφή πλήρους η ατελούς πλέγματος στα δύο ανώτερα επίπεδα μετάλλου .Ο προορισμός του δικτύου τροφοδοσίας είναι να διανέμει τις τάσεις τροφοδοσίας και γείωσης σε όλη τη σχεδίαση.Το δίκτυο αυτό αποτελείται από β κλάδους (ωμικούς ή επαγωγικούς) και n κόμβους .Από τους β κλάδους οι β-ρ είναι κλάδοι καλωδίων ενώ οι ρ είναι κλάδοι ακροδεκτών τροφοδοσίας . Όλοι οι κλάδοι καλωδίων έχουν αντίσταση $R_j = \frac{p}{t} * \frac{l_j}{w_j} = R_{sh} * \frac{l_j}{w_j}$,όπου R_{sh}=αντίσταση φύλλου (ειδική αντίσταση πάχους),και επαγωγή $L_j = \frac{N_0}{2\pi} l_j \log(\frac{8h}{w_j})$ όπου h απόσταση από το υπόστρωμα.

Αναλυτικότερα οι n κόμβοι του δικτύου μπορούν να χωριστούν σε δύο ομάδες q+p. Στους p κόμβους του δικτύου εφαρμόζεται τάση τροφοδοσίας V_{dd} προς τη γή.

Επίσης σε κάθε κόμβο i υπάρχει χωρητικότητα προς την γή

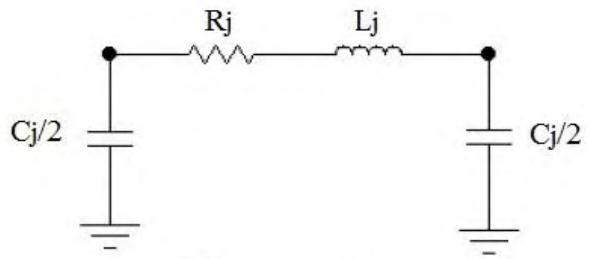
$$C_i = \left(\sum_{\substack{\text{οι κλάδοι } j \\ \text{πού προσπίπτουν} \\ \text{στόν κόμβο } i}} \left(\frac{1}{2} \bar{C}_{pp(ar)} l_{ji} w_{ji} + \frac{1}{2} \bar{C}_{ff(per)} l_{ji} \right) + \right) + C_{pini} + C_{decapi},$$

όπου C_{pin} μία extra παρασιτική χωρητικότητα η οποία εφαρμόζεται όταν υπάρχει πηγή τάσης.
$\bar{C}_{pp(ar)}$: Επιφανειακή χωρητικότητα(ανα μονάδα επιφάνειας) μονάδα μέτρησης fF/μm^2.

Στο κέντρο του chip βάζουμε μία χωρητικότητα αποσύνδεσης C_{decap} για να αντισταθμίσει τον θόρυβο λόγω επαγωγής με όσο το δυνατόν μικρότερη επιφάνεια.

$\bar{C}_{ff(per)}$ χωρητικότητα πλευρικών πεδίων μονάδα μέτρησης fF/μm.

Μία ενδεικτική μορφή του κλάδου ενός δικτύου τροφοδοσίας είναι η ακόλουθη:



Τοπολογία καλωδίου
δικτύου τροφοδοσίας

Η χωρητικότητα C_j του κόμβου j είναι κατανεμημένη δεδομένου ότι κάθε φορά σπάει αριστερά και δεξία του κλάδου .Απο τους υπόλοιπους q κόμβους οι g (με g<<q) συνδέονται σε πηγές ρεύματος που αντιστοιχούν στις ομάδες-blocks πυλών που τροφοδοτούνται και αντλούν ρεύμα από το power grid.

Για τον λόγο ότι έχουμε C και L στοιχεία θα χρησιμοποιήσουμε την Τροποποιημένη μέθοδος κόμβων-αντί για την απλή Μέθοδο των Κόμβων,στο οποίο τα ρεύματα επαγωγής κατά μήκος των κλάδων $i_l(t)$ αποτελούν επιπρόσθετες μεταβλητές επιπλέον των τάσεων κόμβων v_{nd}(t) .

Ειδικά για την περίπτωση του δικτύου τροφοδοσίας όπου όλοι οι επαγωγικοί κλάδοι είναι σε σειρά με ωμικούς μπορούμε να θεωρήσουμε σύνθετους ωμικούς-επαγωγικούς κλάδους.

Εφαρμόζουμε τον νόμο του Kirchoff:

$$\left[A_c|A_{gl}\right] * \begin{bmatrix} i_c(t) \\ i_{gl}(t) \end{bmatrix} = b_n(t) \Rightarrow A_c i_c(t) + A_{gl} i_{gl}(t) = b_n(t)$$

$$\Rightarrow i_c(t) + A_{gl} i_{gl}(t) = b_n(t) \qquad (5.1)$$

Όπου A_c: πίνακας πρόσπτωσης πυκνωτών μεγέθους n×n ο οποίος ταυτίζεται με τον μοναδιαίο άρα $A_c = I$ διότι κάθε κλάδος χωρητικότητας συνδεέται με την γή και προς την γή φεύγει το ρεύμα

Όπου A_{gl}:σύνθετος πίνακας πρόπτωσης αντιστάσεων-επαγωγών μεγέθους n×b

Για τον πίνακα A_{gl} ισχύει : $\begin{cases} 1, \text{όταν ο κλάδος } j \text{ φεύγει απο τον κόμβο } i \\ -1, \text{όταν ο κλάδος } j \text{ φθάνει στον κόμβο } i \\ 0, \text{αλλιώς} \end{cases}$

Όπου $b(t)$:διάνυσμα διεγέρσεων

Εφαρμόζουμε τον νόμο των τάσεων:

$$\begin{bmatrix} A_c^T \\ A_{gl}^T \end{bmatrix} * v_{nd}(\mathrm{t}) = \begin{bmatrix} v_c(t) \\ v_{gl}(t) \end{bmatrix} \Rightarrow \begin{cases} A_c^T v_{nd}(\mathrm{t}) = v_c(t) \Rightarrow v_{nd}(\mathrm{t}) = v_c(t) & (5.2) \\ A_{gl}^T v_{nd}(\mathrm{t}) = v_{gl}(t) & (5.3) \end{cases}$$

Σχέση ρεύματος-τάση:

$$i_c(t) = C_n v_C'(t) \Rightarrow i_c(t) = C_n v_{nd}'(t) \quad (5.4)$$

$$v_{gl}(t) = R_{br} i_{gl}(t) + L_{br} i_{gl}'(t) \quad (5.5)$$

Όπου R_{br} ο διαγώνιος πίνακας των αντιστάσεων των σύνθετων ωμικών-επαγωγικών κλάδων και L_{br} ο πίνακας επαγωγών (διαγώνιος εάν υπάρχουν μόνο αυτεπαγωγές στην οποία περίπτωση η τάση ενός κλάδου εξαρτάται από το ρεύμα του ίδιου του κλάδου ,δεν υπάρχει αλληλεξάρτηση μεταξύ κλάδων δηλαδή ο L_{br} θα γίνει πλήρης αν βάλουμε και αμοιβαίες επαγωγές μεταξύ των επαγωγικών κλάδων) στην περίπτωση μας τον πίνακα αυτό τον λαμβάνουμε σαν διαγώνιο

Αντικαθιστώντας την (5.4) στην (5.1) έχουμε:

$C_n v_{nd}'(t) + A_{gl} i_{gl}(t) = b_n(t)$ (5.6) στην οποία έχουμε n εξισώσεις με n+b αγνώστους $v_{nd}(t)$ καί $i_{gl}(t)$

Αντικαθιστώντας τη σχέση (5.3) στην (5.5) έχουμε :

$R_{br} i_{gl}(t) + L_{br} i_{gl}'(t) - A_{gl}^T v_{nd}(\mathrm{t}) = 0$ (5.7) στην οποία έχουμε b εξισώσεις με n+b αγνώστους $v_{nd}(t)$ καί $i_{gl}(t)$

Γράφοντας το σαν σύστημα έχουμε:

$$C_n v_{nd}'(t) + A_{gl} i_{gl}(t) = b_n(t)$$

$$R_{br} i_{gl}(t) + L_{br} i_{gl}'(t) - A_{gl}^T v_{nd}(\mathrm{t}) = 0$$

Αν γράψουμε τα διανύσματα μεταβλητών $v_{nd}(t)$: διάνυσμα τάσεων κόμβων καί $i_{gl}(t)$:διάνυσμα ρευμάτων κλάδων, ως νέο διάνυσμα:

$\tilde{x}(t) = \begin{bmatrix} v_{nd}(t) \\ i_{gl}(t) \end{bmatrix}$ οι δύο εξισώσεις γράφονται ώς:

$$\begin{bmatrix} 0 & A_{gl} \\ -A_{gl}^T & R_{br} \end{bmatrix} \tilde{x}(t) + \begin{bmatrix} C_n & 0 \\ 0 & L_{br} \end{bmatrix} \tilde{x}'(t) = \begin{bmatrix} b_n(t) \\ 0 \end{bmatrix}$$

Όπου A_{gl} : (n×b), R_{br} : (b×b), L_{br} : (b×b), C_n : (n×n)

$b_n(t)$:διάνυσμα διεγέρσεων από πηγές ρεύματος και από πηγές τάσης της μορφής $b_n(t) = -i_k(t) + G_0 V_{dd}$, όπου G_0ένας $q * p$ πίνακας αγωγιμοτήτων

Αντικαθιστώντας τους παρακάτω πίνακες με σύμβολα έχουμε:

$$\tilde{G} = \begin{bmatrix} 0 & A_{gl} \\ -A_{gl}^T & R_{br} \end{bmatrix}, \tilde{C} = \begin{bmatrix} C_n & 0 \\ 0 & L_{br} \end{bmatrix}, b(t) = \begin{bmatrix} b_n(t) \\ 0 \end{bmatrix}$$

$$\tilde{G}\tilde{x}(t) + \tilde{C}\tilde{x}'(t) = b(t)$$

Εάν οι πηγές ρεύματος (όπως συνήθως συμβαίνει) $i_k(t)$ έχουν προσδιοριστεί μέσω προσομοίωσης των block πυλών 1≤k≤n, ή απλά δεν επιθυμούμε να επιλύσουμε αναλυτικά το σύστημα διαφορικών εξισώσεων τότε η επίλυση του συστήματος θα πρέπει να γίνει αριθμητικά.

Συγκεκριμένα, ένα h είναι ένα σταθερό μικρό βήμα τότε το διάνυσμα παραγώγων $\tilde{x}'(t)$ γράφεται προσεγγιστικά:

$$\tilde{x}'(t) = \frac{\tilde{x}(t) - \tilde{x}(t-h)}{h}$$

Οπότε το σύστημα γράφεται:

$\left(\tilde{G} + \frac{\tilde{C}}{h}\right)\tilde{x}(t) = b(t) + (\frac{\tilde{C}}{h})\tilde{x}(t-h)$ για t=kh ,k=1,2,…,n έχουμε

$$\tilde{x}(kh) = \left(\tilde{G} + \frac{\tilde{C}}{h}\right)^{-1} b(kh) + \left(\tilde{G} + \frac{\tilde{C}}{h}\right)^{-1} (\frac{\tilde{C}}{h})\tilde{x}((k-1)h)$$

$$= D_1 b(kh) + D\tilde{x}((k-1)h) \quad (5.8)$$

Όπου $D_1 = \left(\tilde{G} + \frac{\tilde{C}}{h}\right)^{-1}$ και $D = \left(\tilde{G} + \frac{\tilde{C}}{h}\right)^{-1} (\frac{\tilde{C}}{h})$

Μέσω του παραπάνω συστήματος βρίσκουμε τις τάσεις και τα ρεύματα κλάδων άλλα για να το δούμε αυτό ποίο μαθηματικοποιημένα θα χρειαστεί να κάνουμε μία μετατροπή στο σύστημα των εξισώσεων.

Ποίο συγκεκριμένα:

Εάν θέσουμε $\tilde{x}(kh) = x^k$ και $b(kh) = b^k$ τότε το σύστημα (5.8) γράφεται :

$$\begin{bmatrix} {C_n}/{h} & A_{gl} \\ -A_{gl}^T & R_{br} + {L_{br}}/{h} \end{bmatrix} \begin{bmatrix} v_{nd}^k \\ i_{gl}^k \end{bmatrix} = \begin{bmatrix} {C_n}/{h} & 0 \\ 0 & {L_{br}}/{h} \end{bmatrix} \begin{bmatrix} v_{nd}^{k-1} \\ i_{gl}^{k-1} \end{bmatrix} + \begin{bmatrix} b_n^k \\ 0 \end{bmatrix}$$

Μετά από πράξεις καταλήγουμε στο παρακάτω σύστημα:

$$\begin{cases} \left(A_{gl}(R_{br} + {L_{br}}/{h})^{-1} A_{gl}^T + {C_n}/{h}\right) v_{nd}^k = \frac{C_n}{h}\, v_{nd}^{k-1} - A_{gl}(R_{br} + \frac{L_{br}}{h})^{-1} \frac{L_{br}}{h} i_{gl}^{k-1} + b_n^k \\ i_{gl}^k = (R_{br} + \frac{L_{br}}{h})^{-1} \frac{L_{br}}{h} i_{gl}^{k-1} + (R_{br} + \frac{L_{br}}{h})^{-1} A_{gl}^T v_{nd}^k \end{cases}$$

Όπως παρατηρούμε καταλήγουμε σε δύο αναδρομικές εξισώσεις .Η πρώτη εξίσωση του συστήματος είναι της μορφής Ax=b στην οποία οι άγνωστες τιμές του x είναι οι τιμές των τάσεων σε κάθε κόμβο του δικτύου.Σαν A θα έχουμε τον πίνακα :

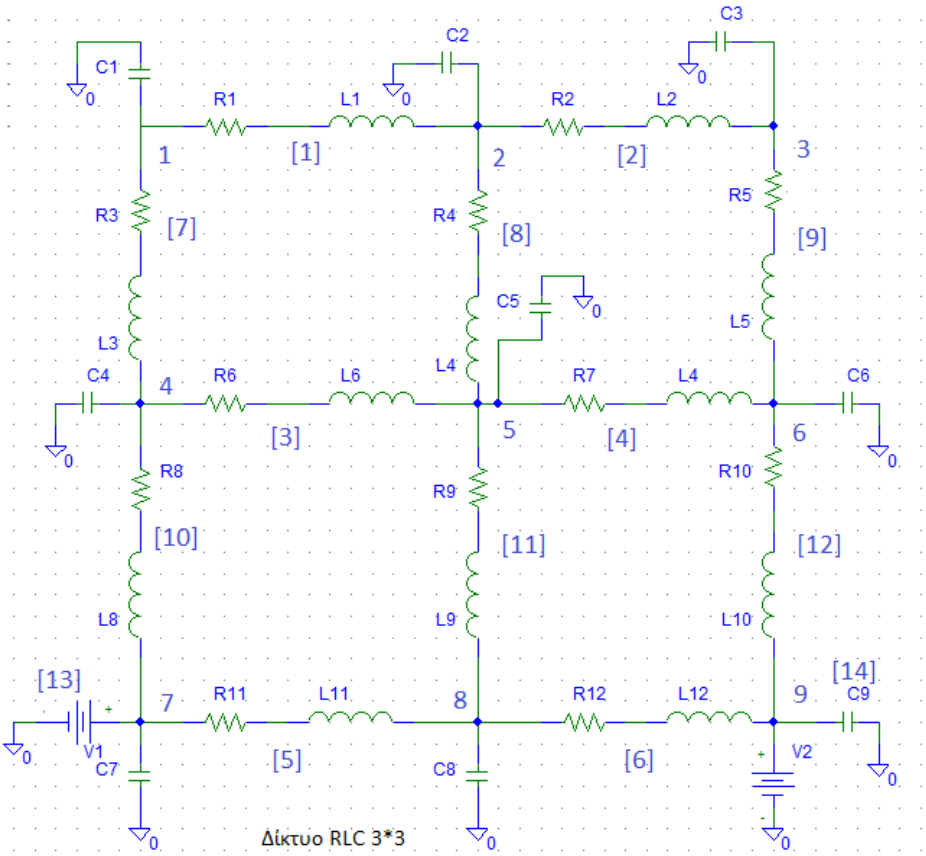
$$A = \left(A_{gl}(R_{br} + {L_{br}}/{h})^{-1} A_{gl}^T + {C_n}/{h}\right)$$

Ενώ σαν δίανυσμα b θα έχουμε:

$$b = \frac{C_n}{h}\, v_{nd}^{k-1} - A_{gl}(R_{br} + \frac{L_{br}}{h})^{-1} \frac{L_{br}}{h}\, i_{gl}^{k-1} + b_n^k$$

Από την στιγμή που βρούμε τις τιμές v_{nd}^k από την πρώτη εξίσωση μπορούμε να τις χρησιμοποιήσουμε για να βρούμε τα ρεύματα κατά μήκος των κλάδων του δικτύου βάση της δεύτερης αναδρομικής εξίσωσης.Το παραπάνω σύστημα μπορεί να λυθεί με οποιαδήποτε μέθοδο επίλυσης γραμμικών εξισώσεων η αναφορά των οποίων έγινε στο πρώτο κεφάλαιο.Ωστόσο η μέθοδος που θα χρησιμοποιήσουμε θα είναι αυτή του domain_decomposition(partitioning). Ο τρόπος με τον οποίο επιλύεται το παραπάνω σύστημα αναλύεται στο 5.2 κεφάλαιο .Στό επόμενο κεφάλαιο δίνεται ένα παράδειγμα της δομής του πίνακα A ενός δικτύου δομής πλέγματος.

5.1 Παράδειγμα Πρακτικού Δικτύου Τροφοδοσιας

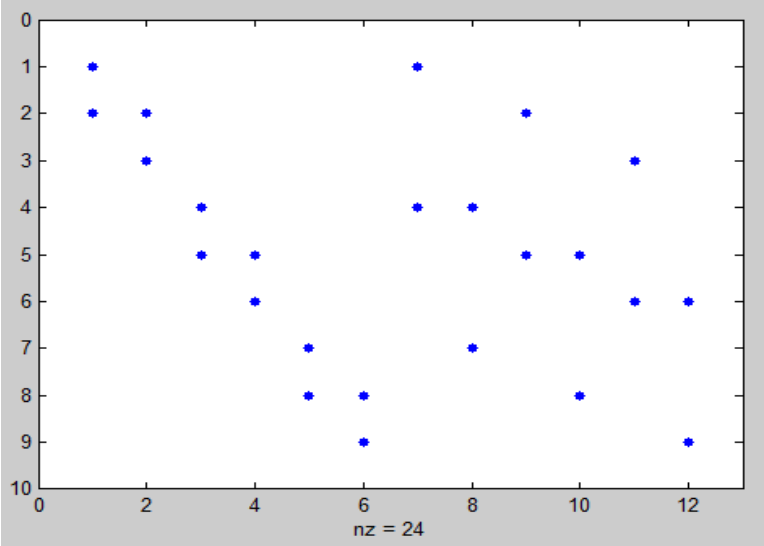


Σημείωση:

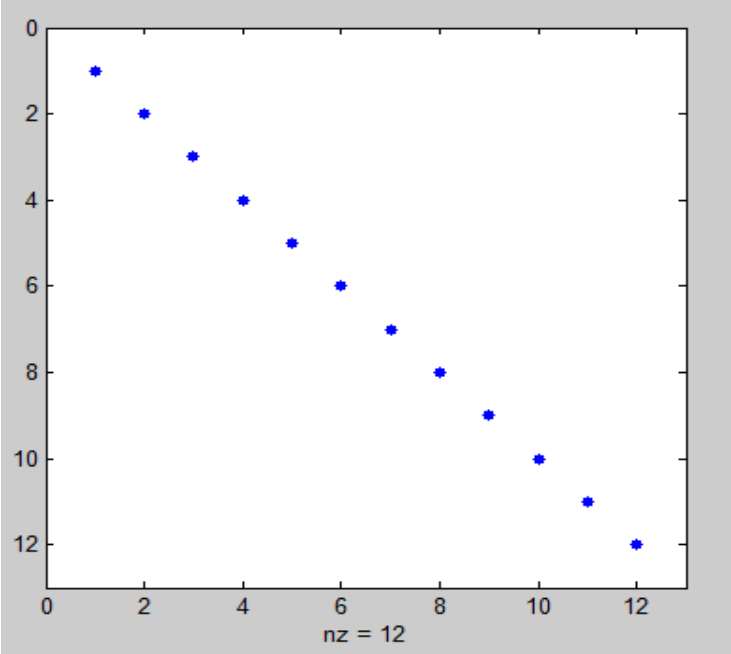
1. Για να μπορέσουμε να υπολογίσουμε τον πίνακα A_{gl} , πρέπει να ορίσουμε μία φορά για το ρεύμα που διαρρέει τους κλάδους. Έστω πως αυτή είναι από αριστερά προς τα δεξιά (→) και από πάνω προς τα κάτω (↓).

Αν η συχνότητα λειτουργίας του κυκλώματος είναι μεγάλη της τάξης των 10GHz τότε εμφανίζονται και παρασιτικές επαγωγές μαζί με τις παρασιτικές χωρητικότητες

Για ένα δίκτυο 3×3 ο ελαττωμένος πίνακας πρόσπτωσης A_{gl} θα είναι μεγέθους n×b όπου n: αριθμός των κόμβών b αριθμός των ακμών και συγκεκριμένα 9×12



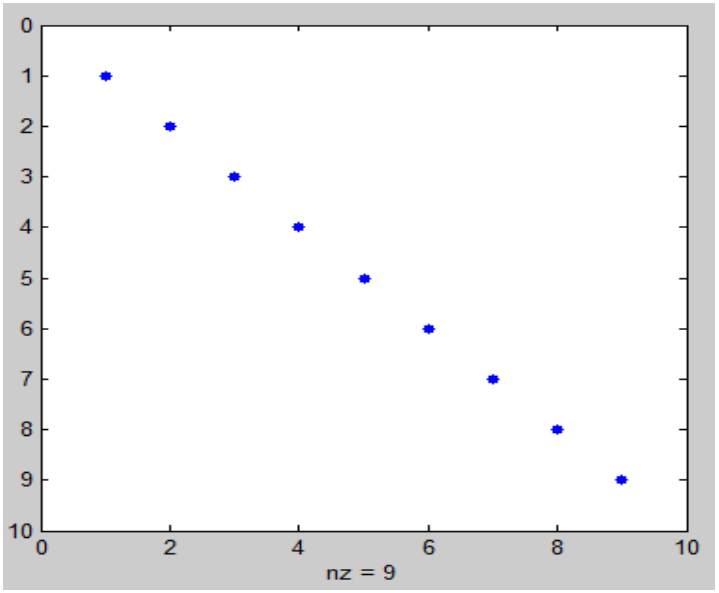
Ο πίνακας R_{br} είναι διαγώνιος μεγέθους $b \times b$



$$R_{br} = (R_b + {}^{L_b}/_{h})^{-1}$$

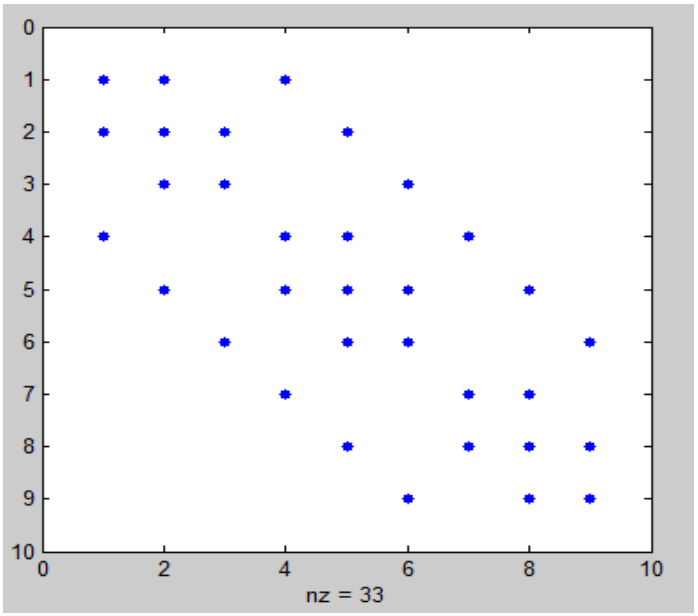
Ο πίνακας των χωρητικοτήτων είναι διαγώνιος μεγέθους n×n και έχει την μορφή:

C_9 =diag$(C_1, C_2, C_3, \dots, C_9)$



Ενώ ο πίνακας της τροποποιημένης μεθόδου των κόμβων είναι n×n:

$$A = A_{gl}(R_{br} + L_{br}/h)A_{gl}^{T} + C/h$$



Απο την θεωρία της μεθόδου των κόμβων γνωρίζουμε ότι για τον πίνακα $A'=A_{gl}R_{br}A_{gl}^{T}$:

1. Τα στοιχεία της κύριας διαγωνίου είναι θετικά ορισμένα και ισούνται με το άθροισμα των στοιχείων του κάθε κλάδου που προσπύπτει στον κόμβο i
2. Τα στοιχεία εκτός διαγωνίου είναι το αρνητικό άθροισμα των στοιχείων του κλάδου που συνδέει τον κόμβο i με τον κόμβο j $(-(R_b + {L_b}/{h})^{-1})$
3. Τα στοιχεία εκτός διαγωνίου είναι συμμετρικά δηλαδή $A'_{ij} = A'_{ji}$
4. Διαγώνια κυριαρχία και στις γραμμές και στις στήλες δηλαδή $|A'_{jj}| \geq \sum_{\substack{i=1 \\ i\neq j}}^{n} |A'_{ij}|, \forall j = 1, \dots, n$ η ιδιότητα αυτή εξασφαλίζει ότι ο πίνακας είναι θετικά ορισμένος

Ενώ η μορφή που έχει είναι η ακόλουθη:

$$\begin{matrix} g_1+g_7 & -g_1 & 0 & -g_7 & \vdots & 0 \\ -g_1 & g_1+g_2+g_8 & -g_2 & 0 & \vdots & 0 \\ 0 & -g_2 & g_2+g_9 & 0 & \vdots & 0 \end{matrix}$$

$$\begin{matrix} -g_7 & 0 & 0 & g_3+g_7+g_{10} & \vdots & 0 \\ \dots & \cdots & \cdots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & g_6+g_{12} \end{matrix}$$

Όπου $g_i = \frac{1}{r_i + \frac{l_i}{h}}$

5.2 Επίλυση μέσω της μεθόδου partitioning

Μία άλλη μέθοδος για την επιτάχυνση της λύσης μεγάλης κλίμακας κυκλωμάτων είναι η διχοτόμηση του κυκλώματος σε μικρότερης κλίμακας κυκλώματα .Η επίλυσης των επιμέρους υποκυκλωμάτων γίνεται σε ξεχωριστούς επεξεργαστές Η παραπάνω μέθοδος στοχεύει στην μείωση του χρόνου/χώρου υπολογισμού δυνητικά όταν δηλαδή χρησιμοποιούμε περισσότερους επεξεργαστες .Στην περιπτωσή μας αυτό δεν γίνεται διότι χρησιμοποιούμε έναν .

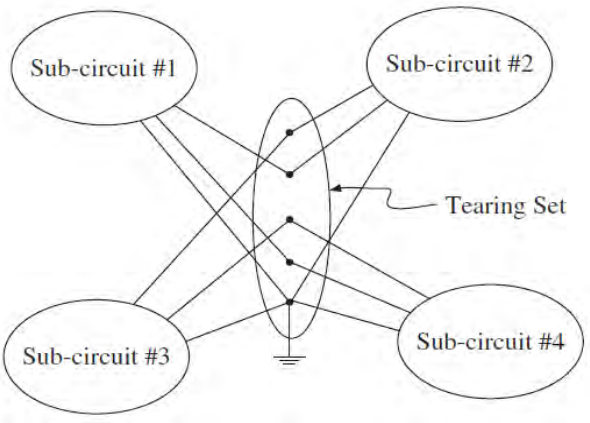
Αλγόριθμος

1.Ξεκινάμε με ένα συνεκτικό γράφημα

2. Βρίσκουμε ένα σύνολο κορυφών, το οποίο αν αφαιρεθεί μετατρέπει το γράφημα μας σε μη συνεκτικό: το σύνολο αυτών των κορυφών ονομάζεται tearing set ,μέσα σε αυτό το σύνολο περιλαμβάνουμε και την γείωση

3.Ανακαλύπτουμε τα sub-circuits από το μη κατευθυνόμενο γράφημα και κατασκευάζουμε το σύνολο των εξισώσεων για το κάθε ένα

4.Συνδυάζουμε αυτές τις εξισώσεις με τις εξισώσεις από το tearing-set



$$\begin{bmatrix} A_1 & 0 & \cdots & 0 & B_1 \\ 0 & A_2 & \cdots & 0 & B_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & A_m & B_m \\ C_1 & C_2 & \cdots & C_m & D_t \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_t \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \\ b_t \end{bmatrix} \quad (5.2)$$

Επίλυση μέσω άμεσης μεθόδου:

Η σχέση (5.2) περιγράφει το σύστημα των εξισώσεων της μεθόδου node tearing όπου A_i, B_i, C_i και D_t είναι πίνακες έτσι ώστε A_i είναι $n_i \times n_i$ (n_i είναι ο αριθμός των κόμβων στο i-subcircuit) κάθε B_i είναι $n_i \times n_t$ (n_t είναι ο αριθμός των κόμβων στο tearing set) C_i είναι $n_t \times n_i$,D_t είναι $n_t \times n_t$.Τα διανύσματα x και b χωρίζονται σε x_i και b_i αντίστοιχα.

$$\begin{cases} A_i x_i + B_i x_t = b_i \\ \sum_{i=1}^{m} C_i x_i + D_t x_t = b_t \end{cases} \qquad \forall i = 1,2,\dots,m \quad (5.3)$$

Επίλυση μέσω άμεσης μεθόδου:

Από την πρώτη εξίσωση του συστήματος (5.3) υπολογίζουμε:

$$x_i = A_i^{-1}(b_i - B_i x_t), \quad \forall i = 1,2,\dots,m \ (5.3.1)$$

$$(D_t - \sum_{i=1}^{m} C_i A_i^{-1} B_i) x_t = (b_t - \sum_{i-1}^{m} C_i A_i^{-1} b_i) \rightarrow$$

$$D_t^{*} x_t = b_t^{*} \ (5.3.2)$$

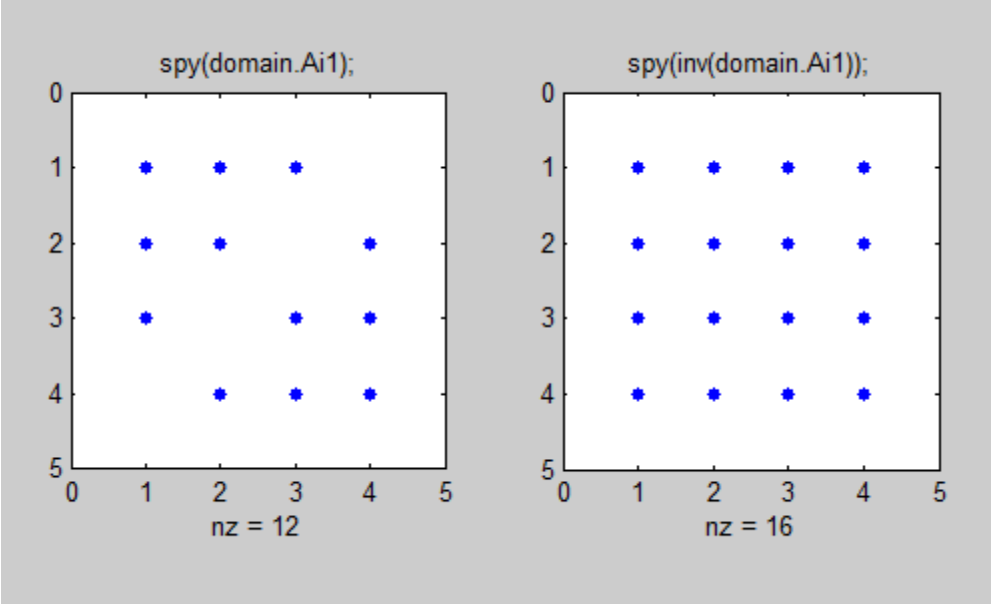
Όπου $D_t^{*} = (D_t - \sum_{i=1}^{m} C_i A_i^{-1} B_i)$ $\kappa\alpha\iota$ $b_t^{*} = (b_t - \sum_{i-1}^{m} C_i A_i^{-1} b_i)$

Βάση της (5.3.2) εάν λύσουμε ως προς x_t και εν συνεχεία αντικαταστήσουμε στην (5.3.1) μπορούμε να βρούμε τα στοιχεία x_i. Για τήν επίλυση αυτού του συστήματος χρειαζόμαστε τον αντίστροφο των υποπινάκων A_i .

Το βασικό πρόβλημα με την εξίσωση (5.3.2) είναι ότι αναστρέφοντας τους αραιούς υποπίνακες Ai οι αντίστροφοι πίνακες δεν είναι πλέον αραιοί όπως μπορούμε να δούμε στο παρακάτω διάγραμμα:

subplot(1,2,1); spy (domain.Ai{1}); title('spy(domain.Ai{1});');

subplot(1,2,2); spy (inv(domain.Ai{1}));title('spy(inv(domain.Ai{1}));');



Στην περίπτωση του διαγράμματος αυτό δεν είναι πάρα πολύ σημαντικό αλλά με μεγαλύτερους πίνακες οδηγεί σε πολύ περισσότερους υπολογισμούς κάτι ισχυρά ανεπιθύμητο. Γι'αυτό αντί για αναστροφή των πινάκων A_i καταφεύγουμε σε υπολογισμό με την μέθοδο LU.

Συγκεκριμένα, αναλύοντας σε Li|Ui τους πίνακες A_i έχουμε:

$$\begin{bmatrix} (L_1|U_1) & 0 & \cdots & 0 & L_1^{-1}B_1 \\ 0 & A_2 & \cdots & 0 & B_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & A_m & B_m \\ C_1U_1^{-1} & C_2 & \cdots & C_m & (D_t^{**}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_t \end{bmatrix} = \begin{bmatrix} L_1^{-1}b_1 \\ b_2 \\ \vdots \\ b_m \\ b_t - C_1A_1^{-1}b_1 \end{bmatrix}$$

$$D_t^{**} = (D_t - C_1A_1^{-1}B_1)$$

Να σημειώσουμε ότι τα A_i, B_i $και$ b_i γία i=2,3,…, m δεν μπορούν να τροποποιηθούν γιατί υπάρχει μηδενικός πίνακας στις αντίστοιχες σειρές ,στην στήλη κάτω από τον

A_1 .Ακόμη C_i γία i=2,3,… , m δεν μπορούν να τροποποιηθούν γιατί υπάρχει μηδενικός πίνακας στις αντίστοιχες στήλες ,στις γραμμές δεξιά του A_1.Η διαδικασία της τροποποίησης για $A_2, A_3, \dots, A_m$, για κάθε A_i ,είναι περιορισμένη για τα B_i, C_i, b_i, b_t και D_t στον βαθμό που μπορούν να παραλληλοποιηθούν.Έτσι θα έχουμε το παρακάτω σύστημα

$Ax = b$, όπου:

$$A=\begin{matrix} (L_1|U_1) & 0 & \cdots & 0 & L_1^{-1}B_1 \\ 0 & (L_2|U_2) & \cdots & 0 & L_2^{-1}B_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & (L_m|U_m) & L_m^{-1}B_m \\ C_1U_1^{-1} & C_2U_2^{-1} & \cdots & C_mU_m^{-1} & D_t^* \end{matrix}$$

$$x=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_t \end{bmatrix} \quad \text{και b} = \begin{bmatrix} L_1^{-1}b_1 \\ L_2^{-1}b_2 \\ \vdots \\ L_m^{-1}b_m \\ b_t^* \end{bmatrix}$$

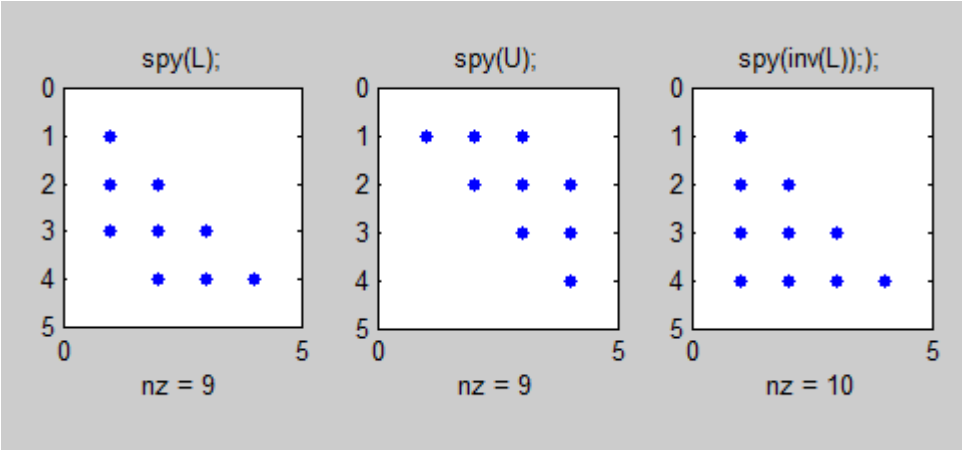
όπου τα D_t^*, b_t^* έχουν υπολογιστεί παραπάνω

$$L_iU_ix_i = b_i - B_ix_t$$

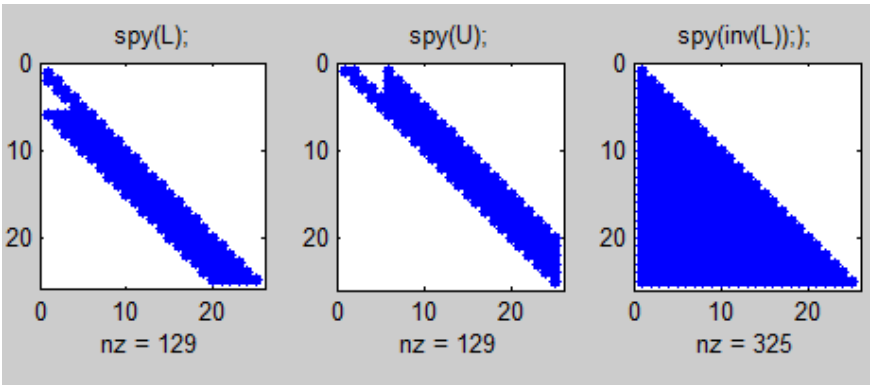
$$U_ix_i = L_i^{-1}b_i - L_i^{-1}B_ix^t \quad (5.3.3)$$

Ο συνολικός χρόνος είναι $O(n_t^3 + \sum_{i=1}^{m} n_i^3)$, ο οποίος είναι πολύ μικρότερος από $O((n_t + \sum_{i=1}^{m} n_i)^3)$. Πράγματι, σύμφωνα με την εξίσωση (5.3.3) μπορούμε να λύσουμε τις επιμέρους λύσεις με πολύ πιό απλούς υπολογισμούς σε σχέση με την (5.3.2). Το L_i^{-1} είναι διαγώνιο και προ-υπολογισμένο. Επίσης μπορούμε να προ-υπολογίσουμε το $L_i^{-1}B_i$. Τέλος είναι εύκολο να προ-υπολογίσουμε το U_i και να εκτελέσουμε backwards substitution στα τελικά νούμερα. Τόσο ο U_i, όσο και ο L_i^{-1}, $L_i^{-1}B_i$ είναι αραιοί πίνακες και δεν χρειάζεται forward substitution γιατί είναι ήδη διαγώνιοι όπως φαίνεται στο παρακάτω διάγραμμα.

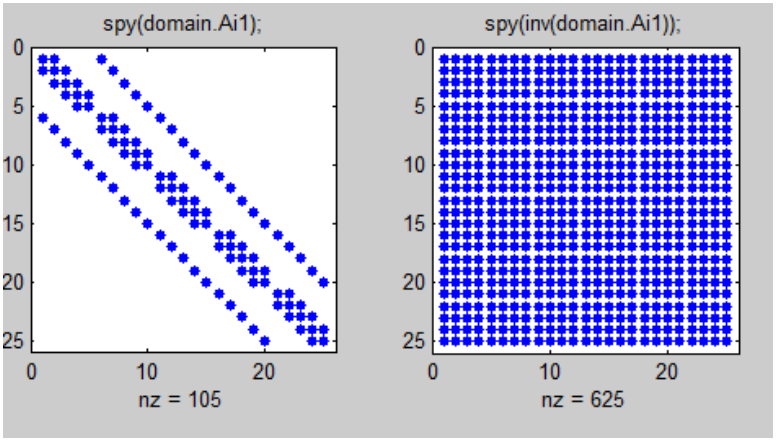
```
[L,U,P] = lu(domain.Ai{1});

subplot(1,3,1); spy (L); title('spy(L);');

subplot(1,3,2); spy (U); title('spy(U);');

subplot(1,3,3); spy (inv(L)); title('spy(inv(L)););');
```



Προφανώς ακόμη πιο εντυπωσιακές είναι οι διαφορές αν έχουμε πολύ μεγαλύτερα πλέγματα π.χ. 11x11:

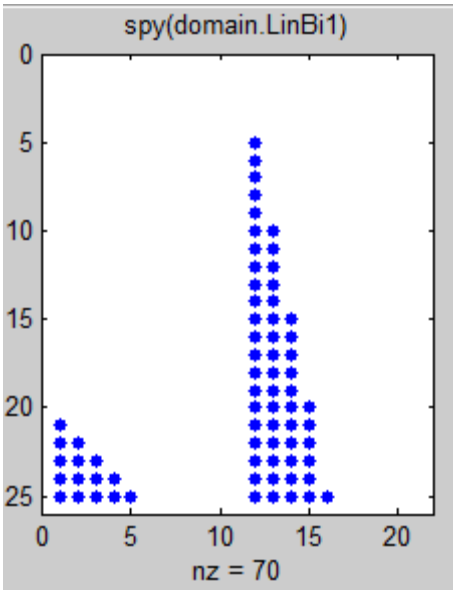


Σε αντίθεση με τη μορφή της εξίσωσης (5.3.2):



Επίσης ακόμη και το $L_i^{-1}B_i$ είναι αρκετά αραιό όπως βλέπουμε στο παρακάτω διάγραμμα:

```
spy(domain.LinBi{1})
 title('spy(domain.LinBi{1})');
```



Συνεπώς βλέπουμε ότι υπάρχουν αρκετά πλεονεκτήματα στο να λύσουμε τα επιμέρους αραιά συστήματα με βάση τις εξισώσεις (5.3.3) αντι για τις (5.3.2). Εμείς υλοποιήσαμε και τις δύο έτσι ώστε να μπορεί αν χρειαστεί να γίνει κάποια σύγκριση.

5.3 Ο κώδικας σε Matlab

Ο κώδικας σε Matlab ακολουθεί αρκετά πιστά τις παραπάνω εξισώσεις. Έχουμε χωρίσει τον κώδικα σε δύο μέρη, τον προ-υπολογισμό των πινάκων του δικτύου που χρειάζεται να γίνει μία μόνο φορά (συνάρτηση d_prepare) για όλο το δίκτυο και τους υπολογισμούς που χρειάζεται να γίνουν κάθε φορά (συνάρτηση d_solve) – για κάθε διαφορετικό διάνυσμα b δηλαδή τον υπολογισμό του x_t και τον υπολογισμό των x_i's σύμφωνα με την εξίσωση 5.3.2 ή 5.3.3. Επίσης πρέπει να επισυμανθεί ότι αν θέλαμε να τρέξουμε παράλληλα σε πολλούς επεξεργαστές θα έπρεπε απλά να αναθέσουμε κάθε έναν από τους υπολογισμούς των x_i's της d_solve σε διαφορετικό επεξεργαστή. Όλος ο υπόλοιπος κώδικας συμπεριλαμβανομένου και του υπολογισμού του x_t δεν παραλληλίζεται.

Παρακάτω βλέπουμε τον κώδικα της "test", μίας procedure που εκτελεί όλους τους υπολογισμούς και μας δίνει ενδεικτικά αποτελέσματα στην οθόνη. Μπορούμε να την εκτελέσουμε στο Matlab γράφοντας απλά test. Βλέποντας αυτή τη procedure μπορούμε να καταλάβουμε καλύτερα πώς δουλεύει ο όλος αλγόριθμος.

```
clear;
gridsparse();

if 0
    %Proairetiki dimiourgia tixaiou r
    r = rand(length(A),1)*100;
end

if 0
    %Grapsimo tou r kai tou a se csv's & mat file
    save 'r.mat' r;
    csvwrite('r.csv',r);
    [i,j,val] = find(A);
    csvwrite('A.csv', [i,j,val]);
elseif 1
    %diavasma tou r apo mat file
    load 'r.mat'
end

hdom = 2; % 2 orizontai domains
vdom = 2; % 2 katakorifa domains

domain = d_prepare( A, hdom, vdom, n1, n2);

%emfanisi ton dio pinakon, A kai AP
if 0
    subplot(1,2,1);
    spy(A);
    subplot(1,2,2);
    spy(domain.AP);
end
```

```
%emfanisi ton pinakon - Ai,Bi,Ci
if 0
    figure();
    d_plot_matrix(domain)
end

% lisi mias tixeas periptosis
[rx,bi] = d_solve( domain, r );

%ipologismos sfalmatos se sxesi me apli diairesi tou Matlab
norm(A\r - rx)
```

Το πρώτο πράγμα που κάνουμε είναι να καλούμε την clear. Αυτή μας καθαρίζει ότι μεταβλητές μπορεί να έχουν μείνει στην μνήμη του Matlab από προηγούμενες εκτελέσεις. Το δεύτερο βήμα είναι να καλέσουμε την gridsparse. Αυτή η procedure μας έχει δοθεί και υπολογίζει τον πίνακα A που χαρακτηρίζει το σύστημα του δικτύου τροφοδοσίας. Το διάνυσμα r που μας δίνεται από τη gridsparse είναι πολύ απλό (έχει όλα μηδενικά οπότε λίγο παρακάτω μας δίνεται η δυνατότητα να υπολογίσουμε ένα άλλο τυχαίο διάνυσμα r (αν το if έχει μη μηδενική τιμή ως συνθήκη – κάτι που μπορούμε να αλλάξουμε με το χέρι).

Λίγο παρακάτω έχουμε την επιλογή να σώσουμε το διάνυσμα r όσο και τον πίνακα A σε csv αρχείο και το διάνυσμα r σε αρχείο τύπου .mat. Σώζωντας τους πίνακες αυτούς μπορούμε μετά να τους ανακτήσουμε από την C και να ελέγξουμε τα αποτελέσματα των υπολογισμών μας στις δύο γλώσσες. Εναλλακτικά, θέτοντας 1 στο elseif μπορούμε να διαβάσουμε το διάνυσμα r από το αρχείο .mat έτσι ώστε να έχει την ίδια τιμή που δίνεται στο csv αρχείο.

Όπως και να’χει μέχρι αυτό το σημείο ο κώδικας σχετίζεται με τη δημιουργία, αποθήκευση και ανάκτηση των μεταβλητών που αφορούν το σύστημα. Ο σημαντικός κώδικας ο οποίος εκτελεί τους υπολογισμούς αρχίζει παρακάτω.

Οι μεταβλητές των hdom και των vdom καθορίζουν τον αριθμό των οριζόντιων και των κάθετων χωρισμάτων του χώρου σε subdomains. Ο αριθμός αυτός σχετίζεται με το μέγεθος του δικτύου μιας και θα πρέπει ο αριθμός των κόμβων να "βγαίνει" όταν διαιρούμε το δίκτυο. Για να γίνεται αυτό θα πρέπει να είναι περιττός ως προς τις διαιρέσεις με hdom και vdom. Για παράδειγμα αν έχουμε ένα δίκτυο 5 x 5 μπορούμε να το χωρίσουμε σε 2x2 subdomains αφού θα είναι 5/2 -> [2 **1** 2] δηλαδή 2 κόμβοι, ένας του tearing σετ και άλλοι δύο κόμβοι του δεύτερου subdomain. Το ίδιο και κατακόρυφα. Θα μπορούσε επίσης να χωριστεί σε 3x3 subdomains μιας και θα ήταν 5/3 ->

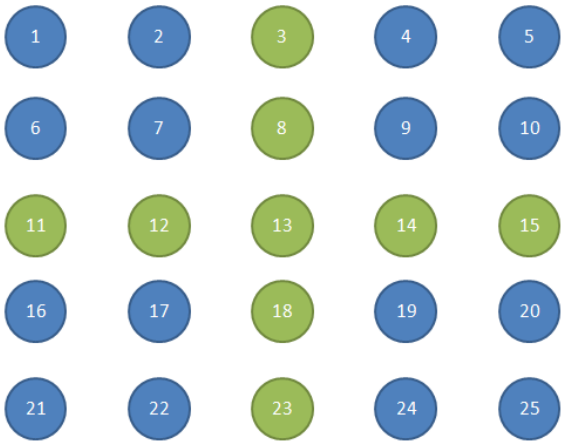
[1 **1** 1 **1** 1] δηλαδή ένας κόμβος ανά subdomain και 2, ένας για κάθε χώρισμα στο tearing set. Σε ένα 7x7 δίκτυο μπορεί να γίνει χωρισμός 2x2 = 4 subdomains γιατί 7/2-> [3 **1** 3] δηλαδή 3 κόμβοι σε κάθε subdomain και ένας στο tearing set. Παρόλα αυτά δεν γίνεται διαίρεση σε 3x3 7/3 -> [2 **1** 2 **1** 1] δηλαδή μας λείπει ένας για το τελευταίο subdomain αλλά γίνεται σε 4x4 γιατί αντίστοιχα 7/4 -> [1 **1** 1 **1** 1 **1** 1] δηλαδή 1 κόμβος σε κάθε subdomain και από ένα στο tearing set. Πιο συγκεκριμένα αν δούμε τι γίνεται με το 7, θα καταλάβουμε λίγο καλύτερα πώς γίνεται ο χωρισμός. 7/2=3.5 το οποίο σημαίνει ότι το 7 είναι περιττός και υπάρχουν 3 κόμβοι όταν χωρίσουμε σε 2 subdomains. Τώρα χωρίζοντας κάθε ένα από τα subdomains στη μέση έχουμε 3/2=1.5 δηλαδή και πάλι είναι περιττός και έχουμε έναν κόμβο ανά subdomain. Αυτό όμως δεν συμβαίνει στα 9x9 αν θέλαμε π.χ. να τα χωρίσουμε σε 4x4 subdomains γιατί 9/2 = 4.5 οπότε μπορούμε πράγματι να τα χωρίσουμε σε 2x2 αλλά 4/2 = 2 το οποίο δεν είναι περιττό. Συνεπώς δεν μπορούμε να χωρίσουμε και πάλι τα subdomains σε υποsubdomains.

Όλοι αυτοί οι υπολογισμοί γίνονται μέσα στην επόμενη σειρά του προγράμματος και συγκεκριμένα μέσα στη d_prepare.

Στην αρχή της d_prepare μπορούμε να δούμε μία κλήση σε μία άλλη συνάρτηση, την συνάρτηση:

```
[d.prem, d.xperm, d.tearsz] = perm(n1, hdom, vdom);
```

Η οποία μας δημιουργεί τους πίνακες prem και xperm που αναδιατάσσουν τους πίνακες του προβλήματος. Για το παρακάτω δίκτυο π.χ.



ο πίνακας perm θα πρέπει να είναι: [1 2 6 7 4 5 9 10 16 17 21 22 19 20 24 25 3 8 11 12 13 14 15 18 23]

Παρατηρούμε λοιπόν εναλλάξ τους indices των κόμβων του πρώτου subdomain, του $2^{ου}$, του $3^{ου}$, του $4^{ου}$ και τέλος τους 9 κόμβους του tearing set.

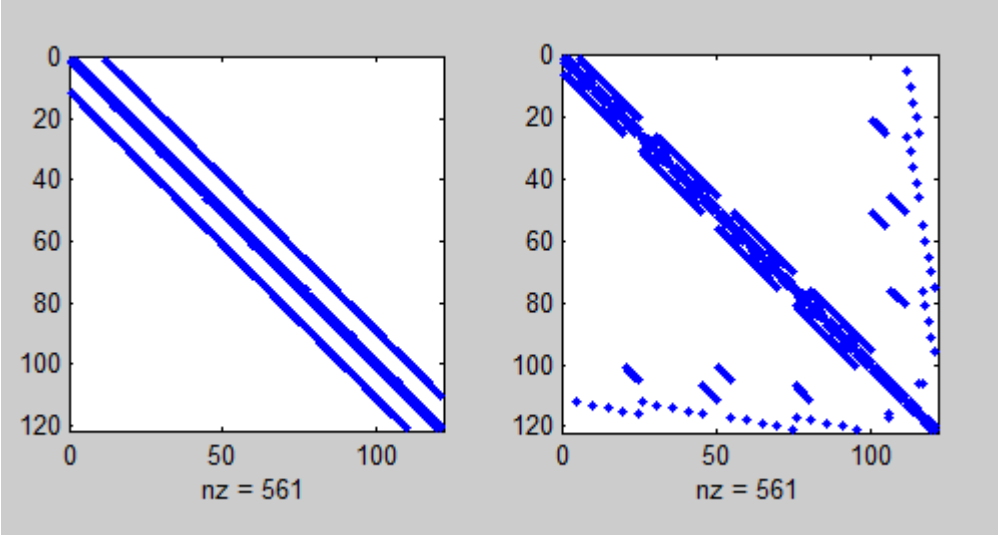
Αμέσως μετά μέσα στην d_prepare ο πίνακας A μετασχηματίζεται στον αναδιατεταγμένο πίνακα Ap:

d.AP = A(d.prem,d.prem);
Το ίδιο γίνεται αργότερα και για τον πίνακα b στην d_solve και στο τέλος, μέσα στη d_solve η λύση περνάει πίσω στην κανονική σειρά του πίνακα A:

rx = rx(d.xperm);
χρησιμοποιώντας τον πίνακα xperm που είναι ο αντίστροφος πίνακας indices του perm:

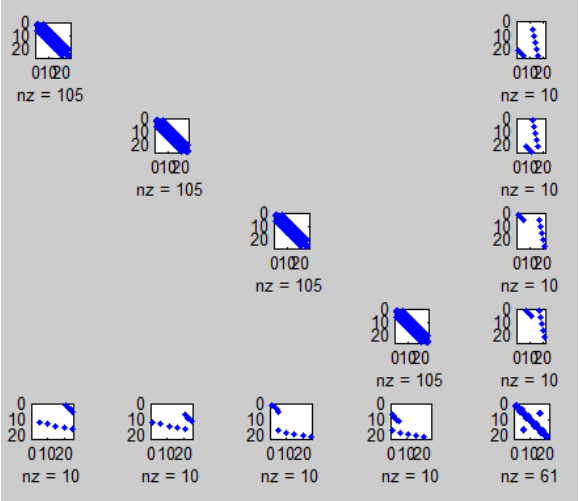
xperm(prem)=1:length(prem);
Μετά στην d_prepare γίνονται όλοι οι υπολογισμοί με βάση τις εξισώσεις που είδαμε παραπάνω (5.3).

Στην test λίγο παρακάτω από την d_prepare έχουμε την πιθανότητα να ενεργοποιήσει κανείς την εμφάνιση του A και του Ap. Αν είναι 1 τότε θα εμφανιστεί κάτι σαν το εξής στην έξοδο:



Δείχνοντας τον αρχικό πίνακα και τον αναδιατεταγμένο πίνακα (στην παραπάνω περίπτωση 11x11 χωρισμένο σε 2x2 subdomains).

Λίγο παρακάτω μας δίνεται η δυνατότητα να επιλέξουμε την εμφάνιση των υποπινάκων A_i,B_i,C_i σε ένα διάγραμμα. Αν αυτό είναι ενεργοποιημένο (1) τότε θα έχουμε κάτι σαν την παρακάτω έξοδο για το παράδειγμα του 11x11 πίνακα που μόλις εξετάσαμε:



Γίνεται προφανές τόσο το πόσος πολύς χώρος με μηδενικά απελευθερώνεται με αυτόν τον τρόπο το οποίο μεταφράζεται σε σώσιμο επεξεργαστικής ισχύος.

Τέλος στο test έχουμε τον υπολογισμό του διανύσματος rx (της λύσης) με την κλήση της d_solve:

```
[rx,bi] = d_solve( domain, r );
```

και ελέγχουμε στο τέλος την ακρίβεια της εξόδου με τη βοήθεια του υπολογισμού μίας νόρμας

```
norm(A\r - rx)
```

η οποία συνήθως έχει εξαιρετικά μικρή τιμή. Να επισημάνουμε εδώ ότι η διαίρεση A\r προφανώς είναι γρήγορη για μικρά δίκτυα αλλά για μεγάλα δίκτυα θα παίρνει σημαντικό χρόνο (εξου και η όλη μέθοδος που αναπτύσσουμε σε αυτή τη διπλματική).

Ο αναλυτικός κώδικας υπάρχει στο παράρτημα Γ και έπειτα από τις παραπάνω διευκρυνίσεις δεν πιστεύουμε ότι θα υπάρχει ιδιαίτερο πρόβλημα στην κατανόησή του.

5.4 Ο κώδικας σε C με τη βοήθεια της βιβλιοθήκης CSparse

Σε μεγάλο βαθμό ο κώδικας σε C ακολουθεί τον κώδικα του Matlab. Το πρόβλημα είναι ότι παρόλο που η βιβλιοθήκη CSparse είναι αρκετά πλούσια, της λείπουν ορισμένα βασικά στοιχεία τα οποία χρειάστηκε να υλοποιήσουμε από το 0 σε C.

Αυτά είναι:

- Μετατροπή διανύσματος σε αραιό πίνακα:

```
cs* vec_to_sparse(double*v, int nsize);
```

- Μετατροπή αραιού πίνακα σε διάνυσμα:

```
double* sparse_to_vec(cs*A);
```

- Εξαγωγή εύρους (αντίστοιχο της σύνταξης A(1:n,1:m) του Matlab):

```
cs* mat_extract(cs*A, int fx, int tx, int fy, int ty);
```

- Αναστροφή πίνακα:

```
cs * inv(cs*A);
```

Η αναστροφή πίνακα γίνεται λύνοντας την εξίσωση $A * A^{-1} = I$ – για κάθε στήλη με τη βοήθεια της cs_lusol. Συνεπώς στην αρχή λύνουμε την (χρησιμοποιώντας τους συμβολισμούς του Matlab) $A * A^{-1}(1,:) = I(1;0)$ ως προς $A^{-1}(1,:)$, μετά την $A * A^{-1}(2,:) = I(2;0)$ ως προς $A^{-1}(2,:)$ κ.ο.κ. Παράλληλα συνθέτουμε τον τελικό πίνακα A^{-1} από τις επιμέρους λύσεις $A^{-1}(1,:)$, $A^{-1}(2,:)$ κ.ο.κ.

Η υλοποίηση όλων αυτών των συναρτήσεων βρίσκεται μέσα στο ddcomp.c όπου βρίσκονται και οι εξής επιπλέον συναρτήσεις που είναι διαθέσιμες μέσω του ddcomp.h:

```
//Proetoimasia domis domain_desc gia to sistima
domain_desc d_prepare( cs* A, int hdom, int vdom,  int n1, int n2);
```

```
//Epilisi gia dedomeno b
double* d_solve( domain_desc d, double * b);

//Anastrofi pinaka A me tin xrisi LU
cs * inv(cs*A);

//Epeleutherosi mnimis tou domain
void free_domain(domain_desc * d);
```

Επίσης πρέπει να αναφέρουμε δύο τεχνάσματα που χρήζουν διευκρύνισης. Το πρώτο είναι σχετικά με την χρήση της συνάρτησης cs_permute. Η χρήση της μας δίνει τη δυνατότητα να εξάγουμε μία αναδιάταξη ενός πίνακα όπως κάναμε στο Matlab με το A(prem,prem). Για να κάνουμε το ίδιο με τη βιβλιοθήκη CSparse χρειάζεται να κάνουμε:

```
d.AP = cs_permute(A, d.xperm, d.prem, 1);
```

δηλαδή χρειάζεται να χρησιμοποιήσουμε τα διανύσματα τόσο του ορθού όσο και του ανάστροφου μετασχηματισμού. Επίσης δεν υπάρχει συνάρτηση που να κάνει αντιγραφή πίνακα. Γι'αυτόν τον λόγο χρησιμοποιήσαμε την συνάρτηση cs_add η οποία μας προσθέτει δύο πίνακες. Κάνοντας:

```
d.Ui[i] = cs_add(lun->L, lun->L, 1.0, 0);
```

Έχουμε ένα αντίγραφο του πίνακα L μέσα στο $U_i[i]$ γιατί ο τελικός πίνακας θα έχει 1 * L + 0 * L = L. Προφανώς αυτό σημαίνει κάποιο υπολογιστικό κόστος αλλά δεν πειράζει για τον μικρό αριθμό επαναλλήψεων όπου το χρησιμοποιούμε.

Τέλος σε ορισμένες περιπτώσεις οι ορισμοί πινάκων είναι σε ανάστροφη σειρά π.χ. η συνάρτηση cs_lu μας δίνει άνω τριγωνικό στο L και κάτω τριγωνικό στον U. Αυτό πιθανόν είναι λόγω διαφορετικών τρόπων αρίθμησης στο matlab και στην CSparse

και έχει χρειαστεί σε κάποιες περιπτώσεις π.χ. να αλλάξουμε τη σειρά των πολλαπλασιασμών στην cs_multiply όπου cs_multiply(B,A) δίνει A*B όπως επίσης και στην d_solve να χρησιμοποιήσουμε cs_lusol(LU_ORDER, **cs_transpose** (d.Ui[i], 1), xi[i], LU_TOLERANCE); αντί για cs_lusol(LU_ORDER, d.Ui[i], xi[i], LU_TOLERANCE); ώστε να έχουμε τα επιθυμητά αποτελέσματα. Παρόλα αυτά με τον συγκεκριμένο αριθμό ανακατατάξεων, το πρόγραμμά μας δίνει τα ίδια αποτελέσματα με το Maltab (με εξαίρεση τα λάθη στρογγυλοποίησης).

Σχετικά με τα λάθη στρογγυλοποίησης εκεί που φαίνεται να υπάρχει το μεγαλύτερο πρόβλημα είναι στην επίλυση συστημάτων μέσω της cs_lusol. Εκεί η ακρίβεια περιορίζεται σε 0.001 (LU_TOLERANCE) και δεν μπορούμε να την αυξήσουμε με κάποιον εύκολο τρόπο. Αν θελήσουμε να έχουμε πιο ακριβή αποτελέσματα αρκεί σε πρώτη φάση να αυξήσουμε την ακρίβεια της cs_lusol.
Επιπλέον η CSparse δεν μας δίνει συναρτήσεις για ανάγνωση πινάκων από αρχεία και γι'αυτό δημιουργήσαμε δύο συναρτήσεις (στα read_mat.h/.c):

```
// Gia kanonikous pinakes:
// csvwrite('ap.csv',full(domain.AP));

double* read_matrix(char * file, int *width, int *height);


// Gia araious pinakes:
// [i,j,val] = find(domain.AP);
// csvwrite('ap-sparse.csv', [i,j,val]);

cs* read_sparse(char * file);
```

Με τη βοήθεια αυτών των συναρτήσεων μπορούμε να διαβάσουμε πίνακες οι οποίοι εξάγονται από το matlab με τη βοήθεια των εντολών που βλέπουμε στα σχόλια από πάνω από την κάθε συνάρτηση. Με αυτόν τον τρόπο μπορούμε να εισάγουμε στην C τους πίνακες του δικτύου έτσι όπως τους δημιουργεί το Matlab και να ακολουθήσουμε τον τρόπο επίλυσης του node tearing.

Αν δούμε το αρχείο main.c θα παρατηρήσουμε στη συνάρτηση main τον τρόπο με τον οποίο χρησιμοποιούνται οι συναρτήσεις για την επίλυση του συστήματος:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

#include "read_mat.h"
#include "ddcomp.h"

int main()
{
    int rw, rh, i;
    double* r = read_matrix("r.csv", &rw, &rh);
    cs* A = read_sparse("A.csv");

    int hdom = 2; // 2 orizontai domains
    int vdom = 2; // 2 katakorifa domains

    int n = (int) sqrt(A->m);

    domain_desc domain = d_prepare(A, hdom, vdom, n, n);
    double* rx = d_solve( domain, r);

    printf("rx_=[");
    for (i = 0; i < n*n; i++)
        printf("%f%s", rx[i], i==(n*n-1) ? "" : ";");
    printf("];\n");

    free_domain(&domain);
    cs_spfree(A); free(r); free(rx);

    return 0;
}
```

Παρατηρούμε ότι είναι αντίστοιχος με τον τρόπο που παρουσιάστηκε στο Matlab. Πρώτα διαβάζουμε τους πίνακες A και r από τα αρχεία A.csv και r.csv αντίστοιχα. Μετά θέτουμε την τιμή του hdom και του vdom στις τιμές του αριθμού των οριζοντίων/καθέτων subdomains που θέλουμε να χωρίσουμε και μετά καλούμε τις συναρτήσεις d_prepare και d_solve για την προετοιμασία και την επίλυση του συστήματος. Αμέσως μετά τυπώνουμε το αποτέλεσμα σε μορφή που μπορεί να γίνει κατευθείαν copy-paste στο Matlab ώστε να μπορούμε να συγκρίνουμε τα αποτελέσματα. Αν παρακολουθήσουμε τον κώδικα θα δούμε ότι υπάρχει σχεδόν 1-1 αντιστοιχία με τον κώδικα του Matlab και σε πολλές περιπτώσεις ο αντίστοιχος κώδικας Matlab βρίσκεται σε σχόλια από πάνω από τον αντίστοιχο κώδικα σε C. Ο κώδικας βρίσκεται στο παράρτημα Δ.

* Να σημειώθεί ότι τόσο στην C όσο και στο Matlab όπου αναφέρεται αριθμός εξίσωσης αυτός είναι σύμφωνος με το μέρος 3.4.1 του βιβλίου Circuit Simulation του Farid N. Najm [1] το οποίο χρησιμοποιήθηκε ως σύγγραμμα αναφοράς.

6.Προσομοίωση παραδείγματος

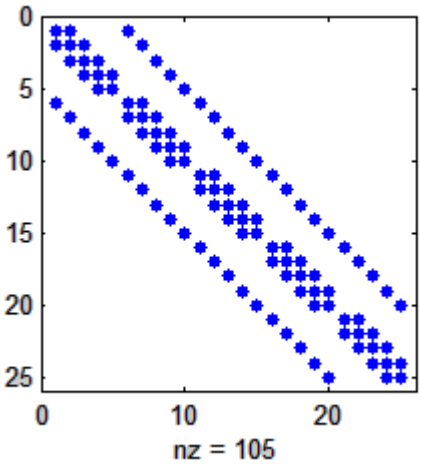
Για την καλύτερη κατανόηση της μεθόδου partitioning θα προσομοιώσουμε ένα δίκτυο τροφοδοσίας με τα εξής χαρακτηριστικά:

n_1	αριθμός κόμβων ανά γραμμή
n_2	αριθμός κόμβων ανά στήλη
vn=2	αριθμός των πηγών τροφοδοσίας του δικτύου
rsh=0.1	αντίσταση οριζόντιου φύλου(σε Ohms/sq)
rsv=0.1	αντίσταση κατακόρυφου φύλου(σε Ohms/sq)
lsh=0	οριζόντια επαγωγή ανά μονάδα μήκους (σε pH/um)
lsv=0	κατακόρυφη επαγωγή ανά μονάδα μήκους (σε pH/um)
sizv=350	κατακόρυφη διάσταση του ολοκληρωμένου κυκλώματος(σε um)
sizh=350	οριζόντια διάσταση του ολοκληρωμένου κυκλώματος (σε um)
pv=sizh/(n1-1)	αριθμός των κάθετων γραμμών
ph=sizv/(n2-1)	αριθμός των οριζόντιων γραμμών
rpin=50	αντίσταση πηγής τροφοδοσίας σε Ohms
lpin=100	αυτεπαγωγή πηγής τροφοδοσίας (σε pH)
cpph=1e-4	χωρητικότητα παράλληλων πλακών στους οριζόντιους κλάδους σε pF/um^2
cppv=1e-4	χωρητικότητα παράλληλων πλακών στους κάθετους κλάδους σε pF/um^2
cffh=1e-4	πλευρική χωρητικότητα στους οριζόντιους κλάδους σε pF/um
cffv=1e-4	πλευρική χωρητικότητα στους κάθετους κλάδους σε pF/um
cpin=1	χωρητικότητα πηγής τροφοδοσίας σε pF
h=10	βήμα δειγματοληψίας (σε ps)
w=ones(n2+n1,1)	διάνυσμα πλάτους των οριζόντιων και κάθετων αγωγών

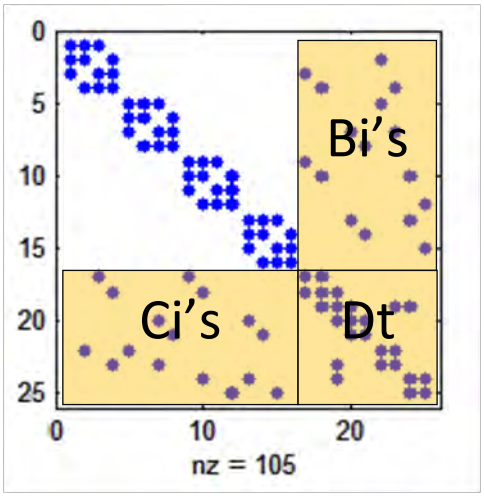
Το συγκεκριμένο δίκτυο έχει n=n1×n2 πλήθος κόμβων και b=n1*(n2-1)+n2*(n1-1) πλήθος κλάδων για λόγους ευκολίας παραλείπουμε τις ακμές των τάσεων τροφοδοσίας.Οι δοκιμαστικές εκτελέσεις έγιναν για διάφορες τιμές των n1 και n2.

Ακόμη για λόγους ευκολίας θέτουμε σε όλους τους κόμβους τιμή πυκνωτών ίση με 1 για τον ίδιο λόγο θέτουμε και τον πίνανα των αυτεπαγωγών ίσο με 0.Τίς τιμές αυτές μπορούμε να τις δούμε μέσα από τον κώδικα του παραρτήματος Γ στο κεφάλαιο 11 στην συνάρτηση gridsparse.

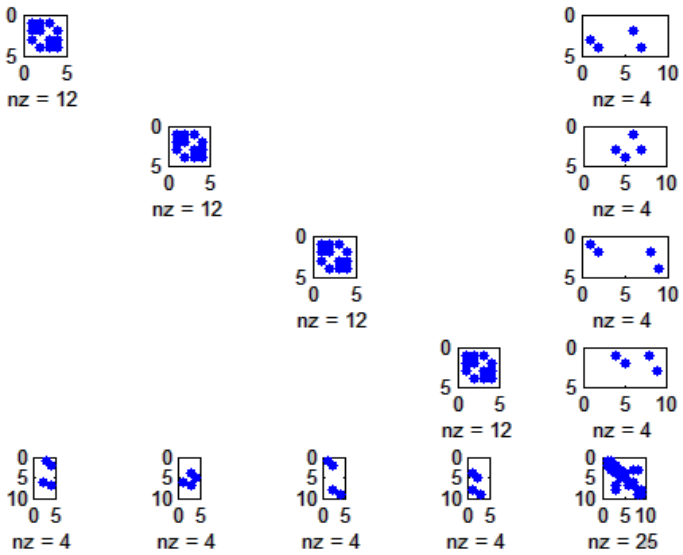
Για n1=n2=5 κατασκευάζοντας ένα δίκτυο 2×2 ο αριθμός των domains είναι 4 ο αρχικός πίνακας που σχηματίζεται πρίν γίνει το partitioning έιναι:



Αμέσως μετά την την συνάρτηση perm ο μετασχηματισμένος πίνακας που σχηματίζεται είναι:

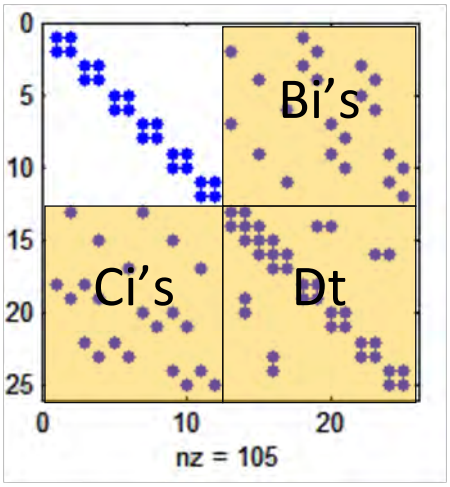


Στον οποίο όπως πράγματι παρατηρείται σχηματίζονται 4 sub-domains οι πάνω αριστερά υποπίνακες είναι οι A_1, A_2, A_3, A_4 ο κατώ δεξιά είναι ο D_t οι κάτω αριστερά είναι οι υποπίνακες C_1, C_2, C_3, C_4 και οι πάνω δεξιά είναι B_1, B_2, B_3, B_4 βάση θεωρίας.Σε αυτούς τους υποπίνακες ο αριθμός των μη μηδενικών στοιχείων φαίνεται παρακάτω:

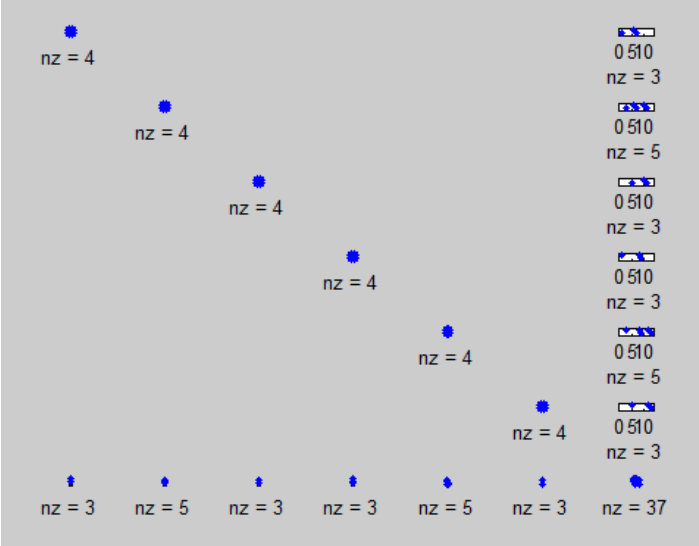


Εν συνεχεία καλείται η σύνάρτηση domain_decomposition η οποία κάνει επίλυση αυτών των υποσυστημάτων

Για tearing του ίδιου δικτύου σε 3×2 subdomains ο αναδιατεταγμένος πίνακας που σχηματίζεται μετά την διαδικασία perm παρουσιάζεται παρακάτω:



Ο αριθμός των μη μηδενικών στοιχείων στον κάθε υποπίνακα παρουσιάζεται στο παρακάτω σχήμα:



Παρατηρούμε ότι όσο μεγαλύτερος είναι ο αριθμός των subdomains τόσο μικρότερος είναι ο αριθμός των στοιχείων των υποπινάκων A_i και κατ'επέκταση τόσο μικρότερος ο αριθμός των μη μηδενικών στοιχείων του. Αντίστοιχα με τον αριθμό των subdomains αυξάνεται και το ποσοστό των εξισώσεων που ανήκουν στο tearing set - δηλαδή τα C_i καθώς και το μέγεθος του D_t. Συνεπώς υπάρχει ένα tradeoff και θα πρέπει να μη σπεύδουμε να αυξήσουμε τον αριθμό των subdomains αν το μέγεθος του πλέγματος δεν το δικαιολογεί γιατί καταλήγουμε να λύνουμε μία αρκετά σύνθετη συνάρτηση του tearing set και trivial υποσυστήματα A_i έχοντας στο τέλος λιγότερο καλή υπολογιστική απόδοση.

Για να τρέξουμε την εξομοίωση σε Matlab αρκεί να γράψουμε την εντολή test. Στο τέλος θα δούμε κάτι σαν το εξής:

```
>> test
ans =
  5.3926e-013
>>
```

Το νούμερο που μας δίνεται είναι η νόρμα (σφάλμα) μεταξύ της λύσης με αναστροφή του A και της λύσης με το Domain Decomposition (Node tearing). Όπως βλέπουμε οι παραπάνω τιμές είναι σχεδόν ίδιες το οποίο μας επαληθεύει ότι η λύση του συστήματός μας γίνεται με σωστό τρόπο.

Για να λύσουμε το ίδιο πρόβλημα σε C πρέπει πρώτα να κάνουμε compile τη βιβλιοθήκη CSparse. Το πρόγραμμα δοκιμάστηκε και μπορεί να κάνει compile σε Linux 64-bit, Linux 32 bit και Windows/Cygwin. Για να το κάνουμε αυτό πηγαίνουμε στον κατάλογο CSparse και κάνουμε:

```
cd CSparse
make clean
make
```

Μετά από λίγο θα έχει ολοκληρωθεί το compile της βιβλιοθήκης και μπορούμε να γυρίσουμε πίσω στον αρχικό κατάλογο του συστήματος. Εκεί μπορούμε να κάνουμε compile το πρόγραμμά μας με make clean και make.

```
cd ..
make clean
make
```

Αμέσως μετά μπορούμε να τρέξουμε το πρόγραμμα εξομοίωσης:

```
$ ./sim
rx_=[635.887857;494.390786;431.431751;376.499243;327.881494;550.089686;426.2
04791;413.473128;426.805005;517.722671;409.897187;504.254267;419.735711;426.
517405;476.625620;515.374029;463.754771;449.827768;536.978094;475.310384;485
.474320;613.966224;583.110413;411.744352;309.101305];
```

Βλέπουμε το αποτέλεσμα. Αυτό (rx_) μπορούμε να το κάνουμε copy-paste πίσω στο Matlab. Από εκεί μπορούμε να υπολογίσουμε το norm(rx-rx_) το οποίο μας δείχνει την διαφορά μεταξύ των δύο διανυσμάτων. Αυτό είναι:

```
>> norm(rx-rx_)
ans =
    0.3183
>>
```

Φυσικά ένα σφάλμα της τάξης του 0.5 κάθε άλλο από μικρό είναι αλλά δυστυχώς αυτό είναι το τίμημα που πληρώνουμε για την περιορισμένη ακρίβεια της επίλυσης συστημάτων με την μέθοδο LU από την βιβλιοθήκη CSparse. Παρόλα αυτά αν κοιτάξει κανείς λίγο τις τιμές των διανυσμάτων καλύτερα μπορεί να παρατηρήσει το εξής:

```
>> rx

rx =
```

```
  635.8281
  494.3299
  431.3746
  376.4466
  ...
  583.0491
  411.6845
  309.0476
>> rx-rx_

ans =
   -0.0598
   -0.0609
   -0.0572
   -0.0526
   ...
   -0.0613
   -0.0599
   -0.0537
>> 100 * (norm(rx-rx_)/mean(rx))
ans =
    0.0681
>>
```

Παρατηρούμε λοιπόν ότι ακόμη κι αν η νόρμα είναι σχετικά μεγάλη, το σφάλμα είναι της τάξης του 0.07% για το μέγεθος των νούμερων τα οποία έχει αυτό το πρόβλημα, το οποίο είναι αρκετά καλό. Παρόλα αυτά θα ήταν επιθυμητό να αυξήσουμε την ακρίβεια των αποτελεσμάτων που δίνει η υλοποίηση σε C χρησιμοποιώντας τη μέθοδο LU ή κάποια άλλη μέθοδο για την επίλυση συστημάτων και αναστροφή πινάκων με μεγαλύτερη ακρίβεια.

7.Πειραματικά αποτελέσματα

Τα χαρακτηριστικά του μηχανήματος πάνω στο οποίο έγιναν οι δοκιμές είναι:

CPU: INTEL CORE I5 2.30GHz

RAM : 4GB

Στους παρακάτω πίνακες μπορούμε να παρατηρήσουμε τα συνοπτικά αποτελέσματα των χρόνων εκτέλεσης της μεθόδου domain decomposition με την βοήθεια του matlab γία κάθε ξεχωριστό δίκτυο.

Επίλυση	n1=n2=5
Domain Decomposition 2×2	
`xi{i} = d.Ain{i} * (bi{i}- d.Bi{i}*xt);`	Χρόνος prepare : 8.2015e-004 Επίλυση : 2.089e-004 Σφάλμα : 2.3537e-015
`xi{i}=d.Ui{i} \ ((d.Lin{i} * bi{i}) - (d.LinBi{i} * xt));`	Χρόνος prepare : 8.6034e-004 Επίλυση : 2.1267e-004 Σφάλμα : 2.5162e-015

Επίλυση	n1=n2=17
Domain Decomposition 2×2	
`xi{i} = d.Ain{i} * (bi{i}- d.Bi{i}*xt);`	Χρόνος prepare : 3.9e-3 Επίλυση : 3.4902e-004 Σφάλμα : 1.0011e-015
`xi{i}=d.Ui{i} \ ((d.Lin{i} * bi{i}) - (d.LinBi{i} * xt));`	Χρόνος prepare : 3.9e-3 Επίλυση : 3.9361e-004 Σφάλμα : 3.0231e-015

Επίλυση	n1=n2=25
Domain Decomposition 2×2	
`xi{i} = d.Ain{i} * (bi{i}- d.Bi{i}*xt);`	Χρόνος prepare:1.58e-2 Επίλυση:8.4737e-004 Σφάλμα:2.8752e-015
`xi{i}=d.Ui{i} \ ((d.Lin{i} * bi{i}) - (d.LinBi{i} * xt));`	Χρόνος prepare:1.51e-2 Επίλυση:9.1095e-004 Σφάλμα:2.2533e-015

Επίλυση	n1=n2=51
Domain Decomposition 2×2	
`xi{i} = d.Ain{i} * (bi{i}- d.Bi{i}*xt);`	Χρόνος prepare:3.513e-1 Επίλυση:1.32e-2 Σφάλμα:9.1247e-016
`xi{i}=d.Ui{i} \ ((d.Lin{i} * bi{i}) - (d.LinBi{i} * xt));`	Χρόνος prepare:3.506e-1 Επίλυση:1.23e-2 Σφάλμα:8.6753e-015

Προσομοιώνοντας τα τέσσερα αυτά κυκλώματα παρατήρουμε ότι η ποίο βέλτιστη λύση (ταχύτητα εκτέλεσης,σφάλμα λύσεων)είναι η δεύτερη.

Έν συνεχεία μέτρησα τους εξής χρόνους βάζοντας τον ανάλογο κώδικα στο πρόγραμμά μου σε C:

α) t_prepare: Χρόνος προετοιμασίας των πινάκων (εκτελείται μία φορά για το πρόβλημα)

β) t_subdomain: Χρόνος επίλυσης ενός μόνο subdomain (υποδικτύου)

γ) t_tearing: Χρόνος επίλυσης του tearing set

δ) t_solve_total: Συνολικός χρόνος επίλυσης ενός κύκλου.

Για να ανακτήσω κάποια ενδιαφέροντα στατιστικά στοιχεία έτρεξα το πρόγραμμά μου για δίκτυα διαφόρων μεγεθών από 11x11=121 έως 83x83=6889 κόμβους. Επίσης έτρεξα για διαφορετικούς αριθμούς υποδικτύων, 2x2 = 4 subdomains, 3x3 = 9 subdomains και 4x4 = 16 subdomains. Βρήκα κατάλληλα δίκτυα μεγέθους έστω n τα οποία να «βγαίνουν» για όλες αυτές τις διαφορετικές κατατμήσεις έστω k απαιτώντας το μέγεθος του τμήματος MT = (n - k + 1)/k να είναι ακέραιο για k=2,3 και 4. Ο αντίστοιχος πίνακας έχει ως εξής:

n	d2	d3	d4
1	0	-0.33333	-0.5
2	0.5	0	-0.25
3	1	0.333333	0
		…	
10	4.5	2.666667	1.75
11	**5**	**3**	**2**
12	5.5	3.333333	2.25
		…	
22	10.5	6.666667	4.75
23	**11**	**7**	**5**
		…	
34	16.5	10.66667	7.75
35	**17**	**11**	**8**
		…	
46	22.5	14.66667	10.75
47	**23**	**15**	**11**
		…	
70	34.5	22.66667	16.75
71	**35**	**23**	**17**
		…	
82	40.5	26.66667	19.75
83	**41**	**27**	**20**

Συνεπώς βλέπουμε ότι τα επιλεγμένα n είναι: 11 23 35 47 59 71 83. Δημιουργήσαμε τους αντίστοιχους πίνακες με το matlab, τους βάλαμε στον κατάλογο test_data κι επίσης γράψαμε ένα πρόγραμμα run.sh που αυτοματοποιεί σχετικά την διαδικασία εκτέλεσης μίας μέτρησης. Αυτό έχει ως εξής:

```
#!/bin/sh
rm -rf __run
mkdir __run
cd __run
cp ../test_data/A$1.csv ./A.csv
cp ../test_data/r$1.csv ./r.csv
../sim
cd ..
rm -rf __run
```

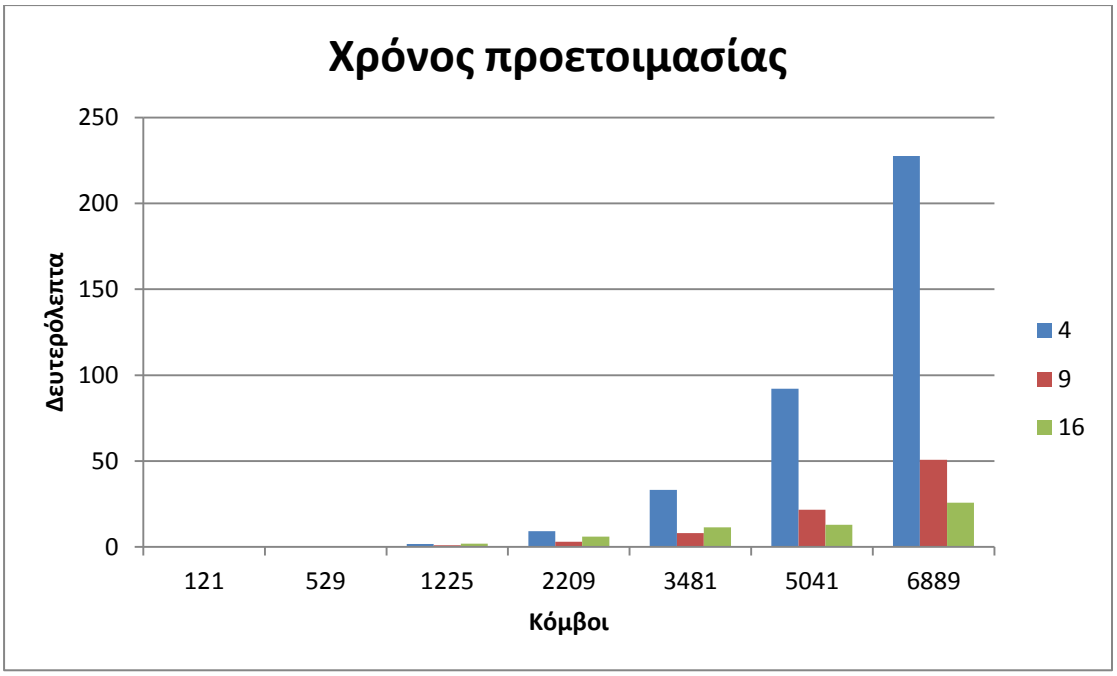
Με τη βοήθειά του μπορούμε να τρέξουμε για όλους τους πίνακες γράφοντας σε ένα linux prompt απλά:

```
for i in 11 23 35 47 59 71 83; do
    ./run.sh $i | grep '##'
done
```

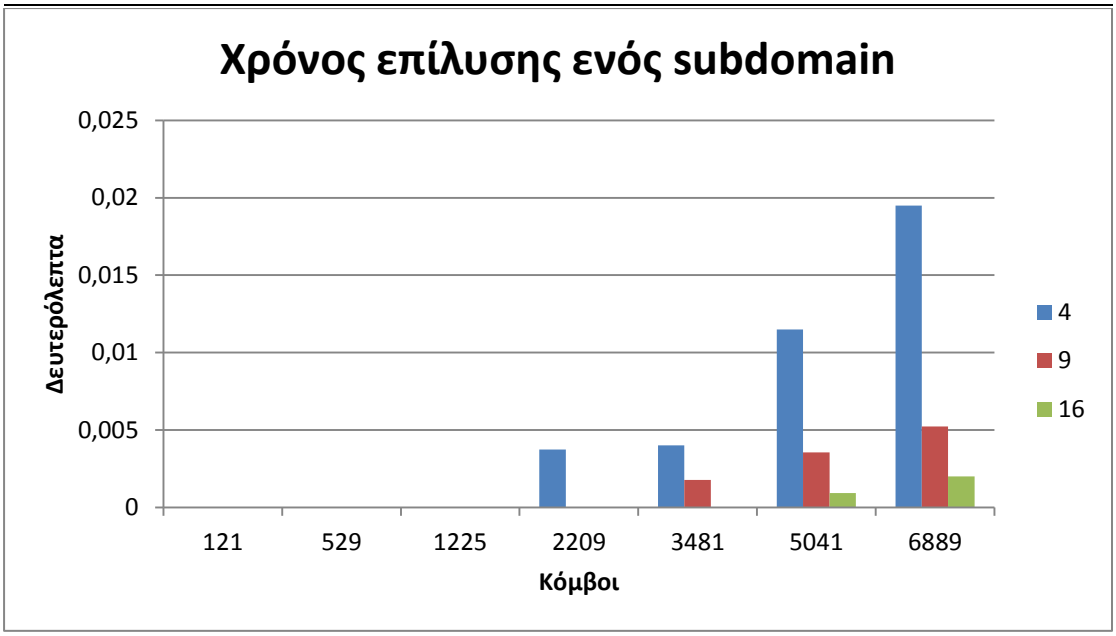
Τα αποτελέσματα των μετρήσεων είναι τα εξής:

n	domains	nodes	tearsz	t_prepare	t_subdomain	t_tearing	t_solve_total
11	4	121	21	0	0	0	0
11	9	121	40	0.016	0	0	0
11	16	121	57	0.015	0	0	0
23	4	529	45	0.219	0	0	0
23	9	529	88	0.156	0	0	0
23	16	529	129	0.375	0	0	0
35	4	1225	69	1.841	0	0.015	0.015
35	9	1225	136	0.983	0	0	0
35	16	1225	201	1.966	0	0	0
47	4	2209	93	9.298	0.00375	0	0.015
47	9	2209	184	2.995	0	0.016	0.016
47	16	2209	273	6.085	0	0	0
59	4	3481	117	33.165001	0.004	0.015	0.031
59	9	3481	232	8.05	0.001778	0	0.016
59	16	3481	345	11.42	0	0.015	0.015
71	4	5041	141	92.087997	0.0115	0	0.046
71	9	5041	280	21.698999	0.003556	0.016	0.048
71	16	5041	417	13.011	0.000937	0.016	0.031
83	4	6889	165	227.496002	0.0195	0	0.078
83	9	6889	328	50.683998	0.005222	0.016	0.063
83	16	6889	489	25.787001	0.002	0.015	0.047

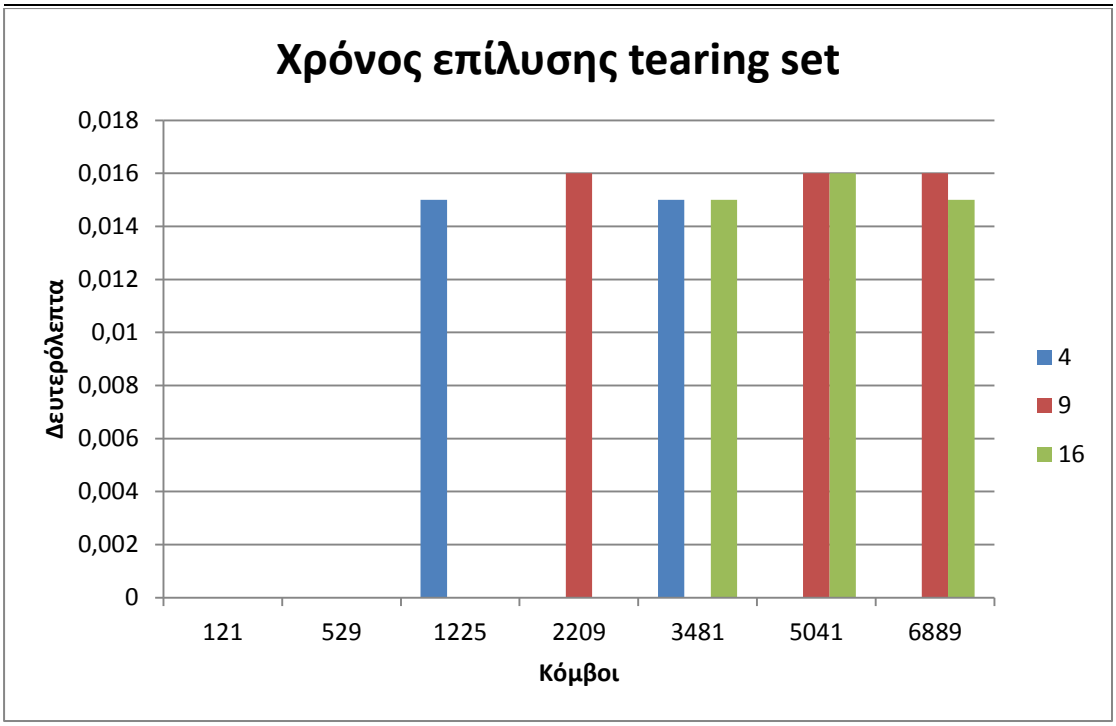
Γραφικά μπορούμε να τα συγκεντρώσουμε στους ακόλουθους πίνακες/διαγράμματα:



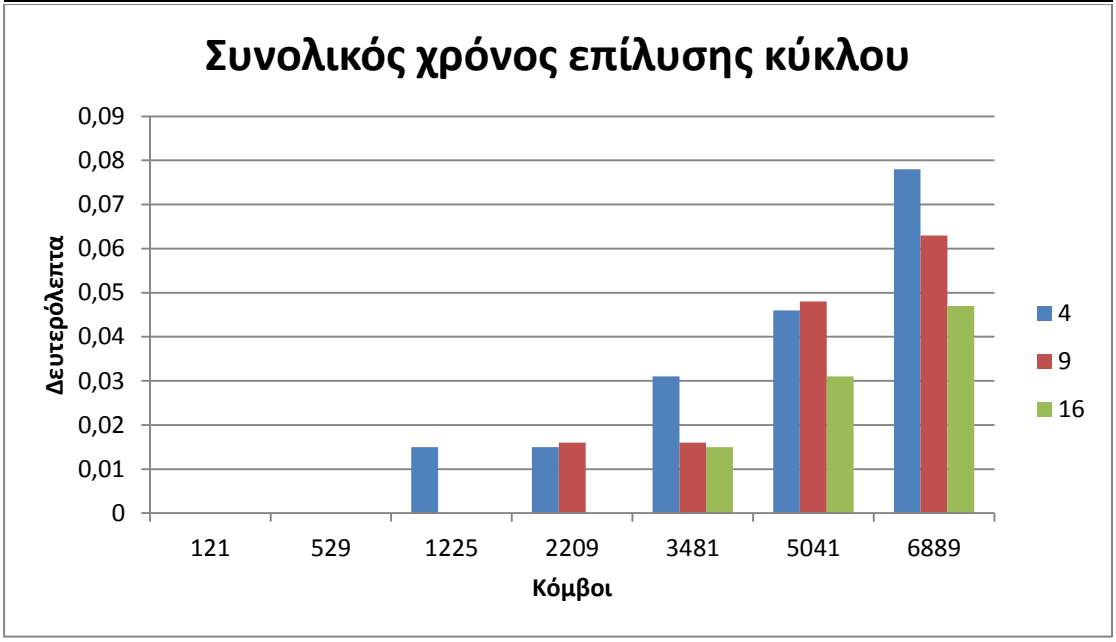
Sum of t_prepare	Column Labels		
Row Labels	**4**	**9**	**16**
121	0	0.016	0.015
529	0.219	0.156	0.375
1225	1.841	0.983	1.966
2209	9.298	2.995	6.085
3481	33.165001	8.05	11.42
5041	92.087997	21.698999	13.011
6889	227.496002	50.683998	25.787001



Sum of t_subdomain	Column Labels		
Row Labels	**4**	**9**	**16**
121	0	0	0
529	0	0	0
1225	0	0	0
2209	0.00375	0	0
3481	0.004	0.001778	0
5041	0.0115	0.003556	0.000937
6889	0.0195	0.005222	0.002



Sum of t_tearing	Column Labels		
Row Labels	4	9	16
121	0	0	0
529	0	0	0
1225	0.015	0	0
2209	0	0.016	0
3481	0.015	0	0.015
5041	0	0.016	0.016
6889	0	0.016	0.015

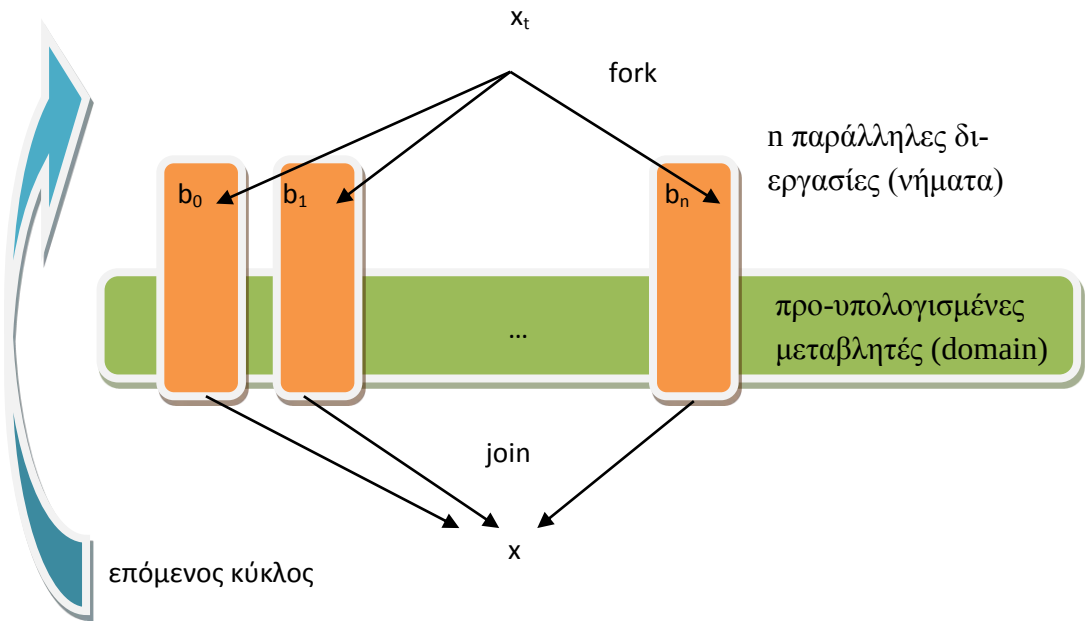


Sum of t_solve_total	Column Labels		
Row Labels	**4**	**9**	**16**
121	0	0	0
529	0	0	0
1225	0.015	0	0
2209	0.015	0.016	0
3481	0.031	0.016	0.015
5041	0.046	0.048	0.031
6889	0.078	0.063	0.047

Παρατηρούμε ότι όπως ήταν αναμενόμενο όλοι οι χρόνοι υπολογισμού αυξάνουν πιο γρήγορα από $O(n^2)$ αλλά επίσης παρατηρούμε ότι όλοι οι χρόνοι μειώνονται σχεδόν ανάλογα μα τον αριθμό των subdomains που έχουμε. Επίσης παρατηρούμε ότι ο χρόνος επίλυσης ενός subdomain κυριαρχεί τον χρόνο επεξεργασίας κάθε κύκλου. Αυτό είναι πολύ καλό γιατί ακριβώς αυτό το κομμάτι είναι που καταφέρνουμε να παραλληλήσουμε με αυτή τη μέθοδο. Συνεπώς αντί να χρειάζεται π.χ. 16x αυτόν τον χρόνο για 4x4 subdomains, θα χρειάζεται κάτι ελαφρώς μεγαλύτερο από 1x αυτού του χρόνου σε έναν 16-πύρηνο επεξεργαστή. Είναι προφανές ότι έχουμε σημαντικά οφέλη.

8.Συμπεράσματα

Σκοπός μας με το domain decomposition είναι να μπορέσουμε να λύσουμε το πρόβλημα παράλληλα. Παρόλα αυτά δεν είναι όλοι οι τρόποι παράλληλης επίλυσης ικανοί να βοηθήσουν σε αυτό το πρόβλημα. Συγκεκριμένα υποθέτωντας ένα σύστημα εξομοίωσης στο οποίο πρέπει να έχουμε λύσει το σύστημα για t πριν προσπαθίσουμε να το λύσουμε για t=t+1 θα πρέπει να ολοκληρώνεται πλήρως η κλήση της d_solve πριν να ξεκινήσουμε την κλήση της επόμενης d_solve. Το θετικό με το πρόβλημά μας είναι ότι ένα τεράστιο μέρος των πινάκων είναι σταθερό και μπορεί να προ-υπολογιστεί και να βρίσκεται προαποθηκευμένο σε διαφορετικούς υπολογιστές που τρέχουν παράλληλα. Το αρνητικό είναι ότι δεν έχουμε την ίδια ευκαίρεια με το x_t. Η επίλυση του x_t θα πρέπει να γίνει σε έναν επεξεργαστή νωρίτερα, για να μπορέσει να σταλεί σε όλους τους άλλους (μαζί με τα αντίστοιχα b_i's) ώστε να μπορέσουμε να υ-πολογιστούν τα x_i's. Στο τέλος αυτής της διαδικασίας τα αποτελέσματα πρέπει και πάλι να συγκεντρωθούν σε ένα τελικό διάνυσμα λύσης x. Αυτό σημαίνει ότι έχουμε αρκετή μεταφορά δεδομένων μεταξύ των παράλληλων επεξεργαστών σε κάθε κύκλο εξομοίωσης και πρέπει να περιμένουμε να ολοκληρωθούν όλοι οι υπολογισμοί πριν να προχωρήσουμε στο επόμενο βήμα.



Αυτό είναι το μοντέλο fork() – join() το οποίο είναι πολύ δημοφιλές αλλά δεν μας δίνει τη δυνατότητα για πραγματικά παράλληλο (ασύγχρονο) μοντέλο υπολογισμού γιατί υπάρχει η εξάρτηση του κάθε κύκλου με τον επόμενο. Συνεπώς η διάρκεια υπολογισμού ενός κύκλου είναι ίση με την πιο αργή διάρκεια υπολογισμού των A_i. Πιο συγκεκριμένα:

$$t_{\text{κύκλου}} = t_{\text{υπολογισμού xt}} + t_{\text{ μεταφοράς xt,bi}} + \text{Max}(t_{\text{επίλυσης xi}} + t_{\text{μεταφοράς xi}}).$$

Όπου $t_{\text{υπολογισμού xt}}$ είναι ο χρόνος επίλυσης του συστήματος x_t, $t_{\text{ μεταφοράς xt,bi}}$ είναι ο χρόνος που χρειάζεται για να μεταφέρουμε τα bytes των δεδομένων των x_t και b_i στους επιμέρους επεξεργαστές (το b_i μπορεί πιθανώς να μεταφέρεται παράλληλα με τον υπολογισμό του x_t οπότε πιθανώς να μπορούμε να γλυτώσουμε κάτι εκεί) και μετά λόγω της παραλληλίας ο χρόνος υπολογισμού των επιμέρους συστημάτων είναι ίσος με τον μέγιστο (Max) των $t_{\text{επίλυσης xi}}$ του χρόνου δηλαδή επίλυσης του επιμέρους συστήματος και του $t_{\text{μεταφοράς xi}}$ του χρόνου δηλαδή μεταφοράς των bytes της λύσης x_i πίσω στον αρχικό επεξεργαστή.

Το σημαντικό είναι ότι με τις απίστευτα γρήγορες ταχύτητες επεξεργασίας των σημερινών υπολογιστών το μεγαλύτερο από αυτές τις καθυστερήσεις μάλλον δεν θα είναι ο χρόνος επίλυσης αλλά ο χρόνος μεταφοράς και πιθανότατα αν έχουμε 10άδες χιλιάδες κόμβους (οπότε και αξίζει να κάνουμε node tearing) όχι του x_t αλλά ο χρόνος μεταφοράς $t_{\text{μεταφοράς xi}}$ γιατί το x_i μπορεί να έχει μεν μέγεθος μερικών kBytes αλλά το χειρότερο είναι ότι τα περισσότερα κανάλια μεταφοράς πληροφορίας μεταξύ επεξεργαστών έχουν μεγάλη καθυστέρηση (latency).

Η καλύτερη λοιπόν δυνατή πλατφόρμα εκτέλεσης αυτού του συστήματος είναι με πολλαπλά νήματα όπου όλα τα x_i, b_i βρίσκονται στην κεντρική μνήμη και οι προυπολογισμένες μεταβλητές των subdomains βρίσκονται στις επιμέρους λανθάνουσες μνήμες (cache) των επεξεργαστών. Το καλύτερο είναι το σύστημα να έχει τόσους επεξεργαστές όσα και τα subdomains. Το να έχουμε περισσότερα subdomains από επεξεργαστές στο σύστημα δεν μας δίνει βελτιωμένη απόδοση του συστήματος.

Επίσης το δίκτυο τροφοδοσίας είναι καλό να έχει 10-αδες χιλιάδες κόμβους έτσι ώστε ο υπολογισμός του x_t δηλαδή η επίλυση του συστήματος του tearing set σε κάθε κύκλο να είναι ασήμαντη υπολογιστικά σε σύγκριση με την επίλυση των αραιών συστημάτων που δίνουν τα x_i. Με αυτόν τον τρόπο θα μπορούμε να εκμεταλλευόμαστε

την ανεξάρτητη επεξεργαστική ισχύ του κάθε επεξεργαστή. Όσο μεγαλώνει ο αριθμός των κόμβων, τόσο μεγαλύτερο είναι το μέγεθος των b_i το οποίο σημαίνει αρκετά υψηλό κόστος επικοινωνίας μεταξύ της μονάδας που υπολογίζει τα x_t και των μονάδων που κάνουν τους επιμέρους υπολογισμούς των x_i. Γι'αυτό, κι επειδή ο χρόνος υπολογισμού είναι ίσος με τον χειρότερο χρόνο υπολογισμού των x_i's, το καλύτερο είναι να ελαχιστοποιήσουμε τον χρόνο μεταφοράς των δεδομένων έχοντας ένα σύστημα με πολλούς επεξεργαστές πάνω στην ίδια Motherboard ή μέσα στο ίδιο Chip και ΟΧΙ κάποιο σύστημα (cluster) με πολλούς επεξεργαστές που επικοινωνούν μέσω δικτύου (π.χ. Ethernet). Αν χρησιμοποιήσουμε Ethernet το πιθανότερο είναι ότι θα έχουμε τους επεξεργαστές να περιμένουν την περισσότερη ώρα να αποσταλούν τα δεδομένα των x_i's πίσω στον κεντρικό επεξεργαστή.

Είδαμε λοιπόν πώς μπορούμε να χρησιμοποιήσουμε παράλληλη επεξεργασία χρησιμοποιώντας αραιούς πίνακες και τη μέθοδο node tearing για να επιλύουμε πολύ πιο αποτελεσματικά και γρήγορα πολύπλοκα συστήματα δικτύων τροφοδοσίας. Παρατέθηκαν όλες οι σχετικές μέθοδοι και υλοποίηση τόσο σε Matlab όσο και σε C με τη βοήθεια της βιβλιοθήκης CSparse. Τέλος εξετάσαμε το καλύτερο μοντέλο επεξεργασίας με βάση τον αλγόριθμό μας ώστε να επλύουμε όσο το δυνατόν ταχύτερα κάθε κύκλο εξομοίωσης. Ήταν ένα αρκετά απαιτητικό και πολυδιάστατο πρόβλημα το οποίο αντιμετωπίσαμε με επιτυχία.

9.Παράρτημα Α(Κώδικας του απλού εξομοιωτή)

Δημιουργία της λίστας η οποία διαβάζει από το αρχείο την τοπολογία του δικτύου

```
#include <stdio.h>
#include <stdlib.h>

/* ορισμός της δομής του στοιχείου της λίστας ή του κυκλώματος */
struct Circuit_Element
{
	char elementSymbol[3];
	char elementName[5];
	int node1;
	int node2;
	int node3;
	double value;
	char group;
	struct Circuit_Element *next;
};


/* κάνω typedef τη δομή struct Circuit_Element */
typedef struct Circuit_Element CircuitElement;

CircuitElement list;


/* υπολογίζει αριθμό κόμβων */
int getRElements()
{
	CircuitElement *temp;
	temp=&list;
	int counter=0;
	while(temp->next!=NULL)
	{
		temp=temp->next;
		if(temp->elementSymbol[0]=='R') counter++;
	}
	return counter;
}


int getGroupRElements()
{
	CircuitElement *temp;
	temp=&list;
	int counter=0;
	while(temp->next!=NULL)
	{
		temp=temp->next;
		if(temp->elementSymbol[0]=='R' && temp->group=='G')
counter++;
	}
```

```
	return counter;
}



/* υπολογίζει ανεξάρτητες πηγές τάσης */
int getVElements()
{
	CircuitElement *temp;
	temp=&list;
	int counter=0;
	while(temp->next!=NULL)
	{
		temp=temp->next;
		if(temp->elementSymbol[0]=='V') counter++;
	}
	return counter;
}


/* υπολογίζει στοιχεία κυκλώματος */
int getAllElements()
{
	CircuitElement *temp;
	temp=&list;
	int counter=0;
	while(temp->next!=NULL)
	{
		counter++;
		temp=temp->next;
	}
	return counter;
}

/* εκτύπωση λίστας */
void printList()
{
	CircuitElement *temp;
	temp=&list;
	int counter=0;
	while(temp->next!=NULL)
	{
		temp=temp->next;
		printf("%s\t", temp->elementName);
		printf("%d\t", temp->node1);
				printf("%d\t", temp->node2);
		if(temp->node3>=0) printf("%d\t", temp->node3);
		if(temp->value>=0) printf("%0.3f\t", temp->value);
		printf("%c\t", temp->group);
		printf("\n");
		counter++;
	}
```

```
	printf("\n\nlist contains %d lines.\n", counter);
}
```

Εν συνεχεία κατασευή του πίνακα MNA όπως περιγράφηκε στο κεφάλαιο 2.2.1

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "timer.h"


int ALL_ELEMENTS; /*αποθηκεύει αριθμό στοιχείων */
int R_ELEMENTS; /*αποθηκεύει αριθμό αντιστάσεων */
int R_GROUP_ELEMENTS; /*αποθηκεύει αριθμό αντιστάσεων group 2 */
int V_ELEMENTS; /*αποθηκεύει αριθμό πηγών */

/* Εκτυπώνει τιμές πίνακα για πίνακες σαν δεκαδικές τιμές με 3 δεκα-
δικά ψηφία */
void printMatrix(double *pinax, int r, int c)
{
	int i,j;
	printf("\n\n----------------------------------\n\n	");
	for(i=0; i<r; i++)
	{
		printf("\n");
		for(j=0; j<c; j++)
			printf("%+5.3f\t", *pinax++);
	}
	printf("\n\n");
}


/* υπολογισμός του πίνακα A */
void setMatrixA(double AM[ALL_ELEMENTS][ALL_ELEMENTS])
{
	int i,j;
	int counter;
	int node_1,node_2;
	/* αρχικοποίηση πίνακα */
	for(i=0; i<ALL_ELEMENTS; i++){
		for(j=0; j<ALL_ELEMENTS; j++)
			AM[i][j]=0;
	  }

/* επιλέγω πρώτα τις αντιστάσεις από group 1*/
	CircuitElement *temp;
	temp=&list;
	while(temp->next!=NULL)
	{
		temp=temp->next;
		if(temp->elementSymbol[0]=='R' && temp->group=='\0')
		{
			node_1=temp->node1;
			node_2=temp->node2;
				if(node_1>0)
					AM[node_1-1][node_1-1] += 1/temp-
>value;
				if(node_2>0)
					AM[node_2-1][node_2-1] += 1/temp-
>value;
				if(node_1>0 && node_2>0)
				{
```

```
                              AM[node_1-1][node_2-1] -= 1/temp-
>value;
                              AM[node_2-1][node_1-1] -= 1/temp-
>value;
                        }
            }
      }
/* επιλέγω μετά τις αντιστάσεις από group 2 */
      temp=&list;
      counter=R_ELEMENTS;
      while(temp->next!=NULL)
      {
            temp=temp->next;
            if(temp->elementSymbol[0]=='R' && temp->group=='G')
            {
                  node_1=temp->node1;
                  node_2=temp->node2;
                  AM[counter][counter] -= temp->value;
                  if(node_1>0)
                  {
                        AM[counter][node_1-1] += 1;
                        AM[node_1-1][counter] += 1;
                  }
                  if(node_2>0)
                  {
                        AM[counter][node_2-1] -= 1;
                        AM[node_2-1][counter] -= 1;
                  }
                  counter++;
            }
      }

/* επιλέγω μετά τις πηγές */
      temp=&list;
      counter=ALL_ELEMENTS-V_ELEMENTS;
      while(temp->next!=NULL)
      {
            temp=temp->next;
            if(temp->elementSymbol[0]=='V')
            {
                  node_1=temp->node1;
                  node_2=temp->node2;
                  if(node_1>0)
                  {
                        AM[counter][node_1-1] += 1;
```

```
                        AM[node_1-1][counter] += 1;
                }
                if(node_2>0)
                {
                        AM[counter][node_2-1] -= 1;
                        AM[node_2-1][counter] -= 1;
                }
                counter++;
        }
    }
}


/* υπολογισμός του πίνακα X */
void setMatrixX(double XM[ALL_ELEMENTS], double ZM[ALL_ELEMENTS])
{
    /* αντιγράφω τον πίνακα στο X από τον Z*/
    int i;
    for(i=0; i<ALL_ELEMENTS; i++) XM[i]=ZM[i];

}
/* υπολογισμός του πίνακα Z */
void setMatrixZ(double ZM[ALL_ELEMENTS])
{
    int i;
    int node_1,node_2;
    int counter;

    /* αρχικοποιώ τον πίνακα */
    for(i=0; i<ALL_ELEMENTS; i++) ZM[i]=0;

/* επιλέγω τις πηγές ρεύματος */
    CircuitElement *temp;
    temp=&list;
    while(temp->next!=NULL)
    {
        temp=temp->next;
        if(temp->elementSymbol[0]=='I')
        {
            node_1=temp->node1;
            node_2=temp->node2;
            if(node_1>0)
            {
```

```
                        ZM[node_1-1] -= temp->value;
                }
                if(node_2>0)
                {
                        ZM[node_2-1] += temp->value;
                }
        }
    }


/* επιλέγω τις πηγές τάσης */
    temp=&list;
    counter=ALL_ELEMENTS-V_ELEMENTS;
    while(temp->next!=NULL)
    {
        temp=temp->next;
        if(temp->elementSymbol[0]=='V')
        {
            ZM[counter] += temp->value;
        }
        counter++;
    }
}




void getMNA()
{
    ALL_ELEMENTS=getAllElements(); /* βρίσκουμε πρώτα αριθμό στοι-
χείων */
    R_ELEMENTS=getRElements(); /*  */
    R_GROUP_ELEMENTS=getGroupRElements(); /*  */
    V_ELEMENTS=getVElements(); /*  */


    double A[ALL_ELEMENTS][ALL_ELEMENTS];  /* Δημιουργία πίνακα A
*/
    double X[ALL_ELEMENTS];  /* Δημιουργία πίνακα X, αποθηκεύονται
οι άγνωστες τιμές*/
    double Z[ALL_ELEMENTS];  /* Δημιουργία πίνακα Z */


    setMatrixA(A);   /* υπολογίζουμε τιμές πίνακα για A */
    setMatrixZ(Z);   /* υπολογίζουμε τιμές πίνακα για Z */
    setMatrixX(X, Z);   /* αντιγράφω τις τιμές του Z στο X */
```

```
	/*
	εκτυπώνω πίνακες αν χρειάζεται να ελέγξω κάτι
	printMatrix(A, ALL_ELEMENTS, ALL_ELEMENTS);
	printMatrix(Z, ALL_ELEMENTS, 1);
	*/

	/* εκτυπώνω τον πίνακα Α πριν καλέσω την gauss */
	printMatrix(A, ALL_ELEMENTS, ALL_ELEMENTS);

	/* εκτυπώνω τη στήλη Z */
	printMatrix(Z, ALL_ELEMENTS, 1);

	/* καλώ τη gauss για τον υπολογισμό της X οι τελικές τιμές θα αποθηκευθούν στο X*/
	double begin = BeginTimer(); // εκκίνηση ρολογιού
	gauss(A, ALL_ELEMENTS, X);
	float elapTicks = EndTimer(begin); // σταμάτημα ρολογιού και υπολογισμός του χρόνου που πέρασε
    float elapMilli, elapSeconds, elapMinutes;
    elapMilli = elapTicks/1000;     // milliseconds
    elapSeconds = elapMilli/1000;   // seconds
    elapMinutes = elapSeconds/60;   // minutes



	/* εκτυπώνω τον πίνακα Α αφού καλέσω την gauss, γίνεται τριγωνικός*/
	printMatrix(A, ALL_ELEMENTS, ALL_ELEMENTS);


	/* εκτυπώνω τη στήλη X με τις τελικές τιμές */
	printMatrix(X, ALL_ELEMENTS, 1);

	/* εκτύπωση του χρόνου που πέρασε για τον υπολογισμό της gauss */
	printf("Gauss time in milliseconds: %lf\n",elapMilli);

}
```

Επιπλέον επίλυση του συστήματος με την μέθοδο του Gauss

```
#include <math.h>  /* required for fabs() */

int gauss(double *A, int n, double *B)
{
```

```
      int row, i, j, pivot_row;
      double max, dum, *pa, *pA, *A_pivot_row;

      pa = A;
      for (row = 0; row < (n - 1); row++, pa += n)
      {
            A_pivot_row = pa;
            max = fabs(*(pa + row));
            pA = pa + n;
            pivot_row = row;
            for (i = row + 1; i < n; pA += n, i++)
                  if ((dum = fabs(*(pA + row))) > max) {
                        max = dum; A_pivot_row = pA; pivot_row = i;
                  }
            if (max == 0.0) return -1; /* the matrix A is singular
*/

/* and if it differs from the current row, interchange the two rows.
*/

            if (pivot_row != row) {
                  for (i = row; i < n; i++) {
                        dum = *(pa + i);
                        *(pa + i) = *(A_pivot_row + i);
                        *(A_pivot_row + i) = dum;
                  }
                  dum = B[row];
                  B[row] = B[pivot_row];
                  B[pivot_row] = dum;
            }

/* Perform forward substitution */

            for (i = row + 1; i < n; i++)
            {
                  pA = A + i * n;
                  dum = - *(pA + row) / *(pa + row);
                  *(pA + row) = 0.0;
                  for (j = row + 1; j < n; j++) *(pA + j) += dum *
*(pa + j);
                  B[i] += dum * B[row];
```

```
            }
      }



/* Perform backward substitution */


      pa = A + (n - 1) * n;
      for (row = n - 1; row >= 0; pa -= n, row--) {
            if ( *(pa + row) == 0.0 ) return -1;  /* matrix is sin-
gular */
            dum = 1.0 / *(pa + row);
            for ( i = row + 1; i < n; i++) *(pa + i) *= dum;
            B[row] *= dum;
            for ( i = 0, pA = A; i < row; pA += n, i++) {
                  dum = *(pA + row);
                  for ( j = row + 1; j < n; j++) *(pA + j) -= dum *
*(pa + j);
                  B[i] -= dum * B[row];
            }
      }
      return 0;
}
```

Τέλος παρουσιάζουμε την συνάρτηση main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "list.h"
#include "mna.h"
#include "gauss.h"

#define LINE_LENGTH 256


/* συνάρτηση που ελέγχει τη γραμμή που διαβάστηκε από το αρχείο (κα-
λείται από την main()) */
char checkLine(char ch[])
{   char * eoln = strstr(ch,"\n");
    if (eoln) *eoln=0;
    eoln = strstr(ch,"\r");
    if (eoln) *eoln=0;

//    if(strlen(ch)==0) return 0;

      if(strlen(ch)==0) return 0;
```

```
	int length;
	int good=0;
	int i,j;
	char c;
	char element[6][10];  /* πίνακας συμβολοσειρών για την αποθή-
κευση των 4 μέχρι 6 στοιχείων για κάθε γραμμή */

	/* αρχικοποιηση του πίνακα */
	for(i=0; i<6; i++)
	{
		for(j=0; j<10; j++)
		{
			element[i][j]='\0';
		}
	}

	/*	ελέγχω τη γραμμή για διαστήματα, στηλοθέτες και σημειώ-
σεις με το σύμβολο %
		ό,τι μένει "καθαρό" αποθηκεύεται στον πίνακα element */

	j=0;
	length=strlen(ch);
	for(i=0; i<length; i++)
	{
		c=ch[i];
		if(c=='%') break;
		if(c==' ' || c=='\t') continue;

		if(i>0 && ((ch[i-1]==' ') || (ch[i-1]=='\t')))
		{
			good++;
			j=0;
		}
		element[good][j]=c;
		j++;
	}


	/* έλεγχος για σύμβολα V,I,R,C,L,D,QN,QP,MN,MP
		αρχικά δηλώνω κάποιες μεταβλητές που θα αποθηκεύσω τα
δεδομένα που ξεκαθάρισα παραπάνω */
	char elementSymbol[3];
		for(i=0; i<3; i++) elementSymbol[i]='\0';
	char elementName[6];
		for(i=0; i<6; i++) elementName[i]='\0';
	int node1=-1;
	int node2=-1;
	int node3=-1;
	double value=-1;
	char group='\0';

	/* καταχώρηση συμβόλου, τα αποθηκεύω σαν κεφαλαία έστω και αν
είναι πεζά  */
	if('V'==toupper(element[0][0])) elementSymbol[0]='V';
	if('I'==toupper(element[0][0])) elementSymbol[0]='I';
	if('R'==toupper(element[0][0])) elementSymbol[0]='R';
	if('C'==toupper(element[0][0])) elementSymbol[0]='C';
	if('L'==toupper(element[0][0])) elementSymbol[0]='L';
	if('D'==toupper(element[0][0])) elementSymbol[0]='D';
```

```
	if('Q'==toupper(element[0][0]) && 'N'==toupper(element[0][1]))
{elementSymbol[0]='Q'; elementSymbol[1]='N';}
	if('Q'==toupper(element[0][0]) && 'P'==toupper(element[0][1]))
{elementSymbol[0]='Q'; elementSymbol[1]='P';}
	if('M'==toupper(element[0][0]) && 'N'==toupper(element[0][1]))
{elementSymbol[0]='M'; elementSymbol[1]='N';}
	if('M'==toupper(element[0][0]) && 'P'==toupper(element[0][1]))
{elementSymbol[0]='M'; elementSymbol[1]='P';}


	/* καταχώρηση ονόματος, τα μετατρέπω σε κεφαλαία αν είναι πεζά
*/
	length=strlen(element[0]);
	for(i=0; i<length; i++)
	{
		elementName[i]=toupper(element[0][i]);
	}

	/* καταχώρηση κόμβων, τα μετατρέπω πρώτα σε ακέραιες τιμές */
	node1=atoi(element[1]);
	node2=atoi(element[2]);

	if(strlen(elementSymbol)==1)
	{
		value=atof(element[3]);
	}
	else
	{
		/* καταχώρηση τιμής  */
		node3=atoi(element[3]);
		value=atof(element[4]);
	}

		/* καταχώρηση τιμής για group 2 */
		if(element[4][0]=='G' || element[4][0]=='g') group='G';

	/* δημιουργώ νέο στοιχείο και το προσθέτω στη λίστα  */
	CircuitElement *temp;
	temp=&list;
	while(temp->next!=NULL) temp=temp->next;

	temp->next=(CircuitElement*)malloc(sizeof(CircuitElement));
	temp=temp->next;

	for(i=0; i<3; i++) temp->elementSymbol[i]=elementSymbol[i];
	for(i=0; i<5; i++) temp->elementName[i]=elementName[i];
	temp->node1=node1;
	temp->node2=node2;
	temp->node3=node3;
	temp->value=value;
	temp->group=group;
	temp->next=NULL;

	/* αν όλα καλά επιστρέφω 1 στη  main()  */
	return 1;
}
```

```
int main ()
{
	list.next=NULL;
	char line[LINE_LENGTH];
	FILE *fp;
	int count=0;

	/* ανοιγω το αρχειο για να διαβάσω */
	if ((fp=fopen("data.txt","r"))==NULL)
	{
		puts("Anoigma arxeiou adynato");
		exit(1);
	}

	/* διαβάζω γραμμή γραμμή το αρχείο */
	while (fgets(line, LINE_LENGTH, fp) != NULL)
	{
		count++;
		/* ελέγχω τη κάθε γραμμή */
		checkLine(line);
	}

	fclose(fp);

	/* εκτυπώνω τη λίστα  αν θέλω να ελέγξω κάτι*/
	/*printList();*/

	/*  καλά τη getMNA όπου αρχίζει η δημιουργία των πινάκων και ο
υπολογισμός */
	printList();
	getMNA();
	getchar();
	return 0;
}
```

10. Παράρτημα Β(υλοποίηση μέσω της μεθόδου sparse)

Η διαφορά της προηγούμενης μεθόδου με την καινούργια είναι η δημιουργία της βιβλιοθήκης MNA μέσω της αναπαράστασης των αραιών πινάκων .Η main συνάρτηση παραμένει η ίδια με την προηγούμενη μέθοδο με την διαφορά ότι αυτή την φορά κάνει include την καινούργια βιβλιοθήκη

Σχεδιασμός του πίνακα MNA με την μορφή sparse

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "timer.h"
#include "cs.h"

int ALL_ELEMENTS;     /*αποθήκευση αριθμό στοιχείων*/
int R_ELEMENTS;   /*αποθήκευση αριθμό αντιστάσεων */
int R_GROUP_ELEMENTS; /*αποθήκευση αριθμό αντιστάσεων group 2 */
int V_ELEMENTS; /*αποθήκευση αριθμό πηγών */


/* Εκτυπώνει τιμές πίνακα για πίνακες σαν δεκαδικές τιμές με 3 δεκα-
δικά ψηφία */
void printMatrix(double *pinax, int r, int c)
{
      int i,j;
      printf("\n\n----------------------------------\n\n      ");
      for(i=0; i<r; i++)
      {
            printf("\n");
            for(j=0; j<c; j++)
                  printf("%+5.3f\t", *pinax++);
      }
      printf("\n\n");
}

void addQt(cs * Qt, int i, int j, double value) {
    /* Ψάχνουμε να δούμε αν υπάρχει ήδη στην λίστα συντεταγμένων */
```

```
    int ti;
    for (ti = 0; ti < Qt->nz; ti++) {
        if (Qt->i[ti] == i && Qt->p[ti] == j) {

           Qt->x[ti]+= value;
            return;
        }
    }
    cs_entry(Qt, i, j, value);
}

/* υπολογισμός του πίνακα A */
void setMatrixA(cs * Qt)
{
      int counter;
      int node_1,node_2;



/* επιλέγω πρώτα τις αντιστάσεις απο group 1*/
      CircuitElement *temp;
      temp=&list;
      while(temp->next!=NULL)
      {
            temp=temp->next;
            if(temp->elementSymbol[0]=='R' && temp->group=='\0')
            {
                  node_1=temp->node1;
                  node_2=temp->node2;
                        if(node_1>0)
                    addQt(Qt, node_1-1, node_1-1, 1/temp->value);
                        if(node_2>0)
                    addQt(Qt, node_2-1, node_2-1, 1/temp->value);
                        if(node_1>0 && node_2>0)
                        {
                    addQt(Qt, node_1-1, node_2-1, -1/temp->value);
                    addQt(Qt, node_2-1, node_1-1, -1/temp->value);
                        }
            }
      }

/* επιλέγω μετά τις αντιστάσεις από group 2 */
      temp=&list;
      counter=R_ELEMENTS;
      while(temp->next!=NULL)
      {
            temp=temp->next;
```

```
            if(temp->elementSymbol[0]=='R' && temp->group=='G')
            {
                  node_1=temp->node1;
                  node_2=temp->node2;
            addQt(Qt, counter, counter, -temp->value);
                  if(node_1>0)
                  {
                addQt(Qt, counter, node_1-1, 1);
                addQt(Qt, node_1-1, counter, 1);
                  }
                  if(node_2>0)
                  {
                addQt(Qt, counter, node_2-1, -1);
                addQt(Qt, node_2-1, counter, -1);
                  }
                  counter++;
            }
      }

/* επιλέγω μετά τις πηγές */
      temp=&list;
      counter=ALL_ELEMENTS-V_ELEMENTS;
      while(temp->next!=NULL)
      {

                        temp=temp->next;
            if(temp->elementSymbol[0]=='V')
            {
                  node_1=temp->node1;
                  node_2=temp->node2;
                  if(node_1>0)
                  {
                addQt(Qt, counter, node_1-1, 1);
                addQt(Qt, node_1-1, counter, 1);
                  }
                  if(node_2>0)
                  {
                addQt(Qt, counter, node_2-1, -1);
                addQt(Qt, node_2-1, counter, -1);
                  }
                  counter++;
            }
      }
}
```

```
/* υπολογισμός του πίνακα X */
void setMatrixX(double XM[ALL_ELEMENTS], double ZM[ALL_ELEMENTS])
{
	/* αντιγράφω τον πίνακα στο X από τον Z*/
	int i;
	for(i=0; i<ALL_ELEMENTS; i++) XM[i]=ZM[i];

}
/* υπολογισμός του πίνακα Z */
void setMatrixZ(double ZM[ALL_ELEMENTS])
{
	int i;
	int node_1,node_2;
	int counter;

	/* αρχικοποιώ τον πίνακα */
	for(i=0; i<ALL_ELEMENTS; i++) ZM[i]=0;

/* επιλέγω τις πηγές ρεύματος */
	CircuitElement *temp;
	temp=&list;
	while(temp->next!=NULL)
	{
		temp=temp->next;
		if(temp->elementSymbol[0]=='I')
		{
			node_1=temp->node1;
			node_2=temp->node2;
			if(node_1>0)
			{
				ZM[node_1-1] -= temp->value;
			}
			if(node_2>0)
			{
				ZM[node_2-1] += temp->value;
			}
		}
	}


/* επιλέγω τις πηγές τάσης*/
	temp=&list;
	counter=ALL_ELEMENTS-V_ELEMENTS;
	while(temp->next!=NULL)
	{
		temp=temp->next;
```

```
            if(temp->elementSymbol[0]=='V')
            {
                  ZM[counter] += temp->value;
            }
            counter++;
      }
}


void getMNA()
{
          ALL_ELEMENTS=getAllElements(); /* βρίσκουμε πρώτα αριθμό
στοιχείων */
      R_ELEMENTS=getRElements(); /*  */
      R_GROUP_ELEMENTS=getGroupRElements(); /*  */
      V_ELEMENTS=getVElements(); /*  */

      //double A[ALL_ELEMENTS][ALL_ELEMENTS];  /* Δημιουργία πίνακα
A */
      double X[ALL_ELEMENTS];  /* Δημιουργία πίνακα X, αποθηκεύονται
οι άγνωστες τιμές*/
      double Z[ALL_ELEMENTS];  /* Δημιουργία πίνακα Z */

    cs * Qt = cs_spalloc (0, 0, 1, 1, 1);
    setMatrixA(Qt);   /* υπολογίζουμε τιμές πίνακα για  A */
    cs * A = cs_compress(Qt);
    cs_spfree(Qt);

      setMatrixZ(Z);   /* υπολογίζουμε τιμές πίνακα για Z */
      setMatrixX(X, Z);   /* αντιγράφω τις τιμές του Z στο X */




          /*

            Εκτυπώνω πίνακες αν χρειάζεται να ελέγξω κάτι
      printMatrix(A, ALL_ELEMENTS, ALL_ELEMENTS);
      printMatrix(Z, ALL_ELEMENTS, 1);
      */

      /* εκτυπώνω τον πίνακα A πρίν καλέσω την gauss  */
      //cs_print(Qt,0);
      cs_print(A,0);

      /* εκτυπώνω τη στήλη Z */
```

```
	printMatrix(Z, ALL_ELEMENTS, 1);

	/* καλώ τη lu για τον υπολογισμό της X οι τελικές τιμές θα α-
ποθηκευθούν στο
		X */
		double begin = BeginTimer();
	cs_lusol(0, A, X, 0.001);
		float elapTicks = EndTimer(begin); /*σταμάτημα ρολογιού
	και υπολογισμός   του χρόνου που πέρασε*/
		 float elapMilli, elapSeconds, elapMinutes;
		 elapMilli = elapTicks/1000;     // milliseconds
		 elapSeconds = elapMilli/1000;   // seconds
		 elapMinutes = elapSeconds/60;   // minutes
	/* εκτυπών τον πίνακα A αφού καλέσω την gauss,γίνεται τριγωνι-
κός*/
	/*printMatrix(A, ALL_ELEMENTS, ALL_ELEMENTS);*/
	/* εκτυπών τη στήλη X με τις τελικές τιμές */
	printMatrix(X, ALL_ELEMENTS, 1);
	 cs_spfree(A);
}
```

11.Παράρτημα Γ(κώδικας με την μέθοδο partitioning σε matlab)

Δημιουργία του δικτύου τροφοδοσίας:

```
n1=5; %number of vertical lines in the grid
n2=5; %number of horizontal lines in the grid
vn=2; %number of voltage sources

n=n1*n2; %number of nodes
b=n1*(n2-1)+n2*(n1-1);%+vn; %number of branches

nper=randperm(n);
ivn=nper(:,1:vn); %node identifiers (random) where voltage sources
are placed

rsh=0.1; %horizontal sheet resistance (in Ohms/sq)
rsv=0.1; %vertical sheet resistance (in Ohms/sq)
lsh=0; %horizontal inductance per unit length (in pH/um)
lsv=0; %vertical inductance per unit length (in pH/um)
sizv=350; %vertical size of chip (in um)
sizh=350; %horizontal size of chip (in um)
pv=sizh/(n1-1); %pitch of vertical lines
ph=sizv/(n2-1); %pitch of horizontal lines
rpin=50; %Vdd pin resistance in Ohms
lpin=100; %pin inductance (in pH)
cpph=1e-4;   %horizontal cpp capacitance in pF/um^2
cppv=1e-4;   %vertical cpp capacitance in pF/um^2
cffh=1e-4;   %horizontal cff capacitance in pF/um
cffv=1e-4;   %vertical cff capacitance in pF/um
cpin=1;  %pin capacitance in pF
w=ones(n2+n1,1); %vector of widths of horizontal and vertical lines
h=10; %sampling step (in ps)

%capacitance and incidence matrix
ch=repmat(cpph*pv*w(1:n2,:)+cffh*pv,1,n1-1);   %horizontal branch
capacitances
cv=repmat(cppv*ph*w(n2+1:n2+n1,:)+cffv*ph,1,n2-1); %vertical branch
capacitances

C=spdiags(cpin*ones(n,1),0,n,n);%sparse(n,n);
Al=sparse(n,b);
%enumerate horizontal branches
for i=1:n2
    for j=1:n1-1
        ni1=(i-1)*n1+j; %node indices for current branch
```

```
        ni2=ni1+1;
        ni3=(i-1)*(n1-1)+j; %branch index conversion from 2D to 1D
        %C(ni1,ni1)=C(ni1,ni1)+(1/2)*ch(i,j);
        %C(ni2,ni2)=C(ni2,ni2)+(1/2)*ch(i,j);
        Al(ni1,ni3)=1;
        Al(ni2,ni3)=-1;
    end
end
%enumerate vertical branches
for i=1:n1
    for j=1:n2-1
        ni1=(j-1)*n1+i; %node indices for current branch
        ni2=ni1+n1;
        ni3=(i-1)*(n2-1)+j+n2*(n1-1); %branch index conversion from
2D to 1D
        %C(ni1,ni1)=C(ni1,ni1)+(1/2)*cv(i,j);
        %C(ni2,ni2)=C(ni2,ni2)+(1/2)*cv(i,j);
        Al(ni1,ni3)=1;
        Al(ni2,ni3)=-1;
    end
end
%enumerate supply branches
% for i=1:vn
%     C(ivn(i),ivn(i))=C(ivn(i),ivn(i))+cpin;
%     Al(ivn(i),n1*(n2-1)+n2*(n1-1)+i)=-1;
% end

%inductance matrix
lh=repmat(lsh*pv*log(150./w(1:n2,:)),1,n1-1); %horizontal branch in-
ductances
lv=repmat(lsv*ph*log(150./w(n2+1:n2+n1,:)),1,n2-1); %vertical branch
inductances

lb=[reshape(lh',n2*(n1-1),1);reshape(lv',n1*(n2-
1),1)];%;lpin*ones(vn,1)];

%L=diag(lb);

%resistance matrix
rh=repmat(rsh*pv./w(1:n2,:),1,n1-1); %horizontal branch resistances
rv=repmat(rsv*ph./w(n2+1:n2+n1,:),1,n2-1); %vertical branch re-
sistances

rb=[reshape(rh',n2*(n1-1),1);reshape(rv',n1*(n2-
1),1)];%;rpin*ones(vn,1)];
%R=diag(rb);
 %system matrix for transient analysis
```

```
RLcol=1./(rb+lb/h);
RL=spdiags(RLcol,0,b,b);
A=Al*RL*Al'+C/h;


r=ones(n,1);
```

To test(test.m)

```
clear;
gridsparse();

if 0
    %Proairetiki dimiourgia tixaiou r
    r = rand(length(A),1)*100;
end

if 0
    %Grapsimo tou r kai tou a se csv's & mat file
    save 'r.mat' r;
    csvwrite('r.csv',r);
    [i,j,val] = find(A);
    csvwrite('A.csv', [i,j,val]);
elseif 1
    %diavasma tou r apo mat file
    load 'r.mat'
end

hdom = 2; % 2 orizontai domains
vdom = 2; % 2 katakorifa domains
sum1=0;
for n = 1:1000
    minTime1=Inf;
    tStart1=tic;
    domain = d_prepare( A, hdom, vdom, n1, n2);
    tElapsed1=toc(tStart1);
    minTime1=min(tElapsed1,minTime1);
    sum1=sum1+minTime1;
end
avg1=sum1/1000;

%emfanisi ton dio pinakon, A kai AP
if 0
    subplot(1,2,1);
    spy(A);
    subplot(1,2,2);
    spy(domain.AP);
end

%emfanisi ton pinakon - Ai,Bi,Ci
if 0
    figure();
    d_plot_matrix(domain)
end

% lisi mias atixeas periptosis
sum=0;
for n=1:1000
    minTime=Inf;
    tStart=tic;
    [rx,bi] = d_solve( domain, r );
    tElapsed=toc(tStart);
    minTime=min(tElapsed,minTime);
    sum=sum+minTime;
end
```

```
avg=sum/1000;

%ipologismos sfalmatos se sxesi me apli diairesi tou Matlab
norm(A\r - rx)
```

Συνάρτηση υπολογισμού διανύσματος μεταθέσεων (perm.m)

```
function [g,x,tearsz] = perm(n, hpartitions,vpartitions)


n1 = n;
n2 = n;


vstep = floor(n1/vpartitions);
hstep = floor(n2/hpartitions);

% * * * x * * *
% * * * x * * *
% * * * x * * *
% x x x x x x x
% * * * x * * *
% * * * x * * *
% * * * x * * *

%Ta megethi ton sets ston pinaka


g = zeros(n1*n2,1);

%Ta sets...
cnt = 1;
for i = 1:vpartitions
    for j = 1:hpartitions
        %arxi tou block
        off0 = 1 + (i-1) * ((vstep+1)*n2) + (j-1)*(hstep+1);
        %dimiourgia tou block
        for k = 1:vstep
            for l = 1:hstep
                offset = off0   + (k-1) * n2; % Katakorifi thesi me-
sa sto block
                offset = offset + (l-1);      % Orizontia thesi mesa
sto block
                g(cnt) = offset;
                cnt=cnt+1;
            end
        end
    end
end
%Kai to tearing set,

tearsz=(n1*n2)-cnt+1;

%orizontios...
```

```
for i = 1:(vpartitions-1)
    %katakorifo offset iso me to partition pou eksetazoume + tin ex-
tra
    %grammi tou tearing set
    offset = 1 + i * vstep * n2 + (i-1) * n2;
    for j=1:n2
        g(cnt) = offset+j-1;
        cnt=cnt+1;
    end
end

%kai kathetos...
for i = 1:(hpartitions-1)
    %orizontion offset iso me to partition pou eksetazoume + tin ex-
tra
    %grammi tou tearing set
    offset = 1 + i * hstep + (i-1);
    for j=1:n1
        %paraleipoume tous komvous pou exoun eidi mpei apo to ori-
zontio
        if (rem(j,vstep+1)~=0)
            g(cnt) = offset+(j-1)*n2;
            cnt=cnt+1;
        end
    end
end

x(g)=1:length(g);
```

Συνάρτηση προετοιμασίας domain decomposition (d_prepare.m)

```
function d = d_prepare( A, hdom, vdom, n1, n2)
%D_PREPARE Dimiourgei tis metavlites gia to domain decomposition
%   Auto xreiazetai na ginei mono mia fora gia ton pinaka tou dik-
tiou
%   trofodosias. Meta mporoume na linoume to sistima me xrisi tis
%   metavlitis d pou epistrefetai kai i opoia periexei olous tous
%   ipopinakes. Episis polles apo tis anastrofes pinakon ginontai se
auto
%   to vima opote i teliki epeksergasia gia kathe dianisma B einai
%   paralilisimi kai arketa grigori,

d.n1 = n1;
d.n2 = n2;
d.hdom = hdom;
d.vdom = vdom;

%dimiourgia pinaka me permutations
[d.prem, d.xperm, d.tearsz] = perm(n1, hdom, vdom);
```

```
%mapping tou provlimatos meta tis anadiatakseis
d.AP = A(d.prem,d.prem);

% Algorithmos domain Decomposition

d.nsize = n1*n2;
d.domcnt = hdom*vdom;
d.tear_starts = d.nsize-d.tearsz+1;
d.subsz = (d.tear_starts-1)/d.domcnt;

% ------------------------------------------------------------------
-------
% Eksagogi olon ton parametron me vasi tin enotita 3.4 tou Circuti
Simulation, 2010, N. Najm
% ------------------------------------------------------------------
-------

offset = 1;
for i=1:d.domcnt
    %eksagogi ton subdomain apo ton matrix
    d.Ai{i}=d.AP(offset:offset+d.subsz-1,offset:offset+d.subsz-1);
    d.Ain{i}=inv(d.Ai{i});
    [Li,Ui] = lu(d.Ai{i});
    d.Lin{i}=inv(Li);
    d.Ui{i}=Ui;
    offset=offset+d.subsz;
end

offset = 1;
for i=1:d.domcnt
    %eksagogi ton subdomain apo ton matrix
    d.Bi{i}=d.AP(offset:offset+d.subsz-1,
d.tear_starts:d.tear_starts+d.tearsz-1);
    d.LinBi{i}=d.Lin{i}*d.Bi{i};
    offset=offset+d.subsz;
end

offset = 1;
for i=1:d.domcnt
    %eksagogi ton subdomain apo ton matrix
    d.Ci{i}=d.AP(d.tear_starts:d.tear_starts+d.tearsz-1, off-
set:offset+d.subsz-1);
    offset=offset+d.subsz;
end

d.Dt = d.AP(d.tear_starts:d.tear_starts+d.tearsz-1,
d.tear_starts:d.tear_starts+d.tearsz-1);

%eksisosi 3.216
DtStar = d.Dt;
for i=1:d.domcnt
    DtStar = DtStar - d.Ci{i}*d.Ain{i}*d.Bi{i};
end

d.invDtStar = inv(DtStar);
```

Συνάρτηση υπολογισμού λύσης domain decomposition (d_solve.m)

```
function [rx, bi] = d_solve( d, b )
%D_SOLVE Linei to sistima gia dedomeno dianisma b
%   Auti edo ekteleitai kathe fora pou theloume na lisoume to dian-
isma gia
%   kainouries times tou b. Paratiroume oti de xreiazetai kamia
anastrofi
%   pinaka alla mono aploi pollaplasiasmoi pinakon kai diansimaton.
Episis
%   ta dio prota for loops mporoune na treksoun parallila se pol-
laplous
```

```
%   ipologistes mias kai den exoun alliloeksartiseis (ta domains
einai
%   aneksartita). Auto simainei oti to provlima mporei pleon na
%   parallilistei se hdom*vdom epeksergastes kai m'auton ton tropo
mporoume
%   na epeksergastoume akoma megalitera diktia.

bP = b(d.prem);

%-------------------------------------------------------------
%eksagogi ton subdomain apo to vector b kai eksisosi 3.217
bt = bP(d.tear_starts:d.tear_starts+d.tearsz-1);
btStar = bt;

offset = 1;
for i=1:d.domcnt

    bi{i}=bP(offset:offset+d.subsz-1);

    btStar = btStar - d.Ci{i}*d.Ain{i}*bi{i};

    offset=offset+d.subsz;
end
%-------------------------------------------------------------

%eksisosi 3.216
xt = d.invDtStar*btStar;

for i=1:d.domcnt
    %eksisosi 3.213 (argi)
    %xi{i} = d.Ain{i} * (bi{i}-d.Bi{i}*xt);
    %eksisosi 3.221 (grigori an kanoume mono backwards substi-
tuntion)
    xi{i} = d.Ui{i} \ ((d.Lin{i} *  bi{i}) - (d.LinBi{i} * xt));
end

%ensomatosi se ena vector - tin teliki lisi rx
rx = zeros(d.nsize,1);
offset = 1;
for i=1:d.domcnt
    rx(offset:offset+d.subsz-1) = xi{i};
    offset=offset+d.subsz;
end
rx(d.tear_starts:d.tear_starts+d.tearsz-1) = xt;

%Anadiataksi me vasi ton xperm oste na einai oi metavlites stin sei-
ra tou
%arxikou pinaka A
rx = rx(d.xperm);
```

12.Παράρτημα Δ (κώδικας με την μέθοδο partitioning σε C)

Άνοιγμα αρχείων (read_mat.h)

```
#ifndef READ_MAT_H
#define READ_MAT_H

#include "cs.h"

// Gia kanonikous pinakes:
// csvwrite('ap.csv',full(domain.AP));

double* read_matrix(char * file, int *width, int *height);


// Gia araious pinakes:
// [i,j,val] = find(domain.AP);
// csvwrite('ap-sparse.csv', [i,j,val]);

cs* read_sparse(char * file);


void print_matrix(double*tt, int width, int height);

#endif
```

Άνοιγμα αρχείων (read_mat.c)

```
#include "read_mat.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Gia kanonikous pinakes:
// csvwrite('ap.csv',full(domain.AP));

double* read_matrix(char * file, int *width, int *height)
{
    FILE * f = fopen(file, "r");

    if (!f)
    {
        fprintf(stderr, "Can't open file\n");
        return 0;
    }

    //1. test
    *width = -1;
    *height = 0;
    char buf[1024];
    while (fgets(buf, 1024, f))
    {
        int lineWidth = 0;
        char * pch = strtok(buf, ",");
        while (pch != NULL)
        {
            lineWidth++;

            //Afairesi xaraktira termatismou grammis
            if (strstr(pch, "\n") != 0) *strstr(pch, "\n") = 0;

            double g;
            if (sscanf(pch, "%lf", &g) != 1)
            {
                fprintf(stderr, "To '%s' den einai tipou double\n", pch);
                return 0;
            }

            pch = strtok(NULL, ",");
        }

        if (*width != lineWidth && *width != -1)
        {
```

```
            fprintf(stderr, "oi pinakas exei diaforetiko arithmo stoixeion
se kathe grammi\n");
            return 0;
        }

        *width = lineWidth;
        (*height)++;
    }

    //2. read
    fseek(f, 0, SEEK_SET);

    double *k = (double*) malloc((*width) * (*height) * sizeof (double));
    int i = 0;
    while (fgets(buf, 1024, f))
    {
        char * pch = strtok(buf, ",");

        //Afairesi xaraktira termatismou grammis
        if (strstr(pch, "\n") != 0) *strstr(pch, "\n") = 0;

        while (pch != NULL)
        {
            sscanf(pch, "%lf", k + i);
            pch = strtok(NULL, ",");
            i++;
        }
    }

    fclose(f);

    return k;
}

// Gia araious pinakes:
// [i,j,val] = find(domain.AP);
// csvwrite('ap-sparse.csv', [i,j,val]);

cs* read_sparse(char * file)
{
    int width, height, i;
    double * g = read_matrix(file, &width, &height);
    if (!g) return 0;

    if (width != 3)
    {
        free(g);
        fprintf(stderr, "Perimenoume tris stiles, to i, to j kai ti ti-
mi\n");
        return 0;
    }
    cs * tmp = cs_spalloc(0, 0, 1, 1, 1);
    if (!tmp)
    {
        fprintf(stderr, "sfalma stin cs_spalloc\n");
        free(g);
        return 0;
    }

    for (i = 0; i < height; i++)
    {
        //                                  v-- Metatropi apo tis
grammes/stiles me vasi to 1 tou matlab se g/s vasi 0
        if (!cs_entry(tmp, (int) g[width * i] - 1, (int) g[width * i + 1] -
1, g[width * i + 2]))
        {
            fprintf(stderr, "sfalma stin cs_entry\n");
            free(g);
            cs_spfree(tmp);
            return 0;
        }
    }

    free(g);

    cs * rv = cs_compress(tmp);
    if (!rv)
```

```
    {
        fprintf(stderr, "sfalma stin cs_compress\n");
        cs_spfree(tmp);
        return 0;
    }

    cs_spfree(tmp);

    return rv;
}

void print_matrix(double*tt, int width, int height)
{
    if (tt)
    {
        int i, j;
        for (i = 0; i < height; i++)
        {
            for (j = 0; j < width; j++)
            {
                printf("%3.4f ", tt[j + i * height]);
            }
            printf("\n");
        }
    }
}
```

Συνάρτηση main (main.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "read_mat.h"
#include "ddcomp.h"

#ifdef ENABLE_TIMING
timers tm;
#endif

int main()
{
    int rw, rh, i;
    double* r = read_matrix("r.csv", &rw, &rh);

    cs* A = read_sparse("A.csv");

    int hdom = 2; // 2 orizontai domains
    int vdom = 2; // 2 katakorifa domains

    int n = (int) sqrt(A->m);

    domain_desc domain = d_prepare(A, hdom, vdom, n, n);

    double* rx = d_solve( domain, r);

    printf("rx_=[");
    for (i = 0; i < n*n; i++)
        printf("%f%s", rx[i], i==(n*n-1) ? "" : ";");
    printf("];\n");

    #ifdef ENABLE_TIMING
    float elap1 = (float)(tm.toc1-tm.tic1)/CLOCKS_PER_SEC;
    float elap2 = (float)(tm.toc2-tm.tic2)/CLOCKS_PER_SEC;
    float elap3 = (float)(tm.toc3-tm.tic3)/CLOCKS_PER_SEC;
    float elap4 = (float)(tm.toc4-tm.tic4)/CLOCKS_PER_SEC;


    printf("##\tn\thdom\tvdom\ttearsz\tt_prepare\t_subdomain\t_tearing\t_
    solve_total\n");
        printf("##\t%d\t%d\t%d\t%d\t%f\t%f\t%f\t%f\n", n, hdom, vdom, do-
    main.tearsz, elap1, elap2/domain.domcnt, elap3, elap4);
    #endif
```

```
    free_domain(&domain);
    cs_spfree(A);
    free(r);
    free(rx);

    return 0;
}
```

Συναρτήσεις επίλυσης domain decomposition (ddcomp.h)

```
#ifndef DDCOMP_H
#define DDCOMP_H

#define ENABLE_TIMING

#ifdef ENABLE_TIMING
    #include <time.h>
    typedef struct {
        clock_t tic1, toc1, tic2, toc2, tic3, toc3, tic4, toc4;
    } timers;
    extern timers tm;
#endif

#include "cs.h"

typedef struct
{
    csi * g;
    csi * x;
    int tearsz;
} perm_set;

perm_set perm(int nss, int hpartitions, int vpartitions);

typedef struct
{
    int n1;
    int n2;
    int hdom;
    int vdom;

       csi * prem;
       csi * xperm;
       int tearsz;

       cs* AP;

    cs**Ai;
    cs**Ain;
    cs**Bi;
       cs**LinBi;
    cs**Ci;
    cs* Dt;
       cs* invDtStar;

    cs**Lin;
    cs**Ui;

    int nsize;
    int domcnt;
    int tear_starts;
    int subsz;

} domain_desc;

//Proetoimasia domis domain_desc gia to sistima
domain_desc d_prepare( cs* A, int hdom, int vdom,  int n1, int n2);

//Epilisi gia dedomeno b
double* d_solve( domain_desc d, double * b);

//Anastrofi pinaka A me tin xrisi LU
cs * inv(cs*A);

//Epeleutherosi mnimis tou domain
```

```
void free_domain(domain_desc * d);

#endif
```

Συναρτήσεις επίλυσης domain decomposition (ddcomp.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "ddcomp.h"
#include "read_mat.h"

#define LU_TOLERANCE  0.001
#define LU_ORDER  0
#define XT_CALC_METHOD_INV 0

perm_set perm(int nss, int hpartitions, int vpartitions)
{
    int i, j, k, l;
    perm_set rv;

    int n1 = nss;
    int n2 = nss;


    int vstep = floor(n1 / vpartitions);
    int hstep = floor(n2 / hpartitions);

    // * * * x * * *
    // * * * x * * *
    // * * * x * * *
    // x x x x x x x
    // * * * x * * *
    // * * * x * * *
    // * * * x * * *

    // Ta megethi ton sets ston pinaka


    rv.g = (csi*) malloc(n1 * n2 * sizeof (csi));
    rv.x = (csi*) malloc(n1 * n2 * sizeof (csi));

    // Ta sets...
    int cnt = 0;
    for (i = 1; i <= vpartitions; i++)
    {
        for (j = 1; j <= hpartitions; j++)
        {
            // arxi tou block
            int off0 = 1 + (i - 1) * ((vstep + 1) * n2) + (j -
1)*(hstep + 1);
            // dimiourgia tou block
            for (k = 1; k <= vstep; k++)
            {
                for (l = 1; l <= hstep; l++)
                {
                    int offset = off0 + (k - 1) * n2;
                    // Katakorifi thesi mesa sto block
                    offset = offset + (l - 1);
                    // Orizontia thesi mesa sto block
                    rv.g[cnt] = offset - 1; // <- metatropi se pin-
akes me vasi to 0 anti gia to 1
                    cnt++;
                }
            }
        }
```

```
    }

    // Kai to tearing set,
    rv.tearsz = (n1 * n2) - cnt + 1 - 1; // <- metatropi se pinakes
me vasi to 0 anti gia to 1

    // orizontios...
    for (i = 1; i <= (vpartitions - 1); i++)
    {
        // katakorifo offset iso me to partition pou eksetazoume +
tin extra
        // grammi tou tearing set
        int offset = 1 + i * vstep * n2 + (i - 1) * n2;
        for (j = 1; j <= n2; j++)
        {
            rv.g[cnt] = offset + j - 1 - 1; // <- metatropi se pin-
akes me vasi to 0 anti gia to 1
            cnt++;
        }
    }

    // kai kathetos...
    for (i = 1; i <= (hpartitions - 1); i++)
    {
        // orizontion offset iso me to partition pou eksetazoume +
tin extra
        // grammi tou tearing set
        int offset = 1 + i * hstep + (i - 1);
        for (j = 1; j <= n1; j++)
        {
            // paraleipoume tous komvous pou exoun eidi mpei apo to
orizontio
            if (j % (vstep + 1))
            {
                rv.g[cnt] = offset + (j - 1) * n2 - 1; // <- meta-
tropi se pinakes me vasi to 0 anti gia to 1
                cnt++;
            }
        }
    }

    for (i = 0; i < n1 * n2; i++)
    {
        rv.x[rv.g[i]] = i;
    }

    return rv;
}

cs* vec_to_sparse(double*v, int nsize) {
    int i;
    cs*r = cs_spalloc(1, nsize, 1, 1, 1);
    for (i=0;i<nsize;i++) {
        cs_entry(r, 0, i, v[i]);
    }
    cs * rv = cs_compress(r);
    cs_spfree(r);
    return rv;
}

double* sparse_to_vec(cs*A) {
    int j, p;
    csi*Ap = A->p;
    csi*Ai = A->i;
    double *Ax = A->x;

    double * out = (double*)malloc(A->n * sizeof(double));

    for (j = 0; j < A->n; j++) out[j] = 0.0;
```

```
    for (j = 0; j < A->n; j++)
    {
        for (p = Ap [j]; p < Ap [j + 1]; p++)
        {
            int row = Ai[p];
            if (row==0) {
                out[j] = Ax ? Ax [p] : 1;
            }
        }
    }

    return out;
}

cs* mat_extract(cs*A, int fx, int tx, int fy, int ty)
{
    int j, p;
    csi*Ap = A->p;
    csi*Ai = A->i;
    double *Ax = A->x;

    cs*r = cs_spalloc(ty - fy + 1, tx - fx + 1, 1, 1, 1);

    for (j = fx; j <= tx; j++)
    {
        for (p = Ap [j]; p < Ap [j + 1]; p++)
        {
            int row = Ai[p];
            if (row >= fy && row <= ty)
            {
                cs_entry(r, row - fy, j - fx, Ax ? Ax [p] : 1);
            }
        }
    }

    cs * rv = cs_compress(r);
    cs_spfree(r);

    return rv;
}

//Anastrofi pinaka A me tin xrisi LU

cs * inv(cs*A)
{
    int i, j;
    cs*r = cs_spalloc(0, 0, 1, 1, 1);

    int mn = A->n;

    double * b = (double*) malloc(mn * sizeof (double));

    for (i = 0; i < mn; i++)
    {
        //Gia kathe stili

        //Arxikopoiisi dianismatos b
        for (j = 0; j < mn; j++) b[j] = 0;
        b[i] = 1;

        //Epilisi tou A * x = eye(); me tin methodo LU gi'autin tin
stili
        cs_lusol(LU_ORDER, A, b, LU_TOLERANCE);

        //Prosthiki tis lisis sto apotelesma tou araiou pinaka
        for (j = 0; j < mn; j++)
        {
            double dv = b[j] > 0 ? b[j] : -b[j];
```

```
            if (dv >= 0.0005)
            {
                cs_entry(r, j, i, b[j]);
            }
        }
    }
    free(b);

    cs * rv = cs_compress(r);
    cs_spfree(r);

    return rv;
}

//Epeleutherosi mnimis tou domain
void free_domain(domain_desc * d) {
    int i;
    for (i=0;i<d->domcnt;i++) {
        if (d->Ai[i]) cs_spfree(d->Ai[i]);
        if (d->Ain[i]) cs_spfree(d->Ain[i]);
        if (d->Bi[i]) cs_spfree(d->Bi[i]);
        if (d->Ci[i]) cs_spfree(d->Ci[i]);
        if (d->Lin[i]) cs_spfree(d->Lin[i]);
        if (d->Ui[i]) cs_spfree(d->Ui[i]);
        if (d->LinBi[i]) cs_spfree(d->LinBi[i]);
    }

    free(d->Ai);
    free(d->Ain);
    free(d->Bi);
    free(d->Ci);
    free(d->Lin);
    free(d->Ui);
    free(d->LinBi);

    cs_spfree(d->AP);
    cs_spfree(d->Dt);
    cs_spfree(d->invDtStar);

    free(d->prem);
    free(d->xperm);

    d->Ai=0;
    d->Ain=0;
    d->Bi=0;
    d->Ci=0;
    d->Lin=0;
    d->Ui=0;
    d->LinBi=0;
}

domain_desc d_prepare(cs* A, int hdom, int vdom, int n1, int n2)
{
    domain_desc d;
    int offset, i;

    #ifdef ENABLE_TIMING
    tm.tic1 = clock();
    #endif


    d.n1 = n1;
    d.n2 = n2;
    d.hdom = hdom;
    d.vdom = vdom;

    //dimiourgia pinaka me permutations
    perm_set ps = perm(n1, hdom, vdom);
    d.prem = ps.g;
```

```
    d.tearsz = ps.tearsz;

    //mapping tou provlimatos meta tis anadiatakseis
    d.AP = cs_permute(A, d.xperm, d.prem, 1);


    // Algorithmos domain Decomposition

    d.nsize = n1*n2;
    d.domcnt = hdom*vdom;
    d.tear_starts = d.nsize - d.tearsz + 1;
    d.subsz = (d.tear_starts - 1) / d.domcnt;

    d.Ai = (cs**) malloc(d.domcnt * sizeof (cs*));
    d.Ain = (cs**) malloc(d.domcnt * sizeof (cs*));
    d.Bi = (cs**) malloc(d.domcnt * sizeof (cs*));
    d.Ci = (cs**) malloc(d.domcnt * sizeof (cs*));
    d.LinBi = (cs**) malloc(d.domcnt * sizeof (cs*));
    d.Lin = (cs**) malloc(d.domcnt * sizeof (cs*));
    d.Ui = (cs**) malloc(d.domcnt * sizeof (cs*));

    // ------------------------------------------------------------
------------
    // Eksagogi olon ton parametron me vasi tin enotita 3.4 tou Cir-
cuti Simulation, 2010, N. Najm
    // ------------------------------------------------------------
------------

    offset = 1;
    for (i = 0; i < d.domcnt; i++)
    {
        //eksagogi ton subdomain apo ton matrix
        d.Ai[i] = mat_extract(d.AP, offset - 1, offset + d.subsz - 1
- 1, offset - 1, offset + d.subsz - 1 - 1);
        d.Ain[i] = inv(d.Ai[i]);

        //MATLAB: [Li,Ui] = lu(d.Ai{i});
        css *S = cs_sqr(LU_ORDER, d.Ai[i], 0); //Kodikas apo tin
cs_lusol
        csn *lun = cs_lu(d.Ai[i], S, LU_TOLERANCE);
        //MATLAB: d.Lin{i}=inv(Li);
        d.Lin[i] = inv(lun->U); //Gia kapoio logo ta L&U einai anap-
oda
        d.Ui[i] = cs_add(lun->L, lun->L, 1.0, 0); //Kolpo gia na ka-
noume antigrafo tou L

        cs_nfree(lun);
        cs_sfree(S);

        offset += d.subsz;
    }

    offset = 1;
    for (i = 0; i < d.domcnt; i++)
    {
        //eksagogi ton subdomain apo ton matrix
        d.Bi[i] = mat_extract(d.AP, offset - 1, offset + d.subsz - 1
- 1, d.tear_starts - 1, d.tear_starts + d.tearsz - 1 - 1);

        //MATLAB: d.LinBi{i}=d.Lin{i}*d.Bi{i};
        d.LinBi[i] = cs_multiply(d.Bi[i], d.Lin[i]);

        offset += d.subsz;
    }

    offset = 1;
    for (i = 0; i < d.domcnt; i++)
    {
```

```
        //eksagogi ton subdomain apo ton matrix
        d.Ci[i] = mat_extract(d.AP, d.tear_starts - 1, d.tear_starts
+ d.tearsz - 1 - 1, offset - 1, offset + d.subsz - 1 - 1);
        offset += d.subsz;
    }


    d.Dt = mat_extract(d.AP, d.tear_starts - 1, d.tear_starts +
d.tearsz - 1 - 1, d.tear_starts - 1, d.tear_starts + d.tearsz - 1 -
1);

    //eksisosi 3.216
    cs* DtStar = d.Dt;
    for (i = 0; i < d.domcnt; i++)
    {
        //MATLAB: DtStar = DtStar - d.Ci{i}*d.Ain{i}*d.Bi{i};
        cs* t1 = cs_multiply(d.Ain[i], d.Ci[i]);
        cs* t2 = cs_multiply(d.Bi[i], t1); // Me anapodi seira apo
to matlab
        cs *t3 = cs_add(DtStar, t2, 1.0, -1.0);

        cs_spfree(t1);
        cs_spfree(t2);
        if (i != 0)
        {
            //Den kanoume free to proto DtStar giati deixnei sto
d.Dt (ligo parapano)
            cs_spfree(DtStar);
        }
        DtStar = t3;
    }

    d.invDtStar = inv(DtStar);
    cs_spfree(DtStar);

    #ifdef ENABLE_TIMING
    tm.toc1 = clock();
    #endif

    return d;
}

//Epilisi gia dedomeno b
double* d_solve( domain_desc d, double * b) {
    int j;
    double* rx = (double*)malloc(d.nsize*sizeof(double));


    #ifdef ENABLE_TIMING
    tm.tic3 = tm.tic4 = clock();
    #endif


    cs* ib = vec_to_sparse(b, d.nsize);
    cs* Bp = cs_permute(ib, d.xperm, d.prem, 1);
    cs_spfree(ib);

    cs* bt = mat_extract(Bp, d.tear_starts-1,
d.tear_starts+d.tearsz-1-1, 0, 0);

    cs*btStar = bt;

    cs** bi = (cs**)malloc(d.domcnt*sizeof(cs*));

    int offset = 1, i;
    for (i = 0; i < d.domcnt; i++)
    {
        //eksagogi ton subdomain apo to vector b kai eksisosi 3.217
        bi[i] = mat_extract(Bp, offset-1, offset+d.subsz-1-1, 0, 0);
```

```
        //MATLAB: btStar = btStar - d.Ci{i}*d.Ain{i}*bi{i};
        cs* t1 = cs_multiply(d.Ain[i], d.Ci[i]);
        cs* t2 = cs_multiply(bi[i], t1); // Me anapodi seira apo to
matlab
        cs *t3 = cs_add(btStar, t2, 1.0, -1.0);

        cs_spfree(t1);
        cs_spfree(t2);
        cs_spfree(btStar);

        btStar = t3;

        offset += d.subsz;
    }
    cs_spfree(Bp);

    // eksisosi 3.216
    cs * xt = cs_multiply(btStar, d.invDtStar);
    cs_spfree(btStar);

    #ifdef ENABLE_TIMING
    tm.tic2 = tm.toc3 = clock();
    #endif


    double** xi = (double**)malloc(d.domcnt*sizeof(double*));
    for (i = 0; i < d.domcnt; i++)
    {
        cs* t1, *t2, *t3;
        if (XT_CALC_METHOD_INV) {
            //eksisosi 3.213 (argi)
            //MATLAB: xi{i} = d.Ain{i} * (bi{i}-d.Bi{i}*xt);
            t1 = cs_multiply(xt, d.Bi[i]);
            t2 = cs_add(bi[i], t1, 1.0, -1.0);
            t3 = cs_multiply(t2, d.Ain[i]);
            xi[i] = sparse_to_vec(t3);
        }
        else {
            //eksisosi 3.221 (grigori an kanoume mono backwards sub-
stituntion)
            //MATLAB: xi{i} = d.Ui{i} \ ((d.Lin{i} *  bi{i}) -
(d.LinBi{i} * xt));
            t1 = cs_multiply(bi[i], d.Lin[i]);
            t2 = cs_multiply(xt, d.LinBi[i]);
            t3 = cs_add(t1, t2, 1.0, -1.0);
            xi[i] = sparse_to_vec(t3);

            cs*Uit = cs_transpose (d.Ui[i], 1); //Gia kapoio peri-
ergo logo xreiazetai transpose
            cs_lusol(LU_ORDER, Uit, xi[i], LU_TOLERANCE);
            cs_spfree(Uit);
        }

        cs_spfree(t1);
        cs_spfree(t2);
        cs_spfree(t3);
    }

    #ifdef ENABLE_TIMING
    tm.toc2 = clock();
    #endif



    offset = 1;
    for (i = 0; i < d.domcnt; i++)
    {
        int j;
```

```
        for (j=0;j<d.subsz;j++) {
                rx[d.prem[offset-1]] = xi[i][j];
                offset++;
        }
    }
    double* dxt = sparse_to_vec(xt);
    for (i = 0; i < d.tearsz; i++)
    {
        rx[d.prem[offset-1]] = dxt[i];
        offset++;
    }

    for (i = 0; i < d.domcnt; i++)
    {
        cs_spfree(bi[i]);
        free(xi[i]);
    }
    free(bi);
    free(xi);
    free(dxt);
    cs_spfree(xt);

    #ifdef ENABLE_TIMING
    tm.toc4 = clock();
    #endif

    return rx;
}
```

13. Βιβλιογραφία-Αναφορές

[1] Circuit Simulation Farid N. Najm

[2] Techniques for Circuit Simulation Mahesh B. Patil

[3] Parallel domain decomposition for simulation of large-scale power grids

[4] Direct Methods for Sparse Linear Systems Timothy A. Davis

[5] http://qucs.sourceforge.net/tech/node14.html#eq:MNAfull

[6] Parallel Direct Methods for Block-Diagonal-Bordered Sparse Matrices

[7] Σημειώσεις από το μάθημα "Προσομοίωση Κυκλωμάτων" Διδάσκων κ.Ευμορφόπουλος

[8] http://nptel.iitm.ac.in/courses/Webcourse-contents/IIT-KANPUR/mathematics-2/node18.html

[9] http://www.ee.ucla.edu/~vandenbe/103/lectures/chol.pdf

[10] Ανάλυση Κυκλωμάτων και Σημάτων Giorgio Rizzoni

[11] Σημειώσεις από το μάθημα "Σχεδίαση Μικροεπεξεργαστών" Διδάσκων κ.Ευμορφόπουλος