# Μοντερνες Αριθμητικες μεθοδοι επιλυσης των γραμμικων συστηματων της Ρευστοδυναμικης.

Μεταπτυχιακη Εργασια

**Εμμανουηλ Θ. Ψυχαρης**

Επιβλεπων Καθηγητης: **Θεοδωρος Κατσαουνης**

Επιτροπη Αξιολογησης:
**Μαρκος Κατσουλακης**
**Γεωργιος Ζουραρης**
**Θεοδωρος Κατσαουνης**

Διατμηματικο Προγραμμα Μεταπτυχιακων Σπουδων
«Μαθηματικα και Εφαρμογες τους»

Τμημα Μαθηματικων
και
Τμημα Εφαρμοσμενων Μαθηματικων

**Πανεπιστημιο Κρητης**

Απριλιος 2010

# 0.1 Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους ανθρώπους εκείνους που, με τον τρόπο του ο καθένας, βοήθησαν στην πραγματοποίηση της παρούσας εργασίας.

Ευχαριστώ τον επιβλέποντα καθηγητή μου κ. Θεόδωρο Κατσαούνη για την καθοδήγηση του σε όλη τη διάρκεια της εκπόνησης της μεταπτυχιακής μου εργασίας. Ευχαριστώ επίσης τα υπόλοιπα μέλη της επιτροπής αξιολόγησης μου, κ. Μάρκο Κατσουλάκη και Γεώργιο Ζουράρη, για τις εύστοχες υποδείξεις και παρατηρήσεις τους. Επιπλέον θα ήθελα να ευχαριστήσω τους καθηγητές του τμήματος Εφαρμοσμένων Μαθηματικών του Πανεπιστημίου Κρήτης, για τις γνώσεις που μου παρείχαν στη διάρκεια των μέχρι τώρα σπουδών μου.

Από τους καθηγητές μου αισθάνομαι την ανάγκη να ευχαριστήσω ξεχωριστά :

- Τον Πρόεδρο του Τμήματος Εφαρμοσμένων Μαθηματικών, καθηγητή κ. Γεώργιο Μακράκη, για τις πολύτιμες γνώσεις και τεχνικές που μου δίδαξε, όλα αυτά τα χρόνια, στην περιοχή των Εφαρμοσμένων Μαθηματικών, με τον ξεχωριστό του τρόπο διδασκαλίας και «απομυθοποίησης» των πολύπλοκων και αφηρημένων μαθηματικών εννοιών και δομών. Τον ευχαριστώ επίσης για την πολύτιμη βοήθεια του κατά την αναζήτηση μου για μεταπτυχιακή εργασία : Παρόλο το φόρτο εργασίας του, η πόρτα του γραφείου του ήταν και είναι πάντα ανοικτή!

- Τον καθηγητή του Τμήματος Εφαρμοσμένων Μαθηματικών, κ. Μιχάλη Πλεξουσάκη, για τις πολύτιμες γνώσεις και τεχνικές που μου δίδαξε, όλα αυτά τα χρόνια, πάνω στην ευρύτερη περιοχή των Υπολογιστικών Μαθηματικών και της Επιστήμης Υπολογιστών. Τον ευχαριστώ επίσης για τη γενικότερη βοήθεια που μου παρείχε οποιαδήποτε στιγμή τον είχα χρειαστεί.

- Τον καθηγητή κ. Δημήτριο Μητσούδη, για τις πολύτιμες επισημάνσεις που μου έκανε πάνω στην παρούσα εργασία καθώς και για τις γνώσεις πάνω στην περιοχή των Υπολογιστικών Μαθηματικών, που μου παρείχε στη διάρκεια των σπουδών μου. Τον ευχαριστώ επίσης για τη γενικότερη βοήθεια που μου παρείχε οποιαδήποτε στιγμή τον είχα χρειαστεί.

- Τον ομότιμο καθηγητή του Τμήματος Μαθηματικών του Πανεπιστημίου Κρήτης, κ.Απόστολο Χατζηδήμο, για το εύρος των γνώσεων που μου πρόσφερε, στην περιοχή της Αριθμητικής Γραμμικής Άλγεβρας.

- Τον καθηγητή του Τμήματος Μαθηματικών του Πανεπιστημίου Κρήτης, κ. Γεώργιο Κωστάκη, για τις γνώσεις που μου παρείχε, την ενθάρυνση του και τη βοήθεια του, οποιαδήποτε στιγμή τη χρειαζόμουν.

Ιδιαίτερα οφείλω να ευχαριστήσω τον συνάδελφο και φίλο Γιώργο Δετοράκη, για την επιστημονική ανταλλαγή απόψεων που είχαμε, καθώς και για την πολύτιμη βοήθεια του σε πολλά τεχνικά ζητήματα της εργασίας μου, σε όλη τη διάρκεια της εκπόνησης της. Ευχαριστώ επίσης τους συναδέλφους και φίλους, Γιώργο Τζεδάκη και Κώστα Τζιράκη, για τη βοήθεια τους σε τεχνικά ζητήματα κατά τη συγγραφή της εργασίας μου.

Ευχαριστώ θερμά την οικογένεια μου, για την εμπιστοσύνη που έδειξαν και δείχνουν σε μένα και για τη στήριξη που μου παρείχαν και μου παρέχουν όλα αυτά τα χρόνια.

Ευχαριστώ από καρδιάς, τα αγαπημένα μου ξαδέλφια: Σοφία, Σέβη, Σοφία, Γιώργο, Βάσω, Πόπη. Ευχαριστώ από καρδιάς τους φίλους μου και ιδιαίτερα την Τένια και το Γιώργο, για τη διαρκή τους υποστήριξη και αγάπη: Παρόλο που είστε «μακριά» μου, την απόσταση αυτή

εκμηδενίζουν η διαρκή σας στήριξη, η πίστη σας σε μένα και η διαρκή σας ενθάρυνση και εμψύχωση για να προχοράω μπροστά! Δίχως εσάς ίσως να μην είχα την ψυχική δύναμη να ολοκληρώσω αυτή την εργασία! Σας χρωστάω ένα μεγάλο ευχαριστώ!

Με την ολοκλήρωση της μεταπτυχιακής μου εργασίας, μου δίνεται η ευκαιρία να σημειώσω ότι αποτέλεσε για μένα, ένα ιδιαίτερα δύσκολο και κοπιαστικό έργο. Είναι δύσκολο να αποτυπωθούν με λόγια τα ποικίλα συναισθήματα που ένιωσα, κατά τη διάρκεια της εκπόνησης της εργασίας: τα ατελείωτα ξενύχτια μπροστά στον υπολογιστή, τις απέραντες ώρες υπομονής και τα αμέτρητα χιλιόμετρα επιμονής... Άλλωστε «ό,τι αξίζει, πονάει κι είναι δύσκολο...»: Ο δρόμος για την «Ιθάκη» είναι ανηφορικός, αλλά ας ευχόμαστε να είναι «μακρύς», γιατί το ταξίδι είναι «γεμάτο περιπέτειες, γεμάτο γνώσεις». Τι άλλο μπορεί να μας προσφέρει η «Ιθάκη» ; Μας έδωσε ό,τι είχε να μας δώσει : *Μας έβγαλε στο δρόμο...*

Μάνος Ψυχάρης
Ηράκλειο, Μαϊος 2010.

## Περίληψη

Η επίλυση πολύ μεγάλων και αραιών γραμμικών συστημάτων αλγεβρικών εξισώσεων αποτελεί βασικό συστατικό στοιχείο των επιστημονικών υπολογισμών. Κατά την αριθμητική προσέγγιση των λύσεων μερικών διαφορικών και ολοκληρωτικών εξισώσεων συχνά εμφανίζεται η ανάγκη να επιλύσουμε τέτοια συστήματα.

Στη ρευστοδυναμική, η διακριτοποίηση των εξισώσεων Stokes, με μεθόδους πεπερασμένων διαφορών ή πεπερασμένων στοιχείων, έχει ως αποτέλεσμα σε κάθε βήμα του αλγορίθμου την επίλυση ενός *μεγάλου* και *αραιού* **συμμετρικού** γραμμικού συστήματος αλγεβρικών εξισώσεων. Στην παρούσα εργασία υλοποιώ και συγκρίνω ένα σύνολο από μεθόδους για την αποδοτική αριθμητική επίλυση των παραπάνω γραμμικών συστημάτων.

# Περιεχόμενα

# Κατάλογος Πινάκων

# Κατάλογος Σχημάτων

# Κεφάλαιο 1

# Εισαγωγή

Η μαθηματική μοντελοποίηση φυσικών φαινομένων και διεργασιών συχνά καταλήγει στην επίλυση ενός συστήματος συζευγμένων μη γραμμικών διαφορικών ή και ολοκληρωτικών εξισώσεων. Οι εξισώσεις αυτές σπάνια έχουν γνωστή αναλυτική λύση, οπότε υπάρχει ανάγκη να καταφύγουμε σε αριθμητικές μεθόδους για την προσέγγιση των λύσεων τους.

Για τους παραπάνω λόγους, η αριθμητική προσέγγιση των λύσεων μερικών διαφορικών και ολοκληρωτικών εξισώσεων αποτελεί ένα πολύ σημαντικό στάδιο κατά τη μοντελοποίηση πραγματικών προβλημάτων της σύγχρονης επιστήμης και τεχνολογίας.

Κατά την αριθμητική προσέγγιση των λύσεων αυτών των εξισώσεων συχνά εμφανίζεται η ανάγκη να επιλύσουμε πολύ μεγάλα γραμμικά συστήματα. Φυσικά η έννοια «μεγάλα» είναι σχετική και εξαρτάται άμεσα από το μέγεθος της διαθέσιμης μνήμης του υπολογιστή. Το σύνηθες χαρακτηριστικό αυτών των συστημάτων είναι ότι ο πίνακας τους είναι **αραιός**, δηλαδή «πολλά» από τα στοιχεία του είναι μηδέν.

*Ειδικά στην περίπτωση που μοντελοποιούμε προβλήματα σε 3 χωρικές διαστάσεις, τα γραμμικά συστήματα που προκύπτουν είναι πραγματικά τεράστια* και η **άμεση επίλυση** τους, με μεθόδους που βασίζονται σε διάφορες αραιές παραλλαγές του βασικού αλγορίθμου της απαλοιφής Gauss, είναι απαγορευτικά ακριβή ως προς την απαιτούμενη μνήμη και υπολογιστικό χρόνο.

Σήμερα για την αριθμητική επίλυση των παραπάνω, πολύ μεγάλων γραμμικών συστημάτων, χρησιμοποιούνται κυρίως **επαναληπτικές μέθοδοι**. Οι επαναληπτικές μέθοδοι δεν απαιτούν την αποθήκευση του πίνακα του γραμμικού συστήματος, αλλά μόνο την ικανότητα να εκτελούμε πολλαπλασιασμούς πίνακα – διάνυσμα. Μια σημαντική περιοχή έρευνας, αποτελεί η εύρεση αποδοτικών επαναληπτικών μεθόδων οι οποίες να απαιτούν μικρό αριθμό πολλαπλασιασμών πίνακα – διάνυσμα και μικρής ποσότητας επιπλέον εργασία, για να υπολογίσουν μια καλή προσεγγιστική λύση του γραμμικού συστήματος. Πολλές φορές οι τεχνικές που χρησιμοποιούνται για την επίλυση μεγάλων γραμμικών συστημάτων, είναι εφαρμόσιμες και σε άλλες περιοχές, όπως για παράδειγμα στο, κεντρικής σημασίας πρόβλημα, του υπολογισμού των ιδιοτιμών.

*Αν εξετάσει κανείς πού καταναλώνεται ο μεγαλύτερος υπολογιστικός χρόνος, όταν εκτελούνται υπολογισμοί στην επιστήμη και στην τεχνολογία, συνήθως ανακαλύπτει έναν υπολογιστικό πυρήνα που αντιστοιχεί σε πρόβλημα από την αριθμητική γραμμική άλγεβρα.*

*Η σύγκριση των διαφόρων μεθόδων αριθμητικής επίλυσης γραμμικών συστημάτων ως προς την απόδοση τους, δεν είναι απλό ζήτημα*: Οι ιδιότητες του πίνακα, του υπό μελέτη γραμμικού συστήματος καθώς και το πρόβλημα από το οποίο προέρχεται το σύστημα, πολλές φορές επηρεάζουν σημαντικά την απόδοση των διαφόρων διαθέσιμων αλγορίθμων επίλυσης. *Τα ειδικότερα χαρακτηριστικά του προβλήματος από το οποίο προέρχεται το γραμμικό σύστημα, συχνά μας οδηγούν στην επιλογή της αποδοτικότερης μεθόδου για το συγκεκριμένο πρόβλημα ή ακόμα*

*και στην κατασκευή μιας νέας.*

Ειδικά όταν το γραμμικό σύστημα προέρχεται από τη διακριτοποίηση κάποιου «συνεχούς» προβλήματος (π.χ. σύστημα μερικών διαφορικών εξισώσεων) αποτελεί με κάποια έννοια διακριτό ανάλογό του. Είναι λογικό λοιπόν να «περνάνε» στο γραμμικό σύστημα πολλές από τις ιδιότητες του αντίστοιχου συνεχούς προβλήματος.

Στη ρευστοδυναμική οι εξισώσεις **Navier-Stokes** περιγράφουν τη ροή ενός *Νευτώνιου, ασυμπίεστου* και *ιξώδους* ρευστού. Στη συνέχεια αναφέρουμε τις *χρονοανεξάρτητες (steady-state)* εξισώσεις Navier-Stokes και Stokes:

Ξεκινάμε με τις *χρονοανεξάρτητες, ασυμπίεστες* εξισώσεις Navier-Stokes: Έστω ότι η περιοχή ροής του ρευστού, είναι ένα φραγμένο και συνεκτικό σύνολο $\Omega \subset \mathbb{R}^d\,(d = 2, 3)$ με κατά τμήματα λείο σύνορο $\Gamma = \partial\Omega$. Δεδομένου ενός εξωτερικού πεδίου δυνάμεων $\mathbf{f} : \Omega \longrightarrow \mathbb{R}^d$ και συνοριακών δεδομένων $\mathbf{g} : \Gamma \longrightarrow \mathbb{R}^d$, το πρόβλημα είναι να βρεθεί ένα διανυσματικό *πεδίο ταχυτήτων* $\mathbf{u} : \Omega \longrightarrow \mathbb{R}^d$ και ένα βαθμωτό *πεδίο πίεσης* $p : \Omega \longrightarrow \mathbb{R}$, έτσι ώστε:

$$\begin{aligned} -\nu\Delta\mathbf{u} + \big(\mathbf{u}\cdot\nabla\big)\mathbf{u} + \nabla p &= \mathbf{f} \text{ στο } \Omega, \\ \mathrm{div}\mathbf{u} = \nabla\cdot\mathbf{u} &= 0 \text{ στο } \Omega, \\ \mathbf{u} &= \mathbf{g}, \text{ στο } \Gamma = \partial\Omega, \end{aligned}$$

όπου $\nu > 0$ είναι το *κινηματικό ιξώδες* της ροής (αντιστρόφως ανάλογο με τον *αριθμό Reynolds* $Re$), $\Delta$ είναι ο τελεστής του Laplace στο $\mathbb{R}^d$ και $\nabla$ είναι ο τελεστής βαθμίδας.

Αν θεωρήσουμε ότι *η ροή είναι «χαμηλής ταχύτητας» ώστε τα φαινόμενα μεταφοράς να μπορούν να αγνοηθούν*, οι εξισώσεις Navier-Stokes ανάγονται στις λεγόμενες εξισώσεις Stokes:

$$\begin{aligned} -\nu\Delta\mathbf{u} + \nabla p &= \mathbf{f} \text{ στο } \Omega, \\ \mathrm{div}\mathbf{u} = \nabla\cdot\mathbf{u} &= 0 \text{ στο } \Omega, \\ \mathbf{u} &= \mathbf{g}, \text{ στο } \Gamma = \partial\Omega, \end{aligned}$$

Η διακριτοποίηση των εξισώσεων Stokes με μεθόδους πεπερασμένων διαφορών ή πεπερασμένων στοιχείων, έχει ως αποτέλεσμα την επίλυση ενός *μεγάλου* και *αραιού* **συμμετρικού** γραμμικού συστήματος συγκεκριμένης δομής.

*Αντικείμενο αυτής της παρούσας εργασίας είναι η σύγκριση ενός συνόλου μοντέρνων μεθόδων που είναι σήμερα διαθέσιμες για την αριθμητική επίλυση των μεγάλων και αραιών γραμμικών συστημάτων που προκύπτουν από τη διακριτοποίηση των εξισώσεων Stokes.*

Η παραπάνω μελέτη δεν είναι τόσο περιοριστική όσο αρχικά φαίνεται, καθώς η μοντελοποίηση μιας πληθώρας προβλημάτων καταλήγει στην αριθμητική επίλυση γραμμικών συστημάτων παρόμοιας δομής.

*Η διάρθρωση της εργασίας έχει ως εξής:*

Το **Κεφάλαιο** 2 ξεκινά με μια γενική αναφορά στις εξισώσεις Stokes και περιγράφεται η δομή, καθώς και διάφορες βασικές ιδιότητες των γραμμικών συστημάτων, στα οποία καταλήγει η διακριτοποίησή τους. Γίνεται *σύνδεση με προβλήματα από τη βελτιστοποίηση*, που καταλήγουν σε γραμμικά συστήματα ίδιας δομής: θα δούμε ότι *η λύση του γραμμικού συστήματος, που προκύπτει από τη διακριτοποίηση των εξισώσεων Stokes, αντιστοιχεί στο μοναδικό σαγματικό σημείο ενός προβλήματος τετραγωνικής βελτιστοποίησης (ελαχιστοποίησης) υπό περιορισμούς.*

Στο **Κεφάλαιο** 3 παρουσιάζεται και αναλύεται συνοπτικά ένα σύνολο από επαναληπτικές μεθόδους επίλυσης του γραμμικού συστήματος. Οι επαναληπτικές μέθοδοι που μελετήθηκαν μπορούν να κατηγοριοποιηθούν σε δύο ομάδες:

1. **Μέθοδοι βελτιστοποίησης** (Uzawa, Augmented Lagrangian, Simplified Augmented Lagrangian) (βλέπε [7, 25, 26])

2. **Μέθοδοι υποχώρων Krylov** (Projected CG, Preconditioned MINRES, MINRES) (βλέπε [1, 3, 4, 6, 8] καθώς και το πολύ αξιόλογο βιβλίο της Greenbaum [10])

Επίσης παρουσιάζεται ο **εμπορικός κώδικας MA57** της βιβλιοθήκης HSL, *ένας από τους πιο μοντέρνους κώδικες που είναι διαθέσιμοι αυτή τη στιγμή, για άμεση επίλυση συμμετρικών και εν γένει μη ορισμένων γραμμικών συστημάτων* (βλέπε [18, 19, 21]).

Στο **Κεφάλαιο** 4 παρουσιάζονται **εκτενείς υπολογισμοί** και δοκιμές των μεθόδων με πίνακες διαφόρων μεγεθών, που αντιστοιχούν σε διαφορετικές διακριτοποιήσεις των εξισώσεων Stokes. Μελετάται επίσης *η αποδοτικότητα της κάθε μεθόδου σε συνάρτηση με τον αριθμό Reynolds της ροής*.

Στο **Κεφάλαιο** 5 αναφέρονται *γενικά συμπεράσματα και παρατηρήσεις*, που προέκυψαν από την παραπάνω υπολογιστική μελέτη των διαφόρων αριθμητικών μεθόδων επίλυσης και παρουσιάζονται *προτάσεις για περαιτέρω έρευνα και μελέτη*.

Τέλος στο **Παράρτημα Α΄** παρουσιάζονται τα *αρχεία πηγαίου κώδικα*, στα οποία έχουν υλοποιηθεί όλες οι μέθοδοι που μελετήθηκαν στην παρούσα εργασία και στο **Παράρτημα Β΄** κάποιοι βοηθητικοί κώδικες που χρησιμοποιήθηκαν.

# Κεφάλαιο 2

# Περιγραφή του προβλήματος

## 2.1 Οι εξισώσεις Stokes

Έστω $\Omega \subset \mathbb{R}^d\,(d=2,3)$ ένα φραγμένο και συνεκτικό σύνολο με κατά τμήματα λείο σύνορο $\Gamma = \partial\Omega$. Θεωρούμε τις εξισώσεις Stokes της ρευστοδυναμικής, οι οποίες περιγράφουν τη ροή ενός **Νευτώνιου, ασυμπίεστου, ιξώδους** ρευστού στο $\Omega$:

Δοσμένης μιας διανυσματικής συνάρτησης $\mathbf{f} : \Omega \longrightarrow \mathbb{R}^d$ και κατάλληλων συνοριακών δεδομένων, το ζητούμενο είναι να βρεθεί μια διανυσματική συνάρτηση $\mathbf{u} : \Omega \longrightarrow \mathbb{R}^d$ και μια βαθμωτή συνάρτηση $p : \Omega \longrightarrow \mathbb{R}$, τέτοιες ώστε:

$$
\begin{aligned}
-\nu\Delta\mathbf{u} + \nabla p &= \mathbf{f},\ \text{στο } \Omega && (2.1)\\
\mathrm{div}\mathbf{u} = \nabla\cdot\mathbf{u} &= 0,\ \text{στο } \Omega && (2.2)\\
&+ \ \text{κατάλληλες συνοριακές συνθήκες}
\end{aligned}
$$

,όπου:

- η διανυσματική συνάρτηση $\mathbf{u} : \Omega \longrightarrow \mathbb{R}^d$ αναπαριστά το διανυσματικό **πεδίο ταχυτήτων** του ρευστού.
- η βαθμωτή συνάρτηση $p : \Omega \longrightarrow \mathbb{R}$ αναπαριστά το βαθμωτό **πεδίο πίεσης** του ρευστού.
- η διανυσματική συνάρτηση $\mathbf{f} : \Omega \longrightarrow \mathbb{R}^d$ αναπαριστά το διανυσματικό **πεδίο εξωτερικών δυνάμεων** που ασκούνται στο ρευστό.
- Η σταθερά $\nu \cong \frac{1}{Re} > 0$ αντιπροσωπεύει το **κινηματικό ιξώδες** της ροής, όπου $Re$ ο **αριθμός Reynolds** της ροής.

Το *κινηματικό ιξώδες* της ρόης $\nu \cong \frac{1}{Re} > 0$ ενσωματώνει μέσα του **φαινόμενα τύρβης**, αφού αυτά έχουν πρώτα υπολογιστεί κατά μέσο όρο και έχουν μοντελοποιηθεί με τη βοήθεια του αριθμού Reynolds, $Re > 0$ (βλέπε [27]).

Η εξίσωση (2.1) αναπαριστά το *ισοζύγιο της ορμής* και η εξίσωση (2.2) (γνωστή στη ρευστοδυναμική ως *εξίσωση συνέχειας*), τη *διατήρηση της μάζας*. **Η κρίσιμη υπόθεση της μοντελοποίησης, για την εξαγωγή των εξισώσεων Stokes είναι ότι η ροή είναι χαμηλής ταχύτητας, ώστε τα φαινόμενα μεταφοράς να μπορούν να αγνοηθούν**. Οι εξισώσεις Stokes αποτελούν για τη ρευστοδυναμική, ένα *θεμελιώδες μοντέλο ιξώδους ροής*.

Από τη σκοπιά του *λογισμού μεταβολών (calculus of variations)*, οι εξισώσεις Stokes (κανονικοποιημένες με $\nu = 1$ εδώ) μπορούν να ερμηνευθούν ως οι μερικές διαφορικές εξισώσεις Euler - Lagrange για το ακόλουθο πρόβλημα **βελτιστοποίησης υπό περιορισμούς**:

$$\begin{aligned} \min J(u) &= \frac{1}{2}\int_\Omega \|\nabla \mathbf{u}\|_2^2 \, d\mathbf{x} - \int_\Omega \mathbf{f}\cdot\mathbf{u}\, d\mathbf{x} \\ \text{υπό τον περιορισμό } \mathrm{div}\mathbf{u} &= 0 \end{aligned} \tag{2.3}$$

Από αυτή την οπτική γωνία η συνάρτηση πίεσης $p$ παίζει το ρόλο του πολλαπλασιαστή Lagrange (βλέπε [23]).

## 2.2 Γραμμικά συστήματα σαγματικού σημείου.

Η διακριτοποίηση των εξισώσεων Stokes με μεθόδους πεπερασμένων διαφορών ή πεπερασμένων στοιχείων οδηγεί στην επίλυση μεγάλων γραμμικών συστημάτων, της μορφής:

$$\mathbf{K}\,\mathbf{x} = \mathbf{b}, \tag{2.4}$$

με:

$$\mathbf{K} = \begin{bmatrix} A & B \\ B^T & O \end{bmatrix} \in \mathbb{R}^{(N+q)\times(N+q)}, \;\; \mathbf{x} = \begin{pmatrix} u \\ p \end{pmatrix} \in \mathbb{R}^{N+q}, \;\; \mathbf{b} = \begin{pmatrix} f \\ g \end{pmatrix} \in \mathbb{R}^{N+q},$$

όπου

- $N >> q$,
- $A \in \mathbb{R}^{N\times N}$ είναι συμμετρικός και θετικά ορισμένος πίνακας ($A^T = A$ και $\big(Ax, x\big)_2 > 0, \; \forall x \in \mathbb{R}^N \setminus \{0\}$),
- $B \in \mathbb{R}^{N\times q}$ έχει πλήρες βαθμό ($rank(B) = q$),
- $f \in \mathbb{R}^N, g \in \mathbb{R}^q$, δοσμένα διανύσματα.

Ο πίνακας $A$ αντιπροσωπεύει τη διακριτοποίηση του όρου $-\nu\Delta\mathbf{u}$ και γι' αυτό είναι θετικά ορισμένος. Ο όρος $Bp$, προκύπτει από τη διακριτοποίηση της βαθμίδας της πίεσης και η συνθήκη $B^T u = g$ προκύπτει από τη διακριτοποίηση της διατήρησης της μάζας ($g = 0$ στο πλαίσιο των ασυμπίεστων εξισώσεων Stokes).

Ο πίνακας $B$ συνήθως έχει ελλιπή βαθμό, διότι η πίεση καθορίζεται με απροσδιοριστία μιας σταθεράς. Για να μετασχηματίσουμε τον πίνακα $B$ σε πίνακα με πλήρη βαθμό, πρέπει να προκαθορίσουμε κάποια επιπλέον συνθήκη για την πίεση. Μια από τις συνθήκες που μπορούμε να επιβάλλουμε είναι να καθορίσουμε την τιμή της πίεσης σε κάποιο σημείο του χωρίου ροής $\Omega$.

Γραμμικά συστήματα της μορφής (2.4) εμφανίζονται επίσης *κατά την εύρεση των κρίσιμων σημείων σε προβλήματα τετραγωνικής βελτιστοποίησης υπό (γραμμικούς) περιορισμούς* (βλέπε [8, 23]):

Θεωρούμε το παρακάτω πρόβλημα βελτιστοποίησης με περιορισμούς για το συναρτησιακό $J : \mathbb{R}^N \longrightarrow \mathbb{R}$ :

$$\begin{aligned} \min J(u) &= \frac{1}{2}\big(Au, u\big)_2 - \big(f, u\big)_2, \\ \text{υπό τον περιορισμό } B^T u &= g, \end{aligned} \tag{2.5}$$

όπου:

- $A \in \mathbb{R}^{N\times N}$, δοσμένος συμμετρικός και θετικά ορισμένος πίνακας ($A^T = A$ και $\big(Ax, x\big)_2 > 0,\ \forall x \in \mathbb{R}^N \setminus \{0\}$),
- $B \in \mathbb{R}^{N\times q}$, με $N > q$, δοσμένος πίνακας με πλήρες βαθμό ($rank(B) = q$ ),
- $f \in \mathbb{R}^N, g \in \mathbb{R}^q$, δοσμένα διανύσματα.

Αν $p \in \mathbb{R}^q$ είναι το διάνυσμα των πολλαπλασιαστών Lagrange, η συνάρτηση Lagrange για το παραπάνω πρόβλημα βελτιστοποίησης, είναι:

$$\begin{aligned} L(u,p) &= J(u) + \big(B^T u - g, p\big)_2 \\ &= \frac{1}{2}\big(Au, u\big)_2 - \big(f, u\big)_2 + \big(B^T u - g, p\big)_2 \end{aligned}$$

Για την παραπάνω συνάρτηση Lagrange, έχουμε:

$$\begin{aligned} \frac{\partial L(u,p)}{\partial u} &= Au - f + Bp, \\ \frac{\partial L(u,p)}{\partial p} &= B^T u - g, \end{aligned}$$

Η κλίση της συνάρτησης Lagrange, είναι:

$$\begin{aligned} \nabla L(u,p) &= \Big(\frac{\partial L(u,p)}{\partial u}, \frac{\partial L(u,p)}{\partial p}\Big) \\ &= \Big(Au - f + Bp, B^T u - g\Big). \end{aligned}$$

Τα κρίσιμα σημεία για το πρόβλημα βελτιστοποίησης (2.5), προκύπτουν μηδενίζοντας την κλίση της συνάρτησης Lagrange.

Βλέπουμε λοιπόν ότι, για την εύρεση των κρίσιμων σημείων του προβλήματος τετραγωνικής βελτιστοποίησης (2.5), πρέπει να επιλύσουμε ένα γραμμικό σύστημα της μορφής (2.4).

Όπως θα δούμε αμέσως παρακάτω, *υπό την προυπόθεση ότι ο πίνακας $B$ έχει πλήρη βαθμό* (υπόθεση που έχει γίνει και ισχύει για όλη τη συνέχεια) *ο πίνακας $K$ του γραμμικού συστήματος (2.4) δεν έχει το μηδέν ιδιοτιμή, άρα είναι μη ιδιόμορφος.* Επομένως το πρόβλημα (2.5) έχει ένα μοναδικό κρίσιμο σημείο, που είναι η μοναδική λύση του γραμμικού συστήματος (2.4).

Μπορεί να αποδειχθεί (βλέπε [8, 23]) ότι το μοναδικό αυτό κρίσιμο σημείο του (2.5), είναι **σαγματικό** σημείο της συνάρτησης Lagrange $L : \mathbb{R}^N \times \mathbb{R}^q \longrightarrow \mathbb{R}$, με $L(u,p) = \frac{1}{2}\big(Au, u\big)_2 - \big(f, u\big)_2 + \big(B^T u - g, p\big)_2$.

Για τον παραπάνω λόγο, γραμμικά συστήματα της μορφής (2.4) αναφέρονται στη σχετική βιβλιογραφία συχνά ως **«προβλήματα σαγματικού σημείου» (saddle point problems)** (βλέπε [8, 23]).

Βλέπουμε λοιπόν ότι *η μοναδική λύση του γραμμικού συστήματος (2.4) που προκύπτει από τη διακριτοποίηση των εξισώσεων Stokes αντιστοιχεί στο μοναδικό σαγματικό σημείο ενός προβλήματος τετραγωνικής βελτιστοποίησης (ελαχιστοποίησης) υπό περιορισμούς.*

Στη συνέχεια παραθέτουμε μια βασική πρόταση σχετική με την επιλυσιμότητα του γραμμικού συστήματος (2.4) και με το φάσμα των ιδιοτιμών του συμμετρικού πίνακα $K$ (βλέπε [8]).

**Πρόταση 1.** *Έστω το γραμμικό σύστημα (2.4). Τότε ισχύουν τα ακόλουθα αποτελέσματα:*

1. *Ο $K \in \mathbb{R}^{(N+q)\times(N+q)}$ είναι μη ιδιόμορφος, αν και μόνο αν, ο $S = -B^T A^{-1} B \in \mathbb{R}^{q\times q}$ είναι μη ιδιόμορφος.*
2. *Ο $K \in \mathbb{R}^{(N+q)\times(N+q)}$ είναι συμμετρικός και μη ορισμένος (indefinite) με $N$ θετικές και $q$ αρνητικές ιδιοτιμές.*

*Απόδειξη.* Εύκολα μπορούμε να επαληθεύσουμε ότι ο πίνακας $K$ επιδέχεται την ακόλουθη block τριγωνική παραγοντοποίηση:

$$\mathbf{K} = \begin{bmatrix} A & B \\ B^T & O \end{bmatrix} = \begin{bmatrix} I & O \\ B^T A^{-1} & I \end{bmatrix} \begin{bmatrix} A & O \\ O & S \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ O & I \end{bmatrix} = PDP^T$$

,όπου:

$$P = \begin{bmatrix} I & O \\ B^T A^{-1} & I \end{bmatrix}, D = \begin{bmatrix} A & O \\ O & S \end{bmatrix}, \text{και } S = -B^T A^{-1} B$$

Για το (1) έχουμε:

$\det K \neq 0 \Leftrightarrow \det(PDP^T) \neq 0 \Leftrightarrow \det P \det D \det P^T \neq 0 \overset{\det P = \det P^T = 1}{\Longleftrightarrow} \det D \neq 0 \Leftrightarrow \det A \det S \neq 0 \overset{\det A \neq 0}{\Longleftrightarrow} \det S \neq 0.$

Για το (2) έχουμε:

Οι συμμετρικοί πίνακες $K$ και $D$ είναι congruent, επομένως από την **αρχή της αδράνειας του Sylvester** (βλέπε [8, 16])[1] έχουν τον ίδιο αριθμό θετικών, αρνητικών και μηδενικών ιδιοτιμών. Άρα ο $K$ έχει $N$ θετικές και $q$ αρνητικές ιδιοτιμές, αφού ο $A$ είναι συμμετρικός και θετικά ορισμένος (από κατασκευής) και ο $S$ είναι συμμετρικός και αρνητικά ορισμένος, όπως μπορούμε να δούμε εύκολα: Θεωρούμε ένα $x \in \mathbb{R}^q \setminus \{0\}$. Τότε έχουμε: $\left(Sx, x\right)_2 = -\left(B^T A^{-1} B x, x\right)_2 = -\left(A^{-1} B x, Bx\right)_2 = -\left(A^{-1} y, y\right)_2, y = Bx$. Όμως: $-\left(A^{-1} y, y\right)_2 < 0$, $\forall y \in \mathbb{R}^N \setminus \{0\}$, αφού $A$ θετικά ορισμένος (άρα και $A^{-1}$ θετικά ορισμένος). Ο πίνακας $B$ έχει πλήρες βαθμό, επομένως η απεικόνιση $x \to Bx$ είναι $1-1$, άρα $\left(Sx, x\right)_2 < 0$, $\forall x \in \mathbb{R}^q \setminus \{0\}$, δηλαδή ο πίνακας $S$ είναι συμμετρικός και αρνητικά ορισμένος. □

## 2.3 Η μη ορισιμότητα του γραμμικού συστήματος.

Όπως είδαμε,ο πίνακας $K$ του γραμμικού συστήματος (2.4) είναι συμμετρικός και μη ορισμένος, με $N$ θετικές και $q$ αρνητικές ιδιοτιμές.

Η μη ορισιμότητα του γραμμικού συστήματος (2.4) είναι θεμελιώδης: Όσο το υπολογιστικό πλέγμα πυκνώνει, το μέγεθος του γραμμικού συστήματος (2.4) μεγαλώνει και τότε, σύμφωνα με τα παραπάνω, τόσο το πλήθος των θετικών ιδιοτιμών, όσο και το πλήθος των αρνητικών ιδιοτιμών του πίνακα $K$ μεγαλώνει. Γι΄ αυτόν ακριβώς το λόγο, πίνακες με αυτά τα χαρακτηριστικά συχνά αναφέρονται ως **ισχυρά μη ορισμένοι πίνακες (highly or strongly indefinite matrices)** (βλέπε [8, 16]).

Στις εφαρμογές ο πίνακας $K$ είναι πολύ μεγάλος και αραιός, οπότε τα παραπάνω γραμμικά συστήματα συνήθως λύνονται με χρήση επαναληπτικών μεθόδων και κυρίως με μεθόδους που βασίζονται σε υπόχωρους Krylov. Δυστυχώς, εξαιτίας των φασματικών ιδιοτήτων του πίνακα

[1] Δυο συμμετρικοί πίνακες $R$ και $S$ λέγονται congruent αν $R = XSX^T$ , για κάποιον μη ιδιόμορφο πίνακα X. Η αρχή της αδράνειας του Sylvester λέει ότι οι congruent πίνακες έχουν τον ίδιο αριθμό θετικών, αρνητικών και μηδενικών ιδιοτιμών.

$K$, που αναφέρθηκαν παραπάνω, οι διάφορες μέθοδοι Krylov συνήθως τείνουν να συγκλίνουν αργά. Γι΄ αυτό το λόγο η ανάγκη για καλούς προρυθμιστές είναι μεγάλη και η χρήση τους σε μεγάλα προβλήματα επιβεβλημένη (βλέπε [8]).

Στο επόμενο Κεφάλαιο θα αναφερθώ σε ένα σύνολο από, επαναληπτικές κυρίως, μεθόδους για τη λύση του (2.4).

# Κεφάλαιο 3

# Μέθοδοι επίλυσης του γραμμικού συστήματος

## 3.1 Γενικά για τις μεθόδους

Για την επίλυση του γραμμικού συστήματος (2.4) χρησιμοποιήθηκαν οι ακόλουθες 6 **επαναληπτικές μέθοδοι**:

1. Projected CG
2. Uzawa
3. Augmented Langrangian
4. Simplified Augmented Langrangian
5. Preconditioned MINRES
6. MINRES

Επίσης χρησιμοποίηθηκε μια **άμεση μέθοδος επίλυσης** και πιο συγκεκριμένα μια παραλλαγή της απαλοιφής Gauss για αραιούς πίνακες, από τη βιβλιοθήκη HSL (βλέπε [21]).

Στη συνέχεια αναφέρω μια σύντομη περιγραφή της κάθε μεθόδου καθώς και των βασικών ιδιοτήτων της.

## 3.2 Η μέθοδος Projected CG

### 3.2.1 Εξαγωγή της μεθόδου Projected CG

Η μέθοδος Projected CG προτάθηκε από τον Randall Bramley (βλέπε [6, 7, 25]). Η βασική ιδέα της μεθόδου συνίσταται στην απαλοιφή της πίεσης, με στόχο τη μετατροπή του γραμμικού συστήματος (2.4) σε ένα ισοδύναμο σύστημα με πίνακα συμμετρικό και θετικά ορισμένο. Στο τελευταίο γραμμικό σύστημα μπορεί να εφαρμοστεί η μέθοδος των συζηγών κλίσεων (βλέπε [1]).

Αναλυτικότερα έχουμε: Το γραμμικό σύστημα (2.4) μπορεί να γραφεί ισοδύναμα σε block μορφή:

$$Au + Bp = f \tag{3.1}$$
$$B^T u = g \tag{3.2}$$

Λύνοντας την (3.1) ως προς $p$, με την έννοια των ελαχίστων τετραγώνων, παίρνουμε: $p = B^{\dagger}(f - Au)$,όπου $B^{\dagger}$ είναι ο ψευδοαντίστροφος κατά Moore-Penrose του $B$ (βλέπε [6]). Αντικαθιστώντας την έκφραση αυτή για το $p$, πίσω στην (3.1) παίρνουμε:

$$\begin{aligned} Au + Bp &= f \Longleftrightarrow \\ Au + BB^{\dagger}\big(f - Au\big) &= f \Longleftrightarrow \\ \big(I - BB^{\dagger}\big)Au &= f - BB^{\dagger}f \Longleftrightarrow \\ \big(I - BB^{\dagger}\big)Au &= \big(I - BB^{\dagger}\big)f. \end{aligned} \tag{3.3}$$

Όμως ο τελεστής $P = I - BB^{\dagger}$, είναι ο τελεστής ορθογώνιας προβολής πάνω στον $null(B^T) = \{x \in \mathbb{R}^N : B^T x = 0\}$. Έτσι η (3.3) γράφεται ισοδύναμα:

$$PAu = Pf. \tag{3.4}$$

Επιπλέον $B^T u = 0 \Leftrightarrow u = Pu$, οπότε αντικαθιστώντας στην (3.4), παίρνουμε:

$$PAPu = Pf. \tag{3.5}$$

Στην περίπτωση που ο πίνακας $B$ έχει πλήρη βαθμό, ο ψευδοαντίστροφος κατά Moore-Penrose του $B$ έχει την αναπαράσταση:

$$B^{\dagger} = \big(B^T B\big)^{-1} B^T. \tag{3.6}$$

Έστω $M = PAP \in \mathbb{R}^{N \times N}$. Τότε έχουμε:

$$\begin{aligned} P^T &= \big(I - BB^{\dagger}\big)^T \\ &= \Big(I - B\big(B^T B\big)^{-1} B^T\Big)^T \\ &= I - \big(B^T\big)^T \Big[\big(B^T B\big)^{-1}\Big]^T B^T \\ &= I - B\Big[\big(B^T B\big)^T\Big]^{-1} B^T \\ &= I - B\big(B^T B\big)^{-1} B^T \\ &= I - BB^{\dagger} \\ &= P \end{aligned}$$

Επομένως $M^T = \big(PAP\big)^T = P^T A^T P^T = PAP = M$, αφού $A$ συμμετρικός και $P$ συμμετρικός, όπως φάνηκε παραπάνω. Επομένως ο πίνακας $M$ είναι συμμετρικός. Έστω ένα $x \in \mathbb{R}^N \setminus \{0\}$. Τότε έχουμε:

$$\begin{aligned} \big(Mx, x\big)_2 &= \big(PAPx, x\big)_2 \\ &= \big(APx, Px\big)_2. \end{aligned} \tag{3.7}$$

Αν θέσουμε $y = Px \in \mathbb{R}^N$ η (3.7) γίνεται: $\big(Mx, x\big)_2 = \big(Ay, y\big)_2 > 0$, $\forall y \in \mathbb{R}^N \setminus \{0\}$, αφού ο $A$ είναι συμμετρικός και θετικά ορισμένος. Συμπεραίνουμε λοιπόν ότι ο πίνακας $M = PAP$ είναι συμμετρικός και θετικά ορισμένος.

Συνοψίζοντας, είδαμε ότι η επίλυση του γραμμικού συστήματος (2.4) είναι ισοδύναμη με την επίλυση του γραμμικού συστήματος:

$$\mathbf{PAPu} = \mathbf{Pf},$$

το οποίο έχει πίνακα συντελεστών **συμμετρικό και θετικά ορισμένο**.

Επίσης μπορεί να αποδειχθεί (βλέπε [6]) ότι οι μικρότερες και οι μεγαλύτερες ιδιάζουσες τιμές των πινάκων $A$ και $PAP$ ικανοποιούν:

$$\begin{aligned} \sigma_{min}(A) &\leq \sigma_{min}(PAP), \\ \sigma_{max}(A) &\geq \sigma_{max}(PAP). \end{aligned}$$

Άρα:

$$\kappa_2(PAP) = \frac{\sigma_{max}(PAP)}{\sigma_{min}(PAP)} \leq \frac{\sigma_{max}(A)}{\sigma_{min}(A)} = \kappa_2(A).$$

Δηλαδή, ο **φασματικός δείκτης κατάστασης του** $PAP$ **δεν είναι χειρότερος από τον φασματικό δείκτη κατάστασης του** $A$. Επομένως δεν είναι κακή ιδέα, αντί για το γραμμικό σύστημα (2.4) να λύσουμε το ισοδύναμο γραμμικό σύστημα:

$$\mathbf{PAPu} = \mathbf{Pf}.$$

και στη συνέχεια να ανακτήσουμε την πίεση από τη σχέση:

$$\mathbf{p} = \mathbf{B}^{\dagger}(\mathbf{f} - \mathbf{Au}).$$

### 3.2.2 Αλγόριθμος της μεθόδου Projected CG

Είδαμε ότι η επίλυση του γραμμικού συστήματος (2.4) είναι ισοδύναμη με την επίλυση του γραμμικού συστήματος:

$$\mathbf{PAPu} = \mathbf{Pf},$$

το οποίο έχει πίνακα συντελεστών **συμμετρικό και θετικά ορισμένο** και με **φασματικό δείκτη κατάστασης όχι χειρότερο από το φασματικό δείκτη κατάστασης του** $A$.

Επομένως, μπορούμε να εφαρμόσουμε τον αλγόριθμο των Συζυγών Κλίσεων (Conjugate Gradients - CG) στο γραμμικό σύστημα $PAPu = Pf$ (βλέπε [1, 6, 7, 25]). Επίσης με την κατάλληλη αρχικοποίηση που θα κάνουμε, θα χρειστεί να υπολογίζουμε μόνο μια προβολή ανά επανάληψη της $CG$ μεθόδου.

Στη συνέχεια δίνουμε τον ψευδοαλγόριθμο της μεθόδου:

1) Αρχικοποίηση:

$$\begin{aligned} \text{παραγοντοποίηση Cholesky: } B^T B &= R^T R \\ \text{Λύσε } B^T B z &= g \\ \text{Ανάθεσε } u_0 &= Bz \\ \text{Ανάθεσε } d_0 = r_0 &= P\big(f - Au_0\big) \\ \rho_0 &= \big(r_0, r_0\big)_2 \\ k &= 0 \end{aligned}$$

2) Επαναληπτικά:

$$\begin{aligned} w_k &= PAd_k \\ \alpha_k &= \frac{\rho_k}{\big(d_k, w_k\big)_2} \\ u_{k+1} &= u_k + \alpha_k d_k \\ r_{k+1} &= r_k - \alpha_k w_k \\ \rho_{k+1} &= \big(r_{k+1}, r_{k+1}\big)_2 \\ \text{if } \rho_{k+1} &< TOL\,\|Pf\|_2 \text{ stop} \\ \beta_{k+1} &= \frac{\rho_{k+1}}{\rho_k} \\ p_{k+1} &= p_k - \alpha_k B^\dagger A d_k \\ d_{k+1} &= r_{k+1} + \beta_{k+1} d_k \end{aligned}$$

### 3.2.3 Παρατηρήσεις για τον αλγόριθμο της μεθόδου Projected CG

- Ο πίνακας $B^TB$ είναι συμμετρικός και θετικά ορισμένος, αφού ο $B$ έχει πλήρη βαθμό. Για τον υπολογισμό της προβολής $w = Pv$, δοσμένου ενός διανύσματος $v \in \mathbb{R}^N$ σχηματίζουμε το πάνω τρίγωνο του πίνακα $B^TB$ (αφού είναι συμμετρικός) και κάνουμε ανάλυση Cholesky:

  $$B^TB = R^TR,$$

  όπου $R \in \mathbb{R}^{q\times q}$ άνω τριγωνικός πίνακας, με θετικά διαγώνια στοιχεία. Στη συνέχεια ο υπολογισμός της προβολής δίνει:

  $$w = Pv = v - BR^{-1}R^{-T}B^Tv,$$

  και απαιτεί την επίλυση δυο τριγωνικών γραμμικών συστημάτων τάξης $q$, ένα με τον πίνακα $R$ και ένα με τον $R^T$.

- Όπως είδαμε παραπάνω, αν λύσουμε την εξίσωση (3.1) ως προς $p$ με την έννοια των ελαχίστων τετραγώνων, παίρνουμε:

  $$p = B^\dagger\big(f - Au\big).$$

  Η παραπάνω σχέση προτείνει να ανανεώσουμε την πίεση στον αλγόριθμο της $CG$ με τον προφανή τρόπο:

  $$\begin{aligned} p_{k+1} &= B^\dagger\big(f - Au_{k+1}\big) \\ &= B^\dagger\Big(f - A\big(u_k + \alpha_k d_k\big)\Big) \\ &= p_k - \alpha_k B^\dagger A d_k \end{aligned}$$

  Το διάνυσμα $B^\dagger Ad_k$ έχει υπολογιστεί ως ενδιάμεσο βήμα κατά τον υπολογισμό του $w_k$.

## 3.3 Η μέθοδος Uzawa

Η block μορφή (3.1)-(3.2) του γραμμικού συστήματος (2.4) προτείνει να δούμε το γραμμικό σύστημα σαν ένα **πρόβλημα βελτιστοποίησης με περιορισμούς (constrained optimization problem)** (βλέπε [7]): Λύνουμε ως προς $u$, υπό τον περιορισμό το $u$ να βρίσκεται στο σύνολο: $\{u \in \mathbb{R}^N : B^T u = g\}$. Με αυτή τη λογική οι $q$ τελευταίες εξισώσεις: $B^T u = g$ αποτελούν τους περιορισμούς και το διάνυσμα $p \in \mathbb{R}^q$, είναι το διάνυσμα των πολλαπλασιαστών Lagrange. Αν λύσουμε το (3.1) ως προς $u$ και στη συνέχεια αντικαταστήσουμε στο (3.2) έχουμε:

$$u = A^{-1}(f - Bp),$$

$$\begin{aligned} B^T u = g &\Leftrightarrow B^T A^{-1}\big(f - Bp\big) = g \\ &\Leftrightarrow A_0 p = f_0, \end{aligned} \tag{3.8}$$

όπου:

$$\begin{aligned} A_0 &= B^T A^{-1} B, \\ f_0 &= B^T A^{-1} f - g. \end{aligned}$$

Το πρόβλημα τώρα μπορεί να λυθεί σε δύο βήματα: Πρώτα λύνουμε το (3.8), και στη συνέχεια υπολογίζουμε το $u$ λύνοντας το $Au = f - Bp$. Ο υπολογισμός του $A^{-1}$ στην έκφραση για το $A_0$, μπορεί να παρακαμφθεί χρησιμοποιώντας μια επαναληπτική μέθοδο για την επίλυση του (3.8). Πιο συγκεκριμένα ο αλγόριθμος **Uzawa** είναι ο παρακάτω (βλέπε [7, 25]):

**Uzawa Algorithm**

$$\begin{aligned} \text{Ανάθεσε } p &= p_0, \text{ όπου } p_0 \text{ είναι η αρχική πίεση.} \\ \text{Ανάθεσε } R_p &= B^T u_0 - g, \text{ όπου } u_0 \text{ είναι η αρχική ταχύτητα.} \end{aligned}$$

$$\begin{aligned} \textbf{while } (\|R_p\|_2 &> TOL) \\ \text{Λύσε } Au &= f - Bp \\ \text{Υπολόγισε } R_p &= B^T u - g \\ \text{Λύσε } Aw &= BR_p \\ \text{Υπολόγισε } t &= \frac{\big(R_p, R_p\big)_2}{\big(R_p, B^T w\big)_2} \\ \text{Ανάθεσε } p &= p + tR_p \\ \textbf{endwhile} & \\ \text{Λύσε } Au &= f - Bp \end{aligned}$$

## 3.4 Η μέθοδος Augmented Lagrangian

Ο ρυθμός σύγκλισης του αλγορίθμου Uzawa που είδαμε παραπάνω μπορεί να βελτιωθεί σημαντικά αν τον εφαρμόσουμε στο «επαυξημένο» σύστημα (βλέπε [26]):

$$\begin{bmatrix} A+rBB^T & B \\ B^T & O \end{bmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f+rBg \\ g \end{pmatrix}, \tag{3.9}$$

το οποίο είναι ισοδύναμο με το (2.4), δηλαδή το παραπάνω σύστημα έχει ακριβώς την ίδια λύση με το (2.4).

Σημείωση: Σε αναλογία με το γραμμικό σύστημα (2.4) μπορεί κανείς εύκολα να δεί ότι το παραπάνω «επαυξημένο» σύστημα αντιστοιχεί στο μηδενισμό της κλίσης της συνάρτησης Lagrange, για την εύρεση των κρίσιμων σημείων του παρακάτω «επαυξημένου» προβλήματος βελτιστοποίησης υπό (γραμμικούς) περιορισμούς, για το συναρτησιακό $\tilde{J} : \mathbb{R}^N \longrightarrow \mathbb{R}$ :

$$\begin{aligned} \min \tilde{J}(u) &= \frac{1}{2}\big((A+rBB^T)u,u\big)_2 - \big(f+rBg,u\big)_2, \\ \text{υπό τον περιορισμό } B^Tu &= g, \end{aligned} \tag{3.10}$$

όπου:

- $A \in \mathbb{R}^{N\times N}$, δοσμένος συμμετρικός και θετικά ορισμένος πίνακας ($A^T = A$ και $\big(Ax,x\big)_2 > 0,\ \forall x \in \mathbb{R}^N \setminus \{0\}$),
- $B \in \mathbb{R}^{N\times q}$, με $N > q$, δοσμένος πίνακας με πλήρη βαθμό ($rank(B) = q$ ),
- $f \in \mathbb{R}^N, g \in \mathbb{R}^q$, δοσμένα διανύσματα,
- $r \geq 0$, δοσμένη μη αρνητική παράμετρος.

Σημείωση: Για $r = 0$, το παραπάνω πρόβλημα βελτιστοποίησης ανάγεται στο πρόβλημα (2.5).

Αν $p \in \mathbb{R}^q$ είναι το διάνυσμα των πολλαπλασιαστών Lagrange, μπορούμε να δούμε, εκτελώντας τις πράξεις, ότι η συνάρτηση Lagrange για το παραπάνω πρόβλημα βελτιστοποίησης, είναι:

$$\begin{aligned} \tilde{L}(u,p) &= \tilde{J}(u) + \big(B^Tu-g,p\big)_2 \\ &= \frac{1}{2}\big(Au,u\big)_2 - \big(f,u\big)_2 + \frac{r}{2}\big(BB^Tu,u\big)_2 - r\big(Bg,u\big)_2 + \big(B^Tu-g,p\big)_2. \end{aligned}$$

Για την παραπάνω συνάρτηση Lagrange, έχουμε:

$$\begin{aligned} \frac{\partial \tilde{L}(u,p)}{\partial u} &= Au - f + rBB^Tu - rBg + Bp, \\ \frac{\partial \tilde{L}(u,p)}{\partial p} &= B^Tu - g. \end{aligned}$$

Η κλίση της συνάρτησης Lagrange, είναι:

$$\begin{aligned} \nabla \tilde{L}(u,p) &= \Big(\frac{\partial \tilde{L}(u,p)}{\partial u}, \frac{\partial \tilde{L}(u,p)}{\partial p}\Big) \\ &= \Big(Au - f + rBB^Tu - rBg + Bp, B^Tu - g\Big). \end{aligned}$$

Βλέπουμε λοιπόν ότι, για την εύρεση των κρίσιμων σημείων του επαυξημένου προβλήματος βελτιστοποίησης (3.10), πρέπει να επιλύσουμε το γραμμικό σύστημα (3.9).

Εφόσον τα γραμμικά συστήματα (3.9) και (2.4) είναι ισοδύναμα, δηλαδή έχουν ακριβώς την ίδια μοναδική λύση και εφόσον, όπως είδαμε στο προηγούμενο κεφάλαιο, το γραμμικό σύστημα (2.4) αντιστοιχεί στο μηδενισμό της συνάρτησης Lagrange για το πρόβλημα βελτιστοποίησης (2.5), συμπεραίνουμε ότι τα προβλήματα βελτιστοποίησης (3.10) και (2.5) έχουν ακριβώς το ίδιο, μοναδικό, κρίσιμο σημείο, το οποίο μάλιστα είναι σαγματικό.

Στην παράμετρο $r > 0$ συνήθως δίνεται μια μεγάλη τιμή για να βελτιώσουμε την κατάσταση του πίνακα $A$. Αυτό οδηγεί στον παρακάτω αλγόριθμο **Augmented Lagrangian** (βλέπε [7],[25]):

**Augmented Lagrangian Algorithm**

$$
\begin{aligned}
\text{Ανάθεσε } p &= p_0, \text{ όπου } p_0 \text{ είναι η αρχική πίεση.} \\
\text{Ανάθεσε } R_p &= B^T u_0 - g, \text{ όπου } u_0 \text{ είναι η αρχική ταχύτητα.} \\
\text{Ανάθεσε } A_r &= A + rBB^T \text{ και } f_r = f + rBg
\end{aligned}
$$

$$
\begin{aligned}
\textbf{while } (\|R_p\|_2 &> TOL) \\
\text{Λύσε } A_r u &= f_r - Bp \\
\text{Υπολόγισε } R_p &= B^T u - g \\
\text{Λύσε } A_r w &= BR_p \\
\text{Υπολόγισε } t &= \frac{\left(R_p, R_p\right)_2}{\left(R_p, B^T w\right)_2} \\
\text{Ανάθεσε } p &= p + tR_p \\
&\textbf{endwhile} \\
\text{Λύσε } A_r u &= f_r - Bp
\end{aligned}
$$

## 3.5 Η μέθοδος Simplified Augmented Lagrangian

Για μεγάλες τιμές του $r$[4], ο αλγόριθμος της Augmented Lagrangian μπορεί να απλοποιηθεί περισσότερο. Ο απλοποιημένος αλγόριθμος που προκύπτει είναι γνωστός ως μέθοδος **Simplified Augmented Lagrangian** (βλέπε [7, 25]):

Ο αλγόριθμος της Simplified Augmented Lagrangian είναι:

**Simplified Augmented Lagrangian Algorithm**

$$\begin{aligned}
\text{Ανάθεσε } p &= p_0, \text{ όπου } p_0 \text{ είναι η αρχική πίεση.} \\
\text{Ανάθεσε } R_p &= B^T u_0 - g, \text{ όπου } u_0 \text{ είναι η αρχική ταχύτητα.} \\
\text{Ανάθεσε } A_r &= A + rBB^T \text{ και } f_r = f + rBg
\end{aligned}$$

$$\begin{aligned}
\textbf{while } (\|R_p\|_2 &> TOL) \\
\text{Λύσε } A_r u &= f_r - Bp \\
\text{Υπολόγισε } R_p &= B^T u - g \\
\text{Ανάθεσε } p &= p + rR_p \\
\textbf{endwhile} & \\
\text{Λύσε } A_r u &= f_r - Bp
\end{aligned}$$

## 3.6 Η μέθοδος Ελάχιστου υπολοίπου(MINRES)

Η **μέθοδος Ελάχιστου υπολοίπου (MINRES)** θεωρείται μια από τις πιο βασικές και standard επαναληπτικές μεθόδους για **συμμετρικά και μη ορισμένα** γραμμικά συστήματα. Προτάθηκε το 1975 από τους Paige και Saunders, μαζί με την SYMMLQ (βλέπε [3]). Βασικό χαρακτηριστικό της μεθόδου είναι ότι απαιτεί μόνο ένα πολλαπλασιασμό πίνακα-διάνυσμα ανά επανάληψη καθώς και ένα μικρό αριθμό πράξεων μεταξύ διανυσμάτων (εσωτερικά γινόμενα και ανανεώσεις διανυσμάτων). Ειδικά στην περίπτωση που έχουμε αραιό πίνακα, το γινόμενο πίνακα-διάνυσμα μπορεί να υπολογισθεί με αρκετά αποδοτικό τρόπο.

Το βασικό ζήτημα λοιπόν, από το οποίο εξαρτάται άμεσα το συνολικό υπολογιστικό κόστος της μεθόδου, για την αριθμητική επίλυση ενός γραμμικού συστήματος, είναι ο αριθμός των επαναλήψεων που απαιτούνται για σύγκλιση στη λύση, με κάποια δεδομένη ικανοποιητική ακρίβεια.

Στη συνέχεια θα προσπαθήσω να δώσω τη βασική ιδέα της MINRES, παραλείποντας τεχνικές λεπτομέρειες: Ο γραμμικός χώρος $\boldsymbol{\mathcal{K}}_k(K, r_0)$ που ορίζεται ως:

$$\boldsymbol{\mathcal{K}}_k(K, r_0) = span(r_0, Kr_0, K^2 r_0, \ldots, K^{k-1} r_0),$$

είναι γνωστός ως **υπόχωρος Krylov** που παράγεται από τον πίνακα $K$ και το αρχικό διάνυσμα υπολοίπου $r_0$.

Για το συμμετρικό και μη ορισμένο γραμμικό σύστημα (2.4) η MINRES υπολογίζει μια ακολουθία διαδοχικών προσεγγίσεων $\{x_k\}_{k=0}^{+\infty} \in x_0 + \boldsymbol{\mathcal{K}}_k(K, r_0)$ της λύσης $x$ για τις οποίες το αντίστοιχο υπόλοιπο $r_k = b - Kx_k$, **ελαχιστοποιεί** την ευκλείδια νόρμα $\|b - K\widetilde{x_k}\|_2$ για όλα τα $\widetilde{x_k}$ πάνω στον αφηνικό υπόχωρο: $x_0 + \boldsymbol{\mathcal{K}}_k(K, r_0)$:

$$\{x_k\}_{k=0}^{+\infty} \in x_0 + \boldsymbol{\mathcal{K}}_k(K, r_0) : \ \|r_k\|_2 = \|b - Kx_k\|_2 = \min_{\widetilde{x_k} \in \ x_0 + \boldsymbol{\mathcal{K}}_k(K, r_0)} \|b - K\widetilde{x_k}\|_2, \qquad (3.11)$$

όπου $x_0$ είναι μια αρχική προσέγγιση της λύσης και $r_0$ το αντίστοιχο υπόλοιπο. Τα αντίστοιχα υπόλοιπα $r_k$ ανήκουν στον αφηνικό υπόχωρο: $r_0 + \boldsymbol{\mathcal{K}}_{k+1}(K, r_0)$.

Από τον ορισμό του υποχώρου Krylov έχουμε ότι:

$$\boldsymbol{\mathcal{K}}_k(K, r_0) \subseteq \boldsymbol{\mathcal{K}}_{k+1}(K, r_0) \tag{3.12}$$

Λόγω του εγκλεισμού (3.12), έχουμε ότι:

$$\|r_{k+1}\|_2 = \|b - Kx_{k+1}\|_2 = \min_{\widetilde{x_{k+1}} \in\, x_0 + \boldsymbol{\mathcal{K}}_{k+1}(K, r_0)} \|b - K\widetilde{x_{k+1}}\|_2 \leq \min_{\widetilde{x_k} \in\, x_0 + \boldsymbol{\mathcal{K}}_k(K, r_0)} \|b - K\widetilde{x_k}\|_2 = \|r_k\|_2, \tag{3.13}$$

αφού παίρνουμε ελάχιστο σε κάποιο μεγαλύτερο σύνολο.

Η MINRES λοιπόν **εγγυάται τη μονοτονική μείωση της ευκλείδιας νόρμας του υπολοίπου πάνω στους εγκιβωτισμένους υποχώρους Krylov, στους οποίους ανήκουν οι διαδοχικές προσεγγίσεις της λύσης που παράγει.**

Ο παραπάνω είναι ένας από τους βασικούς λόγους που πολλές φορές κάνει τη MINRES μέθοδο επιλογής για συμμετρικά και μη ορισμένα γραμμικά συστήματα. Παρόλα αυτά η παρεμφερής μέθοδος SYMMLQ, η οποία έχει κάποια αντίστοιχη, τύπου Petrov - Galerkin, ιδιότητα, προτιμάται πολλές φορές για λόγους ευστάθειας, ειδικά στις περιπτώσεις που απαιτείται μεγάλος αριθμός επαναλήψεων (βλέπε [22]).

Επιπλέον μπορούμε σύντομα να περιγράψουμε πόσο «καλές» είναι οι διαδοχικές προσεγγίσεις τις λύσης, $\{x_k\}_{k=0}^{+\infty} \in x_0 + \boldsymbol{\mathcal{K}}_k(K, r_0)$ που παράγει η MINRES, για το χειρότερο δυνατό διάνυσμα δεξιού μέλους $b$. Για περισσότερες λεπτομέρειες, βλέπε το βιβλίο της Greenbaum ([10]). Αφού $r_k \in r_0 + \boldsymbol{\mathcal{K}}_{k+1}(K, r_0)$ έχουμε ότι το $r_k$ μπορεί να γραφεί: $r_k = P_k(K)r_0$, όπου $P_k$ είναι ένα συγκεκριμένο πολυώνυμο $k$ βαθμού, με $P_k(0) = 1$. Για κάθε άλλο τέτοιο πολυώνυμο $q_k$ έχουμε:

$$\|r_k\|_2 \leq \|q_k(K)r_0\|_2 \leq \|q_k(K)\|_2 \|r_0\|_2. \tag{3.14}$$

Ο πίνακας $K$ είναι συμμετρικός, άρα διαγωνοποιείται μέσω του ορθογώνιου πίνακα $Q$ που έχει στήλες του τα ιδιοδιανύσματα του $K$:

$$K = Q\Lambda Q^T,$$

όπου $\Lambda = diag(\lambda_1, \ldots, \lambda_{N+q})$, είναι ένας διαγώνιος πίνακας με τις ιδιοτιμές του $K$. Έχουμε λοιπόν:

$$\|q_k(K)\|_2 = \|Qq_k(\Lambda)Q^T\|_2 = \|q_k(\Lambda)\|_2,$$

αφού η Ευκλείδια νόρμα παραμένει αναλλοίωτη κάτω από ορθογώνιους μετασχηματισμούς. Η (3.14) τώρα γίνεται:

$$\|r_k\|_2 \leq \|q_k(\Lambda)\|_2 \|r_0\|_2. \tag{3.15}$$

Για τη φασματική νόρμα $\|q_k(\Lambda)\|_2$ έχουμε από τον ορισμό της:

$$\|q_k(\Lambda)\|_2 = \max_{\lambda \in \sigma(K)} |q_k(\lambda)|.$$

Αφού η (3.15) ισχύει για κάθε πολυώνυμο $q_k$, $k$ βαθμού, με $q_k(0) = 1$, μπορούμε στην (3.15) να πάρουμε ελάχιστο πάνω σε όλα τα πολυώνυμα βαθμού $k$, που παίρνουν τιμή 1 στο 0:

$$\|r_k\|_2 \leq \min_{q_k,\, q_k(0)=1} \max_{\lambda \in \sigma(K)} |q_k(\lambda)|\ \|r_0\|_2, \tag{3.16}$$

όπου $\sigma(K) = \{\lambda_1, \ldots, \lambda_{N+q}\}$ είναι το φάσμα των ιδιοτιμών του πίνακα $K$.

Επίσης έχει αποδειχθεί (βλέπε [10]), ότι το άνω φράγμα (3.16) για το μέγεθος του υπολοίπου που δίνει η προσέγγιση της λύσης που παράγει η MINRES στο βήμα $k$, είναι **βέλτιστο**, δηλαδή, **για κάθε $k$ υπάρχει ένα δεξί μέλος $b$, για το οποίο το άνω φράγμα θα ληφθεί!**

Επομένως το ερώτημα για την εκτίμηση του μεγέθους του υπολοίπου της MINRES στο βήμα $k$, ανάγεται σε ένα πρόβλημα της θεωρίας προσεγγίσεων: *Πόσο καλά μπορούμε να προσεγγίσουμε το μηδέν, από το σύνολο των ιδιοτιμών του πίνακα $K$, χρησιμοποιώντας ένα πολυώνυμο $k$ βαθμού, με τιμή 1 στο 0*;

Συγκεκριμένες κατασκευές για το παραπάνω πολυώνυμο, χρησιμοποιώντας πολυώνυμα Chebyshev, μπορούν να γίνουν για ένα συμμετρικό και μη ορισμένο πίνακα, αν έχουμε επιπλέον πληροφορίες για τα διαστήματα στα οποία περιέχονται οι ιδιοτιμές του (βλέπε [10]).

Από την εκτίμηση (3.16) γίνεται φανερό ότι **για ένα συμμετρικό πίνακα $K$ η ταχύτητα σύγκλισης της MINRES εξαρτάται αποκλειστικά και μόνο από τις ιδιοτιμές του**.

Στο σημείο αυτό πρέπει να σημειωθεί ότι σε όλα όσα αναφέραμε παραπάνω, υποθέσαμε ότι οι υπολογισμοί γίνονται σε **ακριβή αριθμητική**. Είναι γνωστό ότι **σε αριθμητική πεπερασμένης ακρίβειας η MINRES (όπως αντίστοιχα και η CG) δε βρίσκει τη βέλτιστη[1] προσέγγιση από τον υπόχωρο Krylov σε κάθε βήμα, ούτε καν κάτι κοντά σε αυτήν!!** Μάλιστα ένας από τους βασικούς λόγους που η CG αρχικά έχασε κάτι από την αίγλη της, ήταν το γεγονός ότι στους υπολογισμούς πεπερασμένης ακρίβειας του υπολογιστή, δε συμπεριφερόταν με τον τρόπο, που η θεωρία είχε προβλέψει σε ακριβή αριθμητική. Υπάρχουν διάφορες ερευνητικές εργασίες σχετικές με τη συμπεριφορά της MINRES (όπως και της CG) σε αριθμητική πεπερασμένης ακρίβειας (βλέπε [22]). Αρκετά ανοιχτά προβλήματα παραμένουν στην περιοχή αυτή.

## 3.7 H Preconditioned MINRES

Όπως είδαμε παραπάνω το βασικό ζήτημα, από το οποίο εξαρτάται άμεσα το συνολικό υπολογιστικό κόστος της μεθόδου MINRES, για την αριθμητική επίλυση ενός γραμμικού συστήματος, είναι ο αριθμός των επαναλήψεων που απαιτούνται για σύγκλιση στη λύση, με κάποια δεδομένη ικανοποιητική ακρίβεια. Είδαμε επίσης ότι **για ένα συμμετρικό πίνακα $K$ η ταχύτητα σύγκλισης της MINRES εξαρτάται αποκλειστικά και μόνο από τις ιδιοτιμές του**.

Μια καλή ιδέα λοιπόν για να προσπαθήσουμε να επιταχύνουμε τη σύγκλιση του γραμμικού συστήματος (2.4) είναι, **να μετασχηματίσουμε το γραμμικό σύστημα (2.4) σε ένα άλλο ισοδύναμο, ο πίνακας του οποίου να έχει καλύτερες φασματικές ιδιότητες**.

Η προρύθμιση συνίσταται στην εφαρμογή ενός γραμμικού τελεστή (πίνακα) $P$ στο αρχικό γραμμικό σύστημα (2.4) ώστε να πάρουμε το ισοδύναμο σύστημα:

$$P^{-1}Kx = P^{-1}b$$

Το παραπάνω γραμμικό σύστημα όμως, είναι γενικά **μη συμμετρικό**. Γενικά αποτελεί πολύ κακή κίνηση να μετασχηματίσουμε το αρχικό συμμετρικό γραμμικό σύστημα, σε ένα ισοδύναμο που είναι μη συμμετρικό: Η αριθμητική επίλυση μη συμμετρικών γραμμικών συστημάτων είναι γενικά πολύ λιγότερο αξιόπιστη και με πολύ πιο μεγάλο κόστος, σε σύγκριση με αυτή των

[1] Η MINRES δίνει βέλτιστη προσέγγιση, σε κάθε βήμα, με την έννοια ότι το αντίστοιχο υπόλοιπο έχει τη μικρότερη ευκλείδια νόρμα πάνω σε ολόκληρο τον αντίστοιχο υπόχωρο Krylov.

συμμετρικών γραμμικών συστημάτων. Γι' αυτό το λόγο θα κατασκευάσουμε προρυθμιστές που διατηρούν την αρχική συμμετρία του γραμμικού συστήματος.

Για να πετύχουμε το παραπάνω στην MINRES **θα πρέπει ο προρυθμιστής να είναι συμμετρικός και θετικά ορισμένος πίνακας.** Αφού λοιπόν ο $P$ είναι συμμετρικός και θετικά ορισμένος, έχει την παραγοντοποίηση: $P = LL^T$ για κάποιον πίνακα $L$ (π.χ. ο $L$ μπορεί να είναι ο παράγοντας $L$ από την Cholesky ή η τετραγωνική ρίζα του συμμετρικού και θετικά ορισμένου πίνακα $P$.) Στο σημείο αυτό να τονίσουμε πως η παραπάνω παραγοντοποίηση του προρυθμιστή είναι απλά μια μαθηματική επινόηση για να εξαγάγουμε το προρυθμισμένο σύστημα. Στην πράξη δεν απαιτείται καμμιά τέτοια παραγοντοποίηση του προρυθμιστή.

Η προρυθμισμένη MINRES λοιπόν, απλά συνίσταται στην εφαρμογή της MINRES, στο συμμετρικό σύστημα:

$$\begin{aligned} L^{-1}KL^{-T}y &= L^{-1}b, \\ L^Tx &= y. \end{aligned}$$

Η ταχύτητα σύγκλισης, όπως είδαμε στην προηγούμενη ενότητα, εξαρτάται αποκλειστικά από τις ιδιοτιμές του συμμετρικού και μη ορισμένου πίνακα $L^{-1}KL^{-T}$. Έχουμε όμως ότι:

$$L^{-T}\mathbf{L^{-1}KL^{-T}}L^T = (LL^T)^{-1}K = P^{-1}K \tag{3.17}$$

Η (3.17) ορίζει **μετασχηματισμό ομοιότητας** κι επομένως:

$$\sigma(L^{-1}KL^{-T}) = \sigma(P^{-1}K).$$

Συμπεραίνουμε λοιπόν ότι οι σημαντικές ιδιοτιμές είναι οι ιδιοτιμές του πίνακα: $P^{-1}K$. Σύμφωνα με τα παραπάνω, για τη σύγκλιση της προρυθμισμένης MINRES ισχύει η εκτίμηση (3.16) αν αντικαταστήσουμε το φάσμα ιδιοτιμών $\sigma(K)$ με το φάσμα ιδιοτιμών: $\sigma(P^{-1}K)$.

### 3.7.1 Κατασκευή μιας οικογένειας προρρυθμιστών της MINRES

Ακολουθώντας μια πρόταση η οποία γίνεται στα [4, 5] κατασκευάζουμε μια οικογένεια προρυθμιστών για τη MINRES ως εξής:

Ο προρυθμιστής πίνακας $M$ θα είναι μια «προσέγγιση» του πίνακα $K$, με την ακόλουθη έννοια:

Ο προρυθμιστής πίνακας $M$, θα είναι ένας πίνακας μπάντας, ο οποίος θα έχει για διαγώνιο, τη διαγώνιο του πίνακα $K$ και ενδεχομένως επιπλέον υπερδιαγωνίους και υποδιαγωνίους όμοιες με τις αντίστοιχες του $K$. Από αυτή την κατασκευή φαίνεται ότι ο προρυθμιστής πίνακας $M$ είναι συμμετρικός, δεν είναι όμως και θετικά ορισμένος, εν γένει. Όπως είδαμε η μέθοδος MINRES, προαπαιτεί τη χρήση ενός συμμετρικού και θετικά ορισμένου προρυθμιστή. Παρακάτω λοιπόν περιγράφουμε πως από το συμμετρικό αυτό πίνακα $M$, που προσεγγίζει τον $K$, θα πάρουμε ένα θετικά ορισμένο πίνακα, που σχετίζεται με τον $M$, με το ελάχιστο δυνατό κόστος.

Κάθε συμμετρικός και μη ιδιόμορφος πίνακας $M$, επιδέχεται την ακόλουθη παραγοντοποίηση:

$$M = P^TU^TDUP,$$

όπου $P$ είναι ένας πίνακας μετάθεσης, $U$ είναι άνω τριγωνικός και $D$ είναι block διαγώνιος πίνακας, με διαγώνια blocks διάστασης 1 ή 2. Η παραπάνω παραγοντοποίηση ενός συμμετρικού

και μη ιδιόμορφου πίνακα, είναι γνωστή ως παραγοντοποίηση Bunch-Parlett (βλέπε [2] ) και αποτελεί το αντίστοιχο της ανάλυσης Crout (ανάλυσης Cholesky) για συμμετρικούς αλλά εν γένει μη ορισμένους πίνακες.

Στην περίπτωση που ο πίνακας $M$ είναι μη ορισμένος, κάποια από τα διαγώνια blocks του $D$ θα έχουν αρνητικές ιδιοτιμές. Ο πίνακας $D$ είναι συμμετρικός, άρα διαγωνοποιείται μέσω ορθογώνιου πίνακα, έστω $Q$:

$$D = Q\Lambda Q^T, \Lambda = diag(\lambda_j).$$

Έστω τώρα ο πίνακας:

$$\bar{D} = Q\bar{\Lambda}Q^T, \bar{\Lambda} = diag(|\lambda_j|),$$

ο οποίος συνδέεται στενά με τον $D$ και είναι συμμετρικός και θετικά ορισμένος. Επιπλέον, ο $\bar{D}$ μπορεί να κατασκευαστεί από τον $D$, με ελάχιστο κόστος. Ο προρρυθμιστής που τελικά παίρνουμε είναι ο πίνακας:

$$\bar{M} = CC^T = P^T U^T \bar{D} U P, \text{όπου } C = P^T U^T \bar{D}^{1/2}.$$

Ο πίνακας $\bar{M}$ είναι συμμετρικός και θετικά ορισμένος, εκ κατασκευής.

Σημείωση: Επειδή το $(2,2)$ block του πίνακα $K$ είναι μηδενικό, τα μηδενικά που υπάρχουν στη διαγώνιο του προρυθμιστή που θα κατασκευαστεί με τον παραπάνω τρόπο, τον κάνουν να έχει κακή κατάσταση. Για το λόγο αυτό αντί για μηδέν στη διαγώνιο του $(2,2)$ block του $K$, βάζουμε μια παράμετρο $r$, που είναι ένας μικρός θετικός αριθμός:

$$\mathbf{M} = \begin{bmatrix} * & * & * & & & & & & \\ * & * & * & * & & & & & \\ * & \ddots & \ddots & \ddots & \ddots & & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & & \\ & & * & * & * & * & * & & \\ & & & \ddots & \ddots & r & \ddots & \ddots & \\ & & & & \ddots & \ddots & r & \ddots & * \\ & & & & & \ddots & \ddots & \ddots & * \\ & & & & & & * & * & r \end{bmatrix}$$

όπου τα αστεράκια δηλώνουν τα αντίστοιχα στοιχεία του πίνακα $K$.

Ο κώδικας που έχουμε υλοποιήσει δέχεται δυο παραμέτρους: $r$ και *band*. Η παράμετρος $r$ (μικρός θετικός αριθμός) είναι αυτή που μόλις αναλύσαμε παραπάνω. Η παράμετρος *band* παίρνει τιμές στο σύνολο $\{0, 1, 2, 3, \ldots, N+q-1\}$ και καθορίζει το εύρος της μπάντας του $K$ που θα χρησιμοποιήσουμε για την κατασκευή του προρυθμιστή.

Ο κώδικας που έχει υλοποιηθεί μπορεί να δεχτεί τιμές της παραμέτρου *band* στο σύνολο $\{0, 1, 2, 3\}$. Είναι δυνατόν, όπως είναι υλοποιημένος ο κώδικάς μας, να δεχθεί ακόμα μεγαλύτερες τιμές της παραμέτρου *band*, αν τροποποιηθούν κατάλληλα τα μήκη κάποιων διανυσμάτων. Λεπτομέρειες για το πώς ακριβώς πρέπει να γίνει αυτό, αναγράφω στο αντίστοιχο αρχείο κώδικα που έχω υλοποιήσει.

Σε όλα τα αριθμητικά πειράματα που έγιναν χρησιμοποίησα τις εξής τιμές των παραμέτρων:

$$band = 2, r = 0.22$$

Μια γρήγορη δοκιμή έδειξε πως μεγάλες τιμές του $r$ δίνουν αναποτελεσματικούς προρρυθμιστές όσον αφορά τον απαιτούμενο υπολογιστικό χρόνο. Η επίλυση του γραμμικού συστήματος $\bar{M}x = y$ με τον προρυθμιστή, σε κάθε επανάληψη, γίνεται με χρήση του κώδικα MA27 (βλέπε [17] ).

## 3.8 Άμεση επίλυση του γραμμικού συστήματος (MA57)

Για την άμεση επίλυση του γραμμικού συστήματος (2.4) χρησιμοποιήθηκε μια παραλλαγή της απαλοιφής Gauss για μεγάλους, αραιούς, συμμετρικούς και ,εν γένει, μη ορισμένους πίνακες. Χρησιμοποιήθηκαν οι κώδικες της μεθόδου MA57 από την εμπορική μαθηματική βιβλιοθήκη επιστημονικών υπολογισμών μεγάλης κλίμακας, HSL (Harwell Subroutine Library) (βλέπε [21]).

Ο αλγόριθμος MA57 (βλέπε [18]) αντικαθιστά τον αντίστοιχο παλαιότερο MA27.[2](βλέπε [17]). Οι βασικές διαφορές τους, είναι η αποκλειστική χρήση ρουτίνων από το BLAS για πράξεις μεταξύ διανυσμάτων και πινάκων, η οποία δε γινόταν στο MA27, καθώς και η επιλογή για χρήση συγκεκριμένης συνάρτησης του κώδικα του προγράμματος MeTiS (βλέπε [20]) για τη *διάταξη* (ordering) του πίνακα. Η συνάρτηση αυτή από το MeTiS *γενικά δίνει πολύ καλύτερες διατάξεις για την απαλοιφή και έτσι εξοικονομεί αρκετά σε μνήμη και σε υπολογιστικό χρόνο, ειδικά για πολύ μεγάλους πίνακες.*

Παραλείποντας τις πάρα πολλές και τεχνικές λεπτομέρειες, η επίλυση ενός συμμετρικού και εν γένει μη ορισμένου, γραμμικού συστήματος με τον κώδικα MA57 ακολουθεί τις παρακάτω φάσεις:

1. Αρχικά καλείται η συνάρτηση MA57ID με την οποία γίνεται *αρχικοποίηση διαφόρων παραμέτρων* που έχουν να κάνουν με την απαλοιφή και την οδήγηση. Ο χρήστης μπορεί να μεταβάλλει τις default τιμές των παραμέτρων, ώστε να προσαρμόσει ακόμα καλύτερα τον αλγόριθμο σε πιο συσκεκριμένες ιδιότητες του γραμμικού συστήματος που επιλύει.

2. Στη συνέχεια καλείται η συνάρτηση MA57AD η οποία δέχεται το μοτίβο του πίνακα (τις θέσεις των μη μηδενικών του στοιχείων) και επιλέγει τους οδηγούς που θα χρησιμοποιηθούν στην απαλοιφή. Η επιλογή των οδηγών βασίζεται είτε σε MeTiS ordering είτε σε *διάταξη κατά προσέγγιση ελάχιστου βαθμού (approximate minimum degree ordering)*, με δυνατότητα επιλογής από το χρήστη. Η επιλογή των οδηγών είναι πολύ κρίσιμη και από αυτήν εξαρτάται άμεσα η απόδοση του κώδικα. Γενικά η οδήγηση που βασίζεται στο πρόγραμμα MeTiS δίνει τα καλύτερα αποτελέσματα για μεγάλους και αραιούς πίνακες, γι΄ αυτό και προτείνεται η χρήση του. Χρησιμοποιούνται δέντρα απαλοιφής για την αναπαράσταση της διαδικασίας. Η παραπάνω διαδικασία είναι γνωστή ως **συμβολική απαλοιφή** (symbolic elimination).

3. Αμέσως μετά καλείται η συνάρτηση MA57BD η οποία κάνει την απαλοιφή (numerical elimination), χρησιμοποιώντας τη διάταξη που υπολόγισε η MA57AD ή κάποια τροποποιημένη, αν αυτό κριθεί απαραίτητο για λόγους οδήγησης.

4. Τέλος καλείται η συνάρτηση MA57CD η οποία χρησιμοποιώντας το προιόν της απαλοιφής που έδωσε η MA57BD και το δεξί μέλος, λύνει το σύστημα.

[2]Ο αλγόριθμος MA27 αποτελεί τον πρώτο κώδικα άμεσης επίλυσης συμμετρικών γραμμικών συστημάτων, που βασίζεται σε multifrontal μέθοδο.

# Κεφάλαιο 4

# Αριθμητικά πειράματα - υλοποίηση των μεθόδων

## 4.1 Τα γραμμικά συστήματα που επιλύθηκαν

Για την σύγκριση των μεθόδων που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, επιλύθηκαν τριών διαφορετικών μεγεθών γραμμικά συστήματα με πίνακα $K$, για κάθε μέθοδο:

- Ένα «μικρό», μεγέθους $832 \times 832$
- Ένα «μεσαίο», μεγέθους $3200 \times 3200$
- Ένα «μεγάλο», μεγέθους $12544 \times 12544$

Επίσης για κάθε κατηγορία μεγέθους επιλύθηκαν 3 γραμμικά συστήματα με πίνακα $K$, για κάθε μέθοδο:

- Ένα γραμμικό σύστημα που αντιστοιχεί σε διακριτοποίηση με αριθμό Reynolds, $Re = 10$
- Ένα γραμμικό σύστημα που αντιστοιχεί σε διακριτοποίηση με αριθμό Reynolds, $Re = 100$
- Ένα γραμμικό σύστημα που αντιστοιχεί σε διακριτοποίηση με αριθμό Reynolds,$Re = 1000$

Συνολικά δηλαδή επιλύθηκαν 9 γραμμικά συστήματα για κάθε μέθοδο. Για να αναφερθούμε σε καθένα από τα 9 συνολικά γραμμικά συστήματα που επιλύθηκαν, για κάθε μέθοδο, χρησιμοποιούμε τον παρακάτω συμβολισμό για τον πίνακα $K$:

$$\mathbf{K_X^{(Re)}} = \begin{bmatrix} A_X^{(Re)} & B_X \\ B_X^T & O \end{bmatrix} \in \mathbb{R}^{(N+q)\times(N+q)}$$

, όπου οι παράμετροι $X \in \{S, M, L\}$, και $Re \in \{10, 100, 1000\}$ μας υποδεικνύουν, σε ποια κατηγορία μεγέθους είναι ο πίνακας και την τιμή του αριθμού Reynolds αντίστοιχα. Όλοι οι πίνακες που δοκιμάστηκαν, έχουν ληφθεί με χρήση του λογισμικού IFISS (βλέπε [24]).

Το δεξί μέλος $\mathbf{b} \in \mathbb{R}^{N+q}$ του κάθε γραμμικού συστήματος $\mathbf{K_X^{(Re)}x = b}$, σε κάθε περίπτωση, επιλέχθηκε ως:

$$b_i = \sum_{j=1}^{N+q} \mathbf{K}_{\mathbf{X_{ij}}}^{\mathbf{(Re)}}, \forall i \in \{1, \ldots, N+q\}, \forall X \in \{S, M, L\}, \forall Re \in \{10, 100, 1000\}$$

| Όνομα | διάσταση $\mathbf{K_X^{(Re)}}$ | διάσταση $\mathbf{A_X^{(Re)}}$ | διάσταση $\mathbf{B_X}$ | $nz(\mathbf{A_X^{(Re)}})$ | $nz(\mathbf{B_X})$ | $nz(\mathbf{K_X^{(Re)}})$ |
|---|---|---|---|---|---|---|
| $\mathbf{K_S^{(10)}}$ | 832 × 832 | 578 × 578 | 578 × 254 | 3826 | 1794 | 7414 |
| $\mathbf{K_S^{(100)}}$ | 832 × 832 | 578 × 578 | 578 × 254 | 3826 | 1794 | 7414 |
| $\mathbf{K_S^{(1000)}}$ | 832 × 832 | 578 × 578 | 578 × 254 | 3826 | 1794 | 7414 |
| $\mathbf{K_M^{(10)}}$ | 3200 × 3200 | 2178 × 2178 | 2178 × 1022 | 16818 | 7682 | 32182 |
| $\mathbf{K_M^{(100)}}$ | 3200 × 3200 | 2178 × 2178 | 2178 × 1022 | 16818 | 7682 | 32182 |
| $\mathbf{K_M^{(1000)}}$ | 3200 × 3200 | 2178 × 2178 | 2178 × 1022 | 16818 | 7682 | 32182 |
| $\mathbf{K_L^{(10)}}$ | 12544 × 12544 | 8450 × 8450 | 8450 × 4094 | 70450 | 31746 | 133942 |
| $\mathbf{K_L^{(100)}}$ | 12544 × 12544 | 8450 × 8450 | 8450 × 4094 | 70450 | 31746 | 133942 |
| $\mathbf{K_L^{(1000)}}$ | 12544 × 12544 | 8450 × 8450 | 8450 × 4094 | 70450 | 31746 | 133942 |

Πίνακας 4.1: Τα μεγέθη και το πλήθος των μη μηδενικών στοιχείων των πινάκων

| $\mathbf{A_X^{(Re)}}$ | $\lambda_{\max}(\mathbf{A_X^{(Re)}})$ | $\lambda_{\min}(\mathbf{A_X^{(Re)}})$ | $\kappa_1(\mathbf{A_X^{(Re)}})$ | $\kappa_2(\mathbf{A_X^{(Re)}})$ | $\kappa_\infty(\mathbf{A_X^{(Re)}})$ | $\kappa_{Fro}(\mathbf{A_X^{(Re)}})$ |
|---|---|---|---|---|---|---|
| $\mathbf{A_S^{(10)}}$ | 3.9493e-001 | 7.6367e-003 | 1.0090e+002 | 5.1714e+001 | 1.0090e+002 | 1.6705e+003 |
| $\mathbf{A_S^{(100)}}$ | 3.9493e-002 | 7.6367e-004 | 1.0090e+002 | 5.1714e+001 | 1.0090e+002 | 1.6705e+003 |
| $\mathbf{A_S^{(1000)}}$ | 3.9493e-003 | 7.6367e-005 | 1.0090e+002 | 5.1714e+001 | 1.0090e+002 | 1.6705e+003 |
| $\mathbf{A_M^{(10)}}$ | 3.9872e-001 | 1.9230e-003 | 4.0265e+002 | 2.0734e+002 | 4.0265e+002 | 1.2266e+004 |
| $\mathbf{A_M^{(100)}}$ | 3.9872e-002 | 1.9230e-004 | 4.0265e+002 | 2.0734e+002 | 4.0265e+002 | 1.2266e+004 |
| $\mathbf{A_M^{(1000)}}$ | 3.9872e-003 | 1.9230e-005 | 4.0265e+002 | 2.0734e+002 | 4.0265e+002 | 1.2266e+004 |
| $\mathbf{A_L^{(10)}}$ | 3.9968e-001 | 4.8162e-004 | 1.6097e+003 | 8.2986e+002 | 1.6097e+003 | 9.7218e+004 |
| $\mathbf{A_L^{(100)}}$ | 3.9968e-002 | 4.8162e-005 | 1.6097e+003 | 8.2986e+002 | 1.6097e+003 | 9.7218e+004 |
| $\mathbf{A_L^{(1000)}}$ | 3.9968e-003 | 4.8162e-006 | 1.6097e+003 | 8.2986e+002 | 1.6097e+003 | 9.7218e+004 |

Πίνακας 4.2: Μεγαλύτερη,μικρότερη ιδιοτιμή και δείκτες κατάστασης των πινάκων $\mathbf{A_X^{(Re)}}$

| $\mathbf{K_X^{(Re)}}$ | $\lambda_{\max}(\mathbf{K_X^{(Re)}})$ | $\lambda_{\min}(\mathbf{K_X^{(Re)}})$ | $\kappa_1(\mathbf{K_X^{(Re)}})$ | $\kappa_2(\mathbf{K_X^{(Re)}})$ | $\kappa_\infty(\mathbf{K_X^{(Re)}})$ | $\kappa_{Fro}(\mathbf{K_X^{(Re)}})$ |
|---|---|---|---|---|---|---|
| $\mathbf{K_S^{(10)}}$ | 5.1416e-001 | -1.1924e-001 | 8.7164e+003 | 4.5868e+003 | 8.7164e+003 | 6.6995e+004 |
| $\mathbf{K_S^{(100)}}$ | 2.6814e-001 | -2.2865e-001 | 7.3550e+002 | 2.4854e+002 | 7.3550e+002 | 6.8346e+003 |
| $\mathbf{K_S^{(1000)}}$ | 2.4959e-001 | -2.4564e-001 | 4.5260e+003 | 1.2515e+003 | 4.5260e+003 | 5.5482e+004 |
| $\mathbf{K_M^{(10)}}$ | 4.3451e-001 | -3.5788e-002 | 1.3629e+005 | 7.5662e+004 | 1.3629e+005 | 2.3586e+006 |
| $\mathbf{K_M^{(100)}}$ | 1.4622e-001 | -1.0635e-001 | 5.4727e+003 | 2.5514e+003 | 5.4727e+003 | 7.5552e+004 |
| $\mathbf{K_M^{(1000)}}$ | 1.2671e-001 | -1.2272e-001 | 8.8927e+003 | 2.4943e+003 | 8.8927e+003 | 1.5072e+005 |
| $\mathbf{K_L^{(10)}}$ | 4.0921e-001 | -9.5343e-003 | 2.2669e+006 | 1.3429e+006 | 2.2669e+006 | 8.6264e+007 |
| $\mathbf{K_L^{(100)}}$ | 8.5565e-002 | -4.5597e-002 | 4.8163e+004 | 2.8083e+004 | 4.8163e+004 | 1.5799e+006 |
| $\mathbf{K_L^{(1000)}}$ | 6.4493e-002 | -6.0496e-002 | 1.7728e+004 | 5.0545e+003 | 1.7728e+004 | 5.6497e+005 |

Πίνακας 4.3: Μεγαλύτερη,μικρότερη ιδιοτιμή και δείκτες κατάστασης των πινάκων $\mathbf{K_X^{(Re)}}$

ώστε το διάνυσμα $\mathbf{x} = (1, \ldots, 1) \in \mathbb{R}^{N+q}$ να είναι η μοναδική λύση καθενός από τα παραπάνω γραμμικά συστήματα.

Στον πίνακα (4.1) φαίνονται τα μεγέθη και το πλήθος των μη μηδενικών στοιχείων των πινάκων που δοκιμάστηκαν σε κάθε μέθοδο. Από τη δομή των πινάκων $\mathbf{K}_{\mathbf{X}}^{(\mathbf{Re})}$, είναι φανερό ότι:

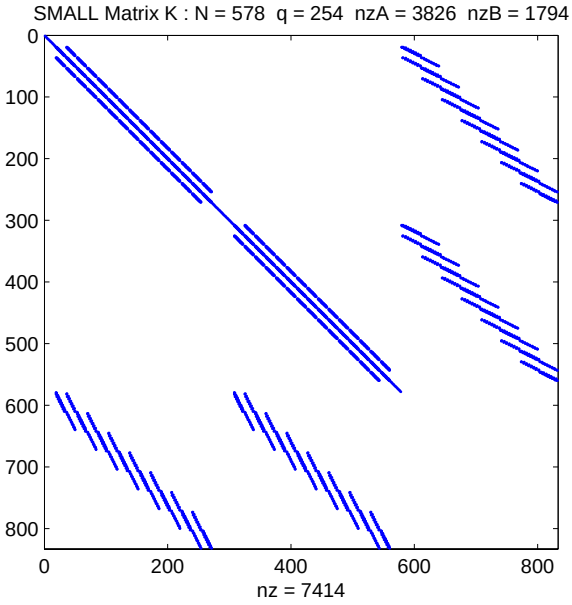
$$nz(\mathbf{K}_{\mathbf{X}}^{(\mathbf{Re})}) = nz(\mathbf{A}_{\mathbf{X}}^{(\mathbf{Re})}) + 2\ nz(\mathbf{B}_{\mathbf{X}}),$$

όπου $nz(\cdot)$ είναι το πλήθος των μη μηδενικών στοιχείων.

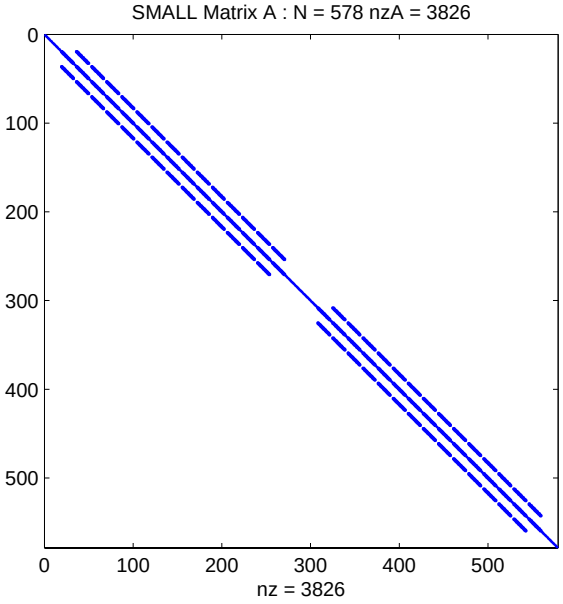
Στον πίνακα (4.2) φαίνονται η μεγαλύτερη ιδιοτιμή, η μικρότερη ιδιοτιμή καθώς και οι δείκτες κατάστασης στις νόρμες $\|\cdot\|_1, \|\cdot\|_2, \|\cdot\|_\infty, \|\cdot\|_{Fro}$, για τους πίνακες $\mathbf{A}_{\mathbf{X}}^{(\mathbf{Re})}$.

Στον πίνακα (4.3) φαίνονται η μεγαλύτερη ιδιοτιμή, η μικρότερη ιδιοτιμή καθώς και οι δείκτες κατάστασης στις νόρμες $\|\cdot\|_1, \|\cdot\|_2, \|\cdot\|_\infty, \|\cdot\|_{Fro}$, για τους πίνακες $\mathbf{K}_{\mathbf{X}}^{(\mathbf{Re})}$.

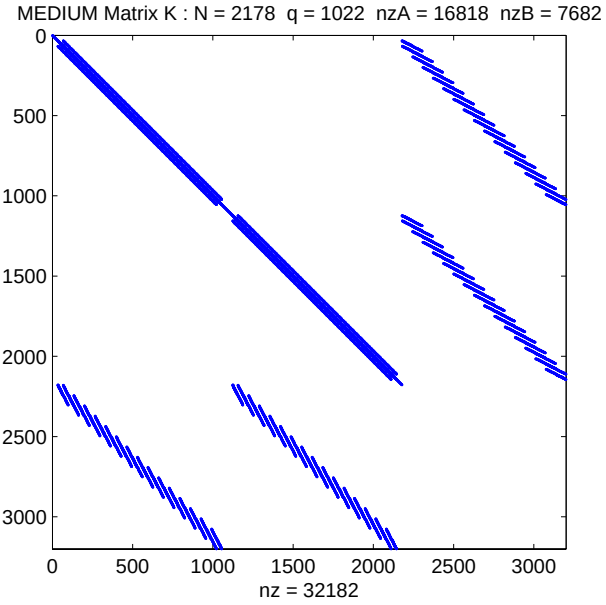
Στα σχήματα (4.1 - 4.6), φαίνεται η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{K}_{\mathbf{X}}^{(\mathbf{Re})}$ και $\mathbf{A}_{\mathbf{X}}^{(\mathbf{Re})}$.



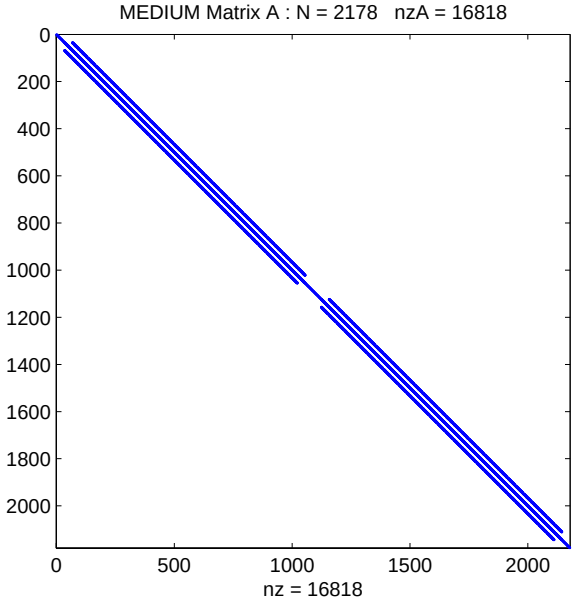

Σχήμα 4.1: Η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{K_S^{(Re)}}$



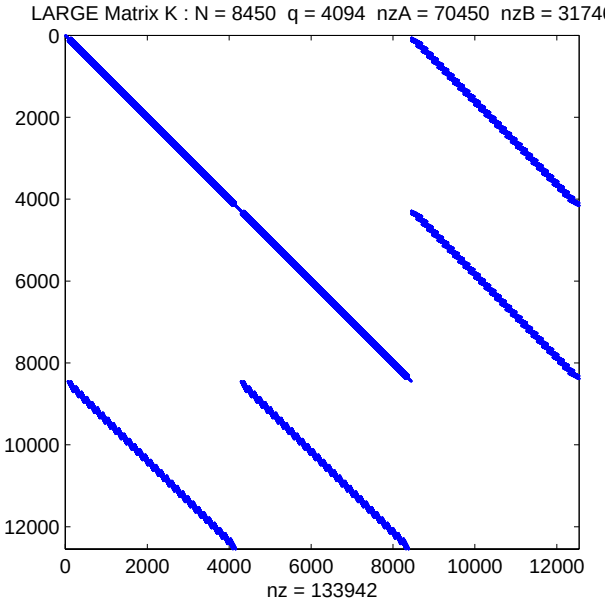

Σχήμα 4.2: Η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{A_S^{(Re)}}$



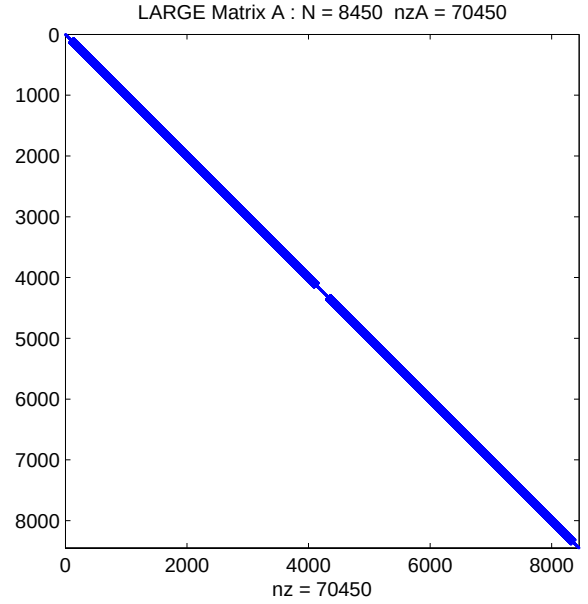

Σχήμα 4.3: Η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{K_M^{(Re)}}$




Σχήμα 4.4: Η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{A_M^{(Re)}}$




Σχήμα 4.5: Η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{K_L^{(Re)}}$




Σχήμα 4.6: Η δομή των μη μηδενικών στοιχείων των πινάκων $\mathbf{A_L^{(Re)}}$

## 4.2 Γενικά για την υλοποίηση των μεθόδων - αποτελέσματα.

Όλες οι μέθοδοι που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, υλοποιήθηκαν σε κώδικες C και Fortran 77 (βλέπε Παράρτημα Α′). Σε όλες τις πράξεις μεταξύ διανυσμάτων και πινάκων έγινε χρήση κατάλληλων υπορουτίνων από το BLAS για βελτιστοποίηση της απόδοσης.

Όλοι οι κώδικες έτρεξαν στο περιβάλλον Cygwin πάνω από το λειτουργικό σύστημα Microsoft Windows 7 Ultimate 64 - bit. Το μηχάνημα ήταν εφοδιασμένο με τον τετραπύρηνο επεξεργαστή Intel Core i7-975 Extreme Edition και με 6 GB μνήμης, χρονισμένης στα 2000 MHz.

## 4.3 Αριθμητικά Αποτελέσματα

Στη συνέχεια παραθέτουμε τα αριθμητικά αποτελέσματα και ένα σύνολο από γραφικές παραστάσεις για κάθε μέθοδο, τα οποία δείχνουν τις συμπεριφορές των μεθόδων, καθώς και τις εξαρτήσεις τους από τις διάφορες παραμέτρους.

### 4.3.1 Αποτελέσματα Μεθόδου Projected CG

Στους πίνακες (4.4-4.5), παραθέτουμε τα αποτελέσματα για τη μέθοδο Projected CG. Στα σχήματα (4.7)–(4.15) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου Projected CG. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου Projected CG:

- Σε όλες τις περιπτώσεις, η Projected CG χρειάστηκε 1 επανάληψη για σύγκλιση και μάλιστα το αντίστοιχο υπόλοιπο της CG είναι της τάξης $10^{-13}$ – $10^{-16}$.

- Σε όλες τις κατηγορίες διαστάσεων των πινάκων και για όλες τις τιμές της ανοχής TOL που δοκιμάστηκαν, όσο αυξάνει ο αριθμός Reynolds της ροής, μειώνεται η υπολογιζόμενη ευκλείδια νόρμα $\|f - Au - Bp\|_2$, ενώ η ευκλείδια νόρμα $\|g - B^T u\|_2$, αυξάνεται ελάχιστα (βλέπε σχήματα (4.10) – (4.12) *Re vs* $\|f - Au - Bp\|_2$).

- Καθώς αυξάνεται η διάσταση (DOF) του γραμμικού συστήματος, αυξάνεται ο απαιτούμενος υπολογιστικός χρόνος, ομοιόμορφα ως προς τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής TOL, που δοκιμάστηκαν (βλέπε σχήματα (4.13) – (4.15) DOF vs CPU).

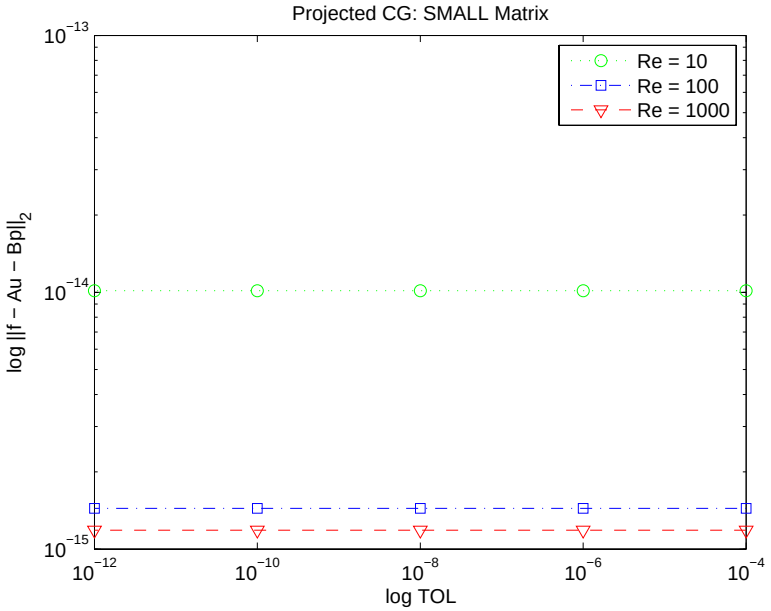
| TOL | Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | #iter | CPU time (sec) |
|---|---|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 1.014389760e-14 | 7.279948071e-15 | 1.248584408e-14 | 1 | 0.062000 |
| 1.0e-06 | 10 | 832 | 1.014389760e-14 | 7.279948071e-15 | 1.248584408e-14 | 1 | 0.047000 |
| 1.0e-08 | 10 | 832 | 1.014389760e-14 | 7.279948071e-15 | 1.248584408e-14 | 1 | 0.047000 |
| 1.0e-10 | 10 | 832 | 1.014389760e-14 | 7.279948071e-15 | 1.248584408e-14 | 1 | 0.046000 |
| 1.0e-12 | 10 | 832 | 1.014389760e-14 | 7.279948071e-15 | 1.248584408e-14 | 1 | 0.047000 |
| 1.0e-04 | 100 | 832 | 1.441544121e-15 | 1.048558021e-14 | 1.058420719e-14 | 1 | 0.031000 |
| 1.0e-06 | 100 | 832 | 1.441544121e-15 | 1.048558021e-14 | 1.058420719e-14 | 1 | 0.063000 |
| 1.0e-08 | 100 | 832 | 1.441544121e-15 | 1.048558021e-14 | 1.058420719e-14 | 1 | 0.047000 |
| 1.0e-10 | 100 | 832 | 1.441544121e-15 | 1.048558021e-14 | 1.058420719e-14 | 1 | 0.046000 |
| 1.0e-12 | 100 | 832 | 1.441544121e-15 | 1.048558021e-14 | 1.058420719e-14 | 1 | 0.047000 |
| 1.0e-04 | 1000 | 832 | 1.185068936e-15 | 5.677064603e-14 | 5.678301365e-14 | 1 | 0.047000 |
| 1.0e-06 | 1000 | 832 | 1.185068936e-15 | 5.677064603e-14 | 5.678301365e-14 | 1 | 0.062000 |
| 1.0e-08 | 1000 | 832 | 1.185068936e-15 | 5.677064603e-14 | 5.678301365e-14 | 1 | 0.047000 |
| 1.0e-10 | 1000 | 832 | 1.185068936e-15 | 5.677064603e-14 | 5.678301365e-14 | 1 | 0.047000 |
| 1.0e-12 | 1000 | 832 | 1.185068936e-15 | 5.677064603e-14 | 5.678301365e-14 | 1 | 0.047000 |
| 1.0e-04 | 10 | 3200 | 2.863078213e-14 | 1.817846619e-14 | 3.391427898e-14 | 1 | 0.764000 |
| 1.0e-06 | 10 | 3200 | 2.863078213e-14 | 1.817846619e-14 | 3.391427898e-14 | 1 | 0.765000 |
| 1.0e-08 | 10 | 3200 | 2.863078213e-14 | 1.817846619e-14 | 3.391427898e-14 | 1 | 0.764000 |
| 1.0e-10 | 10 | 3200 | 2.863078213e-14 | 1.817846619e-14 | 3.391427898e-14 | 1 | 0.765000 |
| 1.0e-12 | 10 | 3200 | 2.863078213e-14 | 1.817846619e-14 | 3.391427898e-14 | 1 | 0.780000 |
| 1.0e-04 | 100 | 3200 | 2.930937031e-15 | 1.921747525e-14 | 1.943969513e-14 | 1 | 0.749000 |
| 1.0e-06 | 100 | 3200 | 2.930937031e-15 | 1.921747525e-14 | 1.943969513e-14 | 1 | 0.779000 |
| 1.0e-08 | 100 | 3200 | 2.930937031e-15 | 1.921747525e-14 | 1.943969513e-14 | 1 | 0.764000 |
| 1.0e-10 | 100 | 3200 | 2.930937031e-15 | 1.921747525e-14 | 1.943969513e-14 | 1 | 0.765000 |
| 1.0e-12 | 100 | 3200 | 2.930937031e-15 | 1.921747525e-14 | 1.943969513e-14 | 1 | 0.765000 |
| 1.0e-04 | 1000 | 3200 | 8.319227723e-16 | 4.332554935e-14 | 4.333353577e-14 | 1 | 0.764000 |
| 1.0e-06 | 1000 | 3200 | 8.319227723e-16 | 4.332554935e-14 | 4.333353577e-14 | 1 | 0.764000 |
| 1.0e-08 | 1000 | 3200 | 8.319227723e-16 | 4.332554935e-14 | 4.333353577e-14 | 1 | 0.781000 |
| 1.0e-10 | 1000 | 3200 | 8.319227723e-16 | 4.332554935e-14 | 4.333353577e-14 | 1 | 0.764000 |
| 1.0e-12 | 1000 | 3200 | 8.319227723e-16 | 4.332554935e-14 | 4.333353577e-14 | 1 | 0.765000 |
| 1.0e-04 | 10 | 12544 | 1.811466551e-13 | 4.610326320e-14 | 1.869214315e-13 | 1 | 22.963000 |
| 1.0e-06 | 10 | 12544 | 1.811466551e-13 | 4.610326320e-14 | 1.869214315e-13 | 1 | 22.978000 |
| 1.0e-08 | 10 | 12544 | 1.811466551e-13 | 4.610326320e-14 | 1.869214315e-13 | 1 | 22.964000 |
| 1.0e-10 | 10 | 12544 | 1.811466551e-13 | 4.610326320e-14 | 1.869214315e-13 | 1 | 22.963000 |
| 1.0e-12 | 10 | 12544 | 1.811466551e-13 | 4.610326320e-14 | 1.869214315e-13 | 1 | 22.964000 |
| 1.0e-04 | 100 | 12544 | 1.798606392e-14 | 4.653238584e-14 | 4.988748769e-14 | 1 | 22.963000 |
| 1.0e-06 | 100 | 12544 | 1.798606392e-14 | 4.653238584e-14 | 4.988748769e-14 | 1 | 22.995000 |
| 1.0e-08 | 100 | 12544 | 1.798606392e-14 | 4.653238584e-14 | 4.988748769e-14 | 1 | 22.978000 |
| 1.0e-10 | 100 | 12544 | 1.798606392e-14 | 4.653238584e-14 | 4.988748769e-14 | 1 | 22.980000 |
| 1.0e-12 | 100 | 12544 | 1.798606392e-14 | 4.653238584e-14 | 4.988748769e-14 | 1 | 22.947000 |
| 1.0e-04 | 1000 | 12544 | 1.982953923e-15 | 5.445756579e-14 | 5.449365631e-14 | 1 | 22.995000 |
| 1.0e-06 | 1000 | 12544 | 1.982953923e-15 | 5.445756579e-14 | 5.449365631e-14 | 1 | 22.979000 |
| 1.0e-08 | 1000 | 12544 | 1.982953923e-15 | 5.445756579e-14 | 5.449365631e-14 | 1 | 22.978000 |
| 1.0e-10 | 1000 | 12544 | 1.982953923e-15 | 5.445756579e-14 | 5.449365631e-14 | 1 | 22.949000 |
| 1.0e-12 | 1000 | 12544 | 1.982953923e-15 | 5.445756579e-14 | 5.449365631e-14 | 1 | 22.979000 |

Πίνακας 4.4: Αποτελέσματα για τη μέθοδο Projected CG

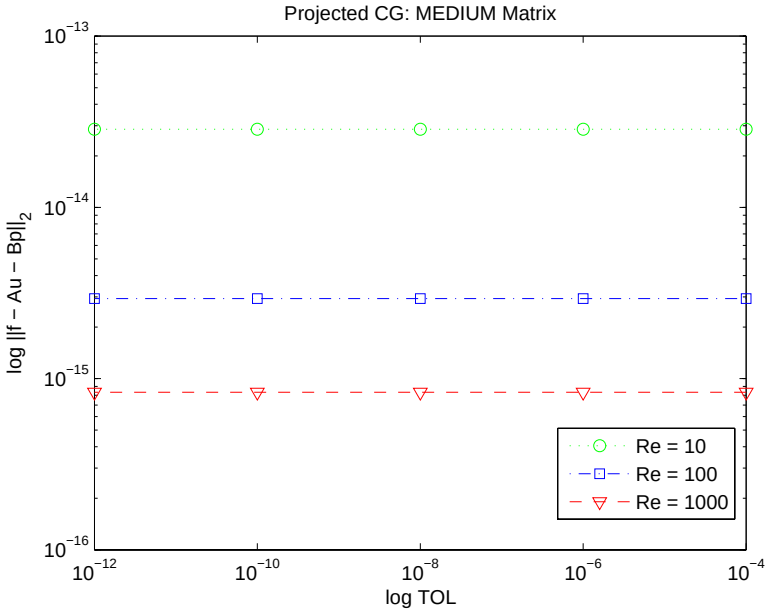
| TOL | Re | DOF | $\|x - x^*\|_1$ | $\|x - x^*\|_2$ | $\|x - x^*\|_\infty$ | CG $\|r\|_2$ |
|---|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 6.487121951e-11 | 4.678009807e-12 | 4.909406215e-13 | 1.011982918e-14 |
| 1.0e-06 | 10 | 832 | 6.487121951e-11 | 4.678009807e-12 | 4.909406215e-13 | 1.011982918e-14 |
| 1.0e-08 | 10 | 832 | 6.487121951e-11 | 4.678009807e-12 | 4.909406215e-13 | 1.011982918e-14 |
| 1.0e-10 | 10 | 832 | 6.487121951e-11 | 4.678009807e-12 | 4.909406215e-13 | 1.011982918e-14 |
| 1.0e-12 | 10 | 832 | 6.487121951e-11 | 4.678009807e-12 | 4.909406215e-13 | 1.011982918e-14 |
| 1.0e-04 | 100 | 832 | 1.480526812e-11 | 9.687975202e-13 | 1.174615960e-13 | 1.407125759e-15 |
| 1.0e-06 | 100 | 832 | 1.480526812e-11 | 9.687975202e-13 | 1.174615960e-13 | 1.407125759e-15 |
| 1.0e-08 | 100 | 832 | 1.480526812e-11 | 9.687975202e-13 | 1.174615960e-13 | 1.407125759e-15 |
| 1.0e-10 | 100 | 832 | 1.480526812e-11 | 9.687975202e-13 | 1.174615960e-13 | 1.407125759e-15 |
| 1.0e-12 | 100 | 832 | 1.480526812e-11 | 9.687975202e-13 | 1.174615960e-13 | 1.407125759e-15 |
| 1.0e-04 | 1000 | 832 | 2.120104092e-11 | 1.383995108e-12 | 6.059597268e-13 | 1.176040308e-15 |
| 1.0e-06 | 1000 | 832 | 2.120104092e-11 | 1.383995108e-12 | 6.059597268e-13 | 1.176040308e-15 |
| 1.0e-08 | 1000 | 832 | 2.120104092e-11 | 1.383995108e-12 | 6.059597268e-13 | 1.176040308e-15 |
| 1.0e-10 | 1000 | 832 | 2.120104092e-11 | 1.383995108e-12 | 6.059597268e-13 | 1.176040308e-15 |
| 1.0e-12 | 1000 | 832 | 2.120104092e-11 | 1.383995108e-12 | 6.059597268e-13 | 1.176040308e-15 |
| 1.0e-04 | 10 | 3200 | 1.917370041e-09 | 6.115695759e-11 | 2.653433029e-12 | 2.865800754e-14 |
| 1.0e-06 | 10 | 3200 | 1.917370041e-09 | 6.115695759e-11 | 2.653433029e-12 | 2.865800754e-14 |
| 1.0e-08 | 10 | 3200 | 1.917370041e-09 | 6.115695759e-11 | 2.653433029e-12 | 2.865800754e-14 |
| 1.0e-10 | 10 | 3200 | 1.917370041e-09 | 6.115695759e-11 | 2.653433029e-12 | 2.865800754e-14 |
| 1.0e-12 | 10 | 3200 | 1.917370041e-09 | 6.115695759e-11 | 2.653433029e-12 | 2.865800754e-14 |
| 1.0e-04 | 100 | 3200 | 2.444875413e-10 | 7.421832845e-12 | 3.208544541e-13 | 2.914642833e-15 |
| 1.0e-06 | 100 | 3200 | 2.444875413e-10 | 7.421832845e-12 | 3.208544541e-13 | 2.914642833e-15 |
| 1.0e-08 | 100 | 3200 | 2.444875413e-10 | 7.421832845e-12 | 3.208544541e-13 | 2.914642833e-15 |
| 1.0e-10 | 100 | 3200 | 2.444875413e-10 | 7.421832845e-12 | 3.208544541e-13 | 2.914642833e-15 |
| 1.0e-12 | 100 | 3200 | 2.444875413e-10 | 7.421832845e-12 | 3.208544541e-13 | 2.914642833e-15 |
| 1.0e-04 | 1000 | 3200 | 9.533474010e-11 | 2.278563203e-12 | 3.363975765e-13 | 7.949058430e-16 |
| 1.0e-06 | 1000 | 3200 | 9.533474010e-11 | 2.278563203e-12 | 3.363975765e-13 | 7.949058430e-16 |
| 1.0e-08 | 1000 | 3200 | 9.533474010e-11 | 2.278563203e-12 | 3.363975765e-13 | 7.949058430e-16 |
| 1.0e-10 | 1000 | 3200 | 9.533474010e-11 | 2.278563203e-12 | 3.363975765e-13 | 7.949058430e-16 |
| 1.0e-12 | 1000 | 3200 | 9.533474010e-11 | 2.278563203e-12 | 3.363975765e-13 | 7.949058430e-16 |
| 1.0e-04 | 10 | 12544 | 5.467366992e-08 | 9.679723537e-10 | 2.360178719e-11 | 1.811519130e-13 |
| 1.0e-06 | 10 | 12544 | 5.467366992e-08 | 9.679723537e-10 | 2.360178719e-11 | 1.811519130e-13 |
| 1.0e-08 | 10 | 12544 | 5.467366992e-08 | 9.679723537e-10 | 2.360178719e-11 | 1.811519130e-13 |
| 1.0e-10 | 10 | 12544 | 5.467366992e-08 | 9.679723537e-10 | 2.360178719e-11 | 1.811519130e-13 |
| 1.0e-12 | 10 | 12544 | 5.467366992e-08 | 9.679723537e-10 | 2.360178719e-11 | 1.811519130e-13 |
| 1.0e-04 | 100 | 12544 | 5.851620410e-09 | 1.017117592e-10 | 2.491340467e-12 | 1.798408788e-14 |
| 1.0e-06 | 100 | 12544 | 5.851620410e-09 | 1.017117592e-10 | 2.491340467e-12 | 1.798408788e-14 |
| 1.0e-08 | 100 | 12544 | 5.851620410e-09 | 1.017117592e-10 | 2.491340467e-12 | 1.798408788e-14 |
| 1.0e-10 | 100 | 12544 | 5.851620410e-09 | 1.017117592e-10 | 2.491340467e-12 | 1.798408788e-14 |
| 1.0e-12 | 100 | 12544 | 5.851620410e-09 | 1.017117592e-10 | 2.491340467e-12 | 1.798408788e-14 |
| 1.0e-04 | 1000 | 12544 | 1.018606421e-09 | 1.574565264e-11 | 1.077360423e-12 | 1.961803286e-15 |
| 1.0e-06 | 1000 | 12544 | 1.018606421e-09 | 1.574565264e-11 | 1.077360423e-12 | 1.961803286e-15 |
| 1.0e-08 | 1000 | 12544 | 1.018606421e-09 | 1.574565264e-11 | 1.077360423e-12 | 1.961803286e-15 |
| 1.0e-10 | 1000 | 12544 | 1.018606421e-09 | 1.574565264e-11 | 1.077360423e-12 | 1.961803286e-15 |
| 1.0e-12 | 1000 | 12544 | 1.018606421e-09 | 1.574565264e-11 | 1.077360423e-12 | 1.961803286e-15 |

Πίνακας 4.5: Νόρμες σφαλμάτων και νόρμαςπολοίπου της CG,για τη μέθοδο Projected CG

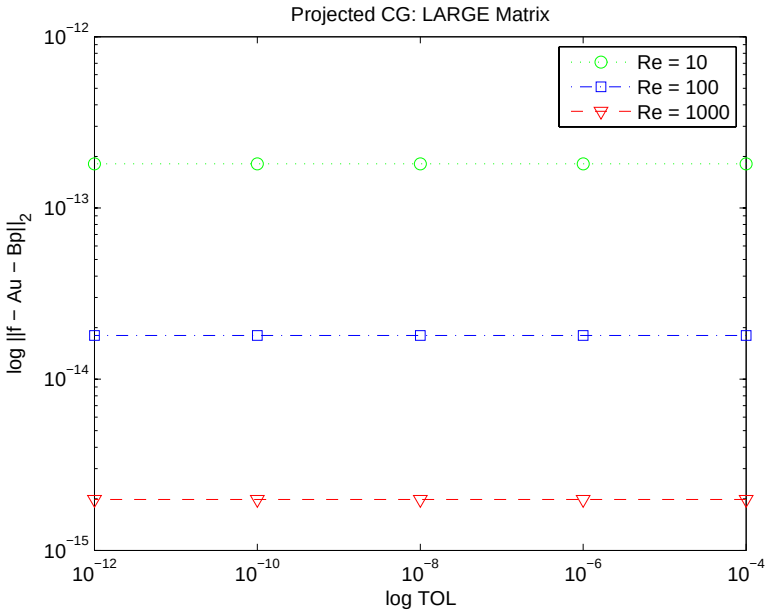
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο Projected CG:



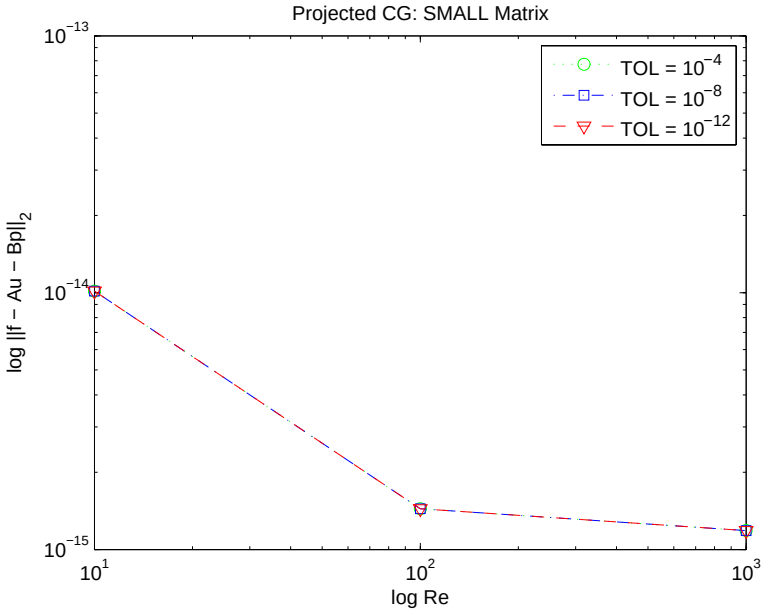

Σχήμα 4.7: Μέθοδος Projected CG: TOL vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



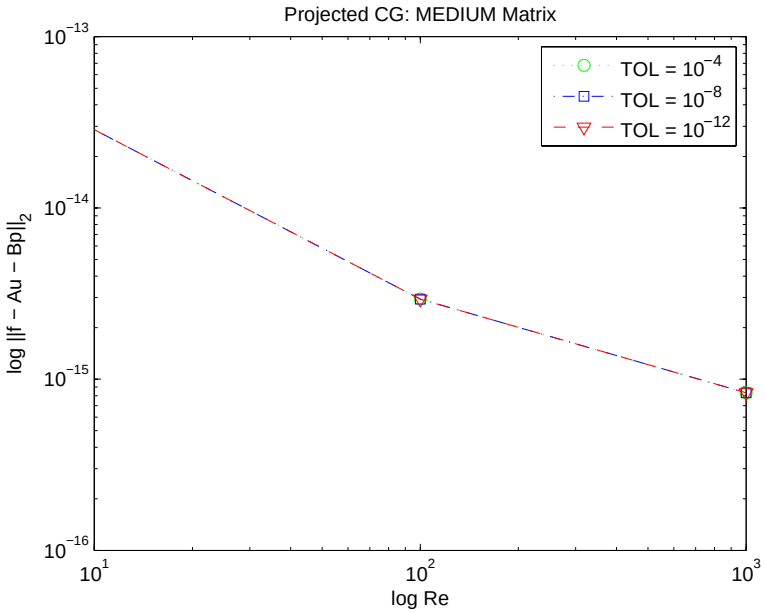

Σχήμα 4.8: Μέθοδος Projected CG: TOL vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



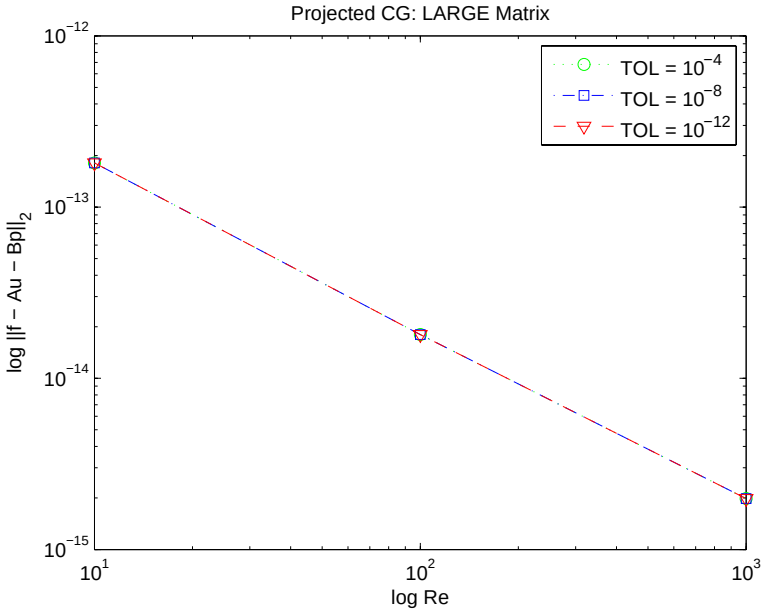

Σχήμα 4.9: Μέθοδος Projected CG: TOL vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



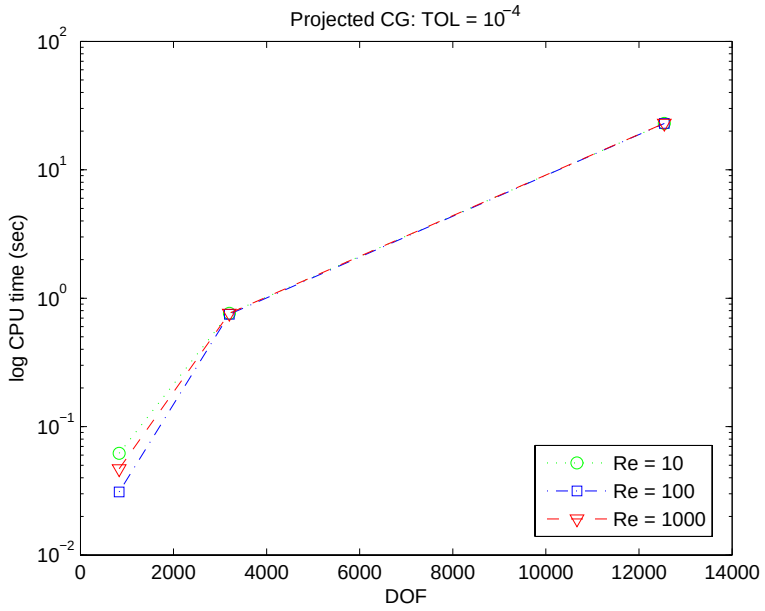

Σχήμα 4.10: Μέθοδος Projected CG: Re vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



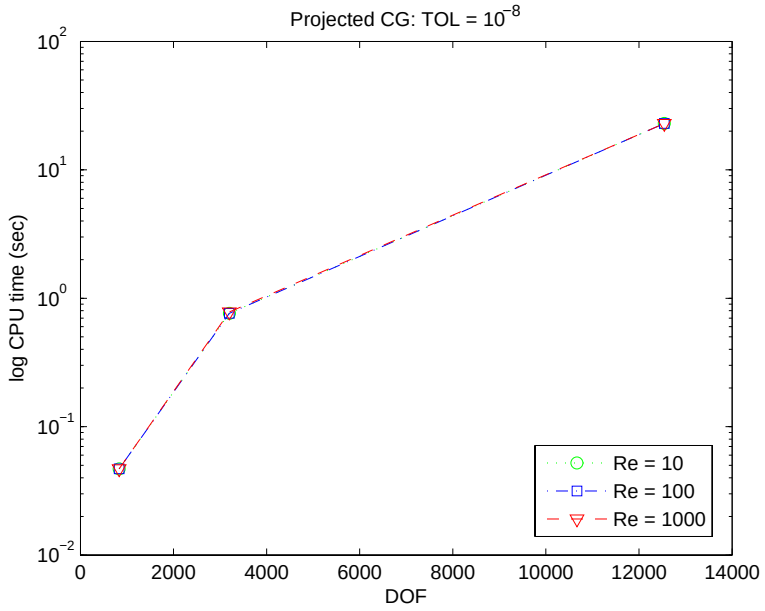

Σχήμα 4.11: Μέθοδος Projected CG: Re vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



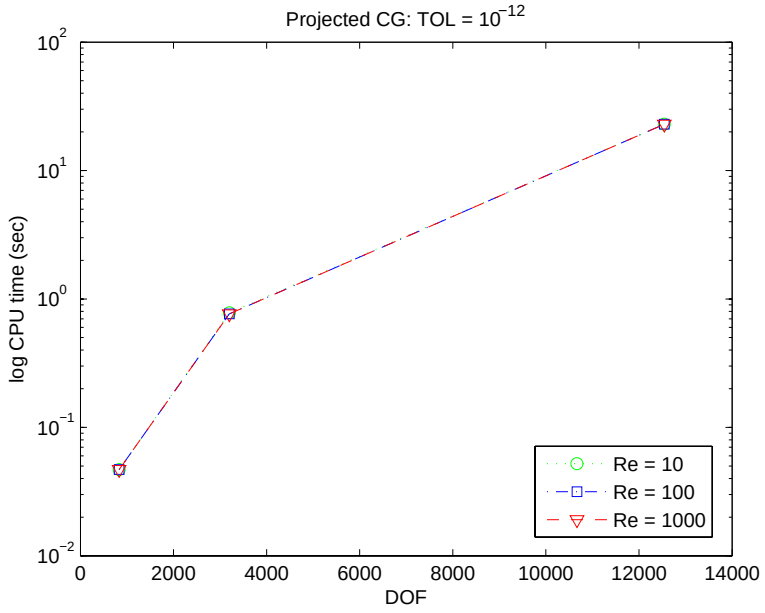

Σχήμα 4.12: Μέθοδος Projected CG: Re vs $\|f - Au - Bp\|_2$ (LARGE Matrix).




Σχήμα 4.13: Μέθοδος Projected CG: DOF vs CPU time, για TOL $= 10^{-4}$.




Σχήμα 4.14: Μέθοδος Projected CG: DOF vs CPU time, για TOL $= 10^{-8}$.




Σχήμα 4.15: Μέθοδος Projected CG: DOF vs CPU time, για TOL $= 10^{-12}$.

### 4.3.2 Αποτελέσματα Μεθόδου Uzawa

Στους πίνακες (4.6-4.7), παραθέτουμε τα αποτελέσματα για τη μέθοδο Uzawa. Στα σχήματα (4.16)–(4.36) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου Uzawa. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου Uzawa:

- Σε όλες τις κατηγορίες διαστάσεων των πινάκων και για όλες τις τιμές της ανοχής TOL που δοκιμάστηκαν, όσο αυξάνει ο αριθμός Reynolds της ροής, μειώνεται η υπολογιζόμενη ευκλείδια νόρμα $\|f - Au - Bp\|_2$, ενώ η ευκλείδια νόρμα $\|g - B^T u\|_2$, παραμένει σχεδόν αμετάβλητη (βλέπε σχήματα (4.25) – (4.27) *Re vs* $\|f - Au - Bp\|_2$),(4.16) – (4.18) *TOL vs* $\|f - Au - Bp\|_2$, (4.28) – (4.30) *Re vs* $\|g - B^T u\|_2$).

- Σε όλες τις κατηγορίες διαστάσεων των πινάκων,και για όλες τις τιμές της ανοχής TOL που δοκιμάστηκαν, όσο αυξάνει ο αριθμός Reynolds της ροής, αυξάνει και ο απαιτούμενος υπολογιστικός χρόνος (βλέπε σχήματα (4.31) – (4.33) Re vs CPU ).

- Καθώς αυξάνεται η διάσταση του γραμμικού συστήματος, ο απαιτούμενος υπολογιστικός χρόνος αυξάνεται θεαματικά, για όλες τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής TOL, που δοκιμάστηκαν (βλέπε σχήματα (4.34) – (4.36) DOF vs CPU ). *Είναι άξιο αναφοράς το γεγονός ότι για τον μεγάλο πίνακα που δοκιμάστηκε κάποια runs χρειάστηκαν πάνω από* 9.5 *ώρες καθαρού υπολογιστικού χρόνου, το καθένα!* (βλέπε πίνακα 4.6).

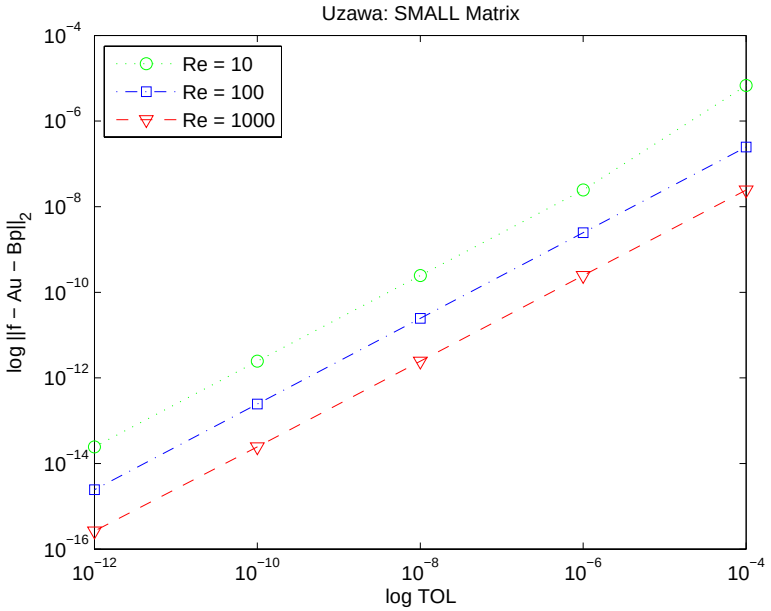
| TOL | Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | #iter | CPU time (sec) |
|---|---|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 6.777894710e-06 | 1.002702929e-04 | 1.004991119e-04 | 1210 | 0.733000 |
| 1.0e-06 | 10 | 832 | 2.451799046e-08 | 1.003263334e-06 | 1.003562878e-06 | 4313 | 4.618000 |
| 1.0e-08 | 10 | 832 | 2.457191503e-10 | 1.009239522e-08 | 1.009538603e-08 | 7480 | 10.858000 |
| 1.0e-10 | 10 | 832 | 2.448822256e-12 | 1.003011920e-10 | 1.003310812e-10 | 10632 | 18.735000 |
| 1.0e-12 | 10 | 832 | 2.450433213e-14 | 1.003047182e-12 | 1.003346456e-12 | 13782 | 27.956000 |
| 1.0e-04 | 100 | 832 | 2.463985522e-07 | 1.002936930e-04 | 1.002939957e-04 | 2760 | 1.716000 |
| 1.0e-06 | 100 | 832 | 2.465137212e-09 | 1.010860098e-06 | 1.010863104e-06 | 5900 | 6.864000 |
| 1.0e-08 | 100 | 832 | 2.454858542e-11 | 1.008266739e-08 | 1.008269727e-08 | 9054 | 13.915000 |
| 1.0e-10 | 100 | 832 | 2.450756569e-13 | 1.005381531e-10 | 1.005384518e-10 | 12206 | 22.339000 |
| 1.0e-12 | 100 | 832 | 2.449201967e-15 | 1.005489077e-12 | 1.005492060e-12 | 15356 | 31.981000 |
| 1.0e-04 | 1000 | 832 | 2.457294326e-08 | 1.006654492e-04 | 1.006654522e-04 | 4321 | 3.120000 |
| 1.0e-06 | 1000 | 832 | 2.457851467e-10 | 1.009274166e-06 | 1.009274196e-06 | 7480 | 9.422000 |
| 1.0e-08 | 1000 | 832 | 2.448813595e-12 | 1.003018169e-08 | 1.003018199e-08 | 10632 | 17.051000 |
| 1.0e-10 | 1000 | 832 | 2.449143267e-14 | 1.002934722e-10 | 1.002934752e-10 | 13782 | 26.052000 |
| 1.0e-12 | 1000 | 832 | 2.636728096e-16 | 1.014611557e-12 | 1.014611591e-12 | 16928 | 36.130000 |
| 1.0e-04 | 10 | 3200 | 2.011378690e-05 | 9.994160141e-05 | 1.019455154e-04 | 2084 | 3.401000 |
| 1.0e-06 | 10 | 3200 | 2.212976808e-08 | 1.002641248e-06 | 1.002885436e-06 | 12721 | 66.503000 |
| 1.0e-08 | 10 | 3200 | 2.201470654e-10 | 1.000517570e-08 | 1.000759739e-08 | 28071 | 257.776000 |
| 1.0e-10 | 10 | 3200 | 2.202560694e-12 | 1.000960004e-10 | 1.001202306e-10 | 43503 | 518.267000 |
| 1.0e-12 | 10 | 3200 | 2.203387915e-14 | 1.000862347e-12 | 1.001104854e-12 | 58883 | 861.968000 |
| 1.0e-04 | 100 | 3200 | 7.950802825e-07 | 1.000312007e-04 | 1.000343604e-04 | 5144 | 13.213000 |
| 1.0e-06 | 100 | 3200 | 2.207573048e-09 | 1.003710647e-06 | 1.003713074e-06 | 20446 | 120.932000 |
| 1.0e-08 | 100 | 3200 | 2.201345324e-11 | 1.000278136e-08 | 1.000280558e-08 | 35757 | 345.807000 |
| 1.0e-10 | 100 | 3200 | 2.203159552e-13 | 1.001054951e-10 | 1.001057375e-10 | 51217 | 646.780000 |
| 1.0e-12 | 100 | 3200 | 2.216561925e-15 | 1.003185457e-12 | 1.003187906e-12 | 66590 | 1016.893000 |
| 1.0e-04 | 1000 | 3200 | 2.211359417e-08 | 1.000827439e-04 | 1.000827463e-04 | 12732 | 33.244000 |
| 1.0e-06 | 1000 | 3200 | 2.200755823e-10 | 1.000341536e-06 | 1.000341560e-06 | 28085 | 181.195000 |
| 1.0e-08 | 1000 | 3200 | 2.202554564e-12 | 1.000963704e-08 | 1.000963728e-08 | 43503 | 453.011000 |
| 1.0e-10 | 1000 | 3200 | 2.201730596e-14 | 1.000835693e-10 | 1.000835717e-10 | 58881 | 776.916000 |
| 1.0e-12 | 1000 | 3200 | 2.469124943e-16 | 1.013010298e-12 | 1.013010328e-12 | 74280 | 1168.183000 |
| 1.0e-04 | 10 | 12544 | 4.612854118e-05 | 9.997511913e-05 | 1.101038907e-04 | 654 | 17.004000 |
| 1.0e-06 | 10 | 12544 | 8.566217761e-08 | 1.000045697e-06 | 1.003707829e-06 | 21790 | 699.352000 |
| 1.0e-08 | 10 | 12544 | 2.019779845e-10 | 1.001754716e-08 | 1.001958314e-08 | 94036 | 5131.840000 |
| 1.0e-10 | 10 | 12544 | 2.019209163e-12 | 1.000449066e-10 | 1.000652814e-10 | 166448 | 13122.929000 |
| 1.0e-12 | 10 | 12544 | 2.031866257e-14 | 1.000181004e-12 | 1.000387369e-12 | 239046 | 24114.915000 |
| 1.0e-04 | 100 | 12544 | 2.288140835e-06 | 9.997111802e-05 | 9.999730010e-05 | 9239 | 42.994000 |
| 1.0e-06 | 100 | 12544 | 2.017737244e-09 | 1.000771411e-06 | 1.000773445e-06 | 57588 | 1730.987000 |
| 1.0e-08 | 100 | 12544 | 2.020005043e-11 | 1.001093584e-08 | 1.001095622e-08 | 130174 | 7512.884000 |
| 1.0e-10 | 100 | 12544 | 2.018836064e-13 | 1.000114786e-10 | 1.000116823e-10 | 202810 | 16937.293000 |
| 1.0e-12 | 100 | 12544 | 2.030392789e-15 | 1.002835123e-12 | 1.002837178e-12 | 275390 | 29213.184000 |
| 1.0e-04 | 1000 | 12544 | 8.565047412e-08 | 9.998544250e-05 | 9.998547918e-05 | 21783 | 217.809000 |
| 1.0e-06 | 1000 | 12544 | 2.015674211e-10 | 1.000743181e-06 | 1.000743201e-06 | 94350 | 3234.821000 |
| 1.0e-08 | 1000 | 12544 | 2.019098996e-12 | 1.000382872e-08 | 1.000382893e-08 | 166440 | 10293.165000 |
| 1.0e-10 | 1000 | 12544 | 2.018655301e-14 | 1.000002453e-10 | 1.000002473e-10 | 239148 | 21205.527000 |
| 1.0e-12 | 1000 | 12544 | 2.297681081e-16 | 1.009542694e-12 | 1.009542721e-12 | 311681 | 34635.202000 |

Πίνακας 4.6: Αποτελέσματα για τη μέθοδο Uzawa

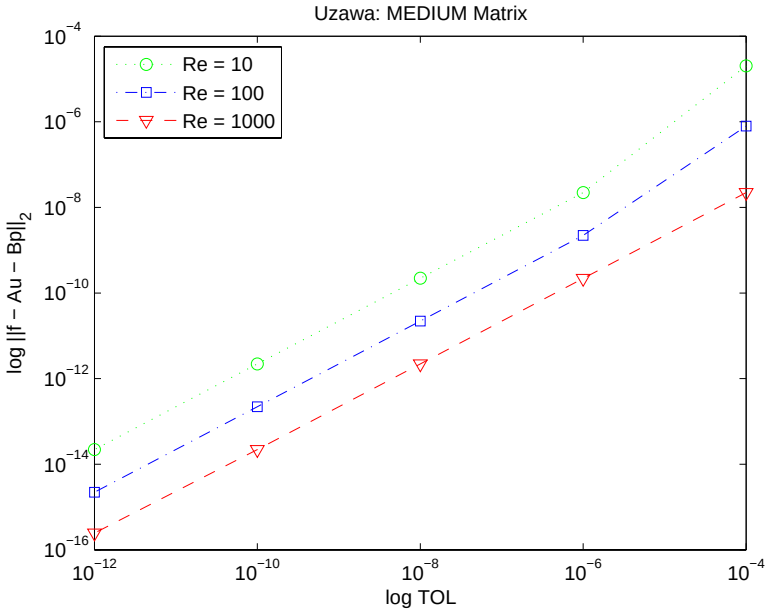
| TOL | Re | DOF | $\|x-x^*\|_1$ | $\|x-x^*\|_2$ | $\|x-x^*\|_\infty$ |
|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 9.213088453e+00 | 6.034756020e-01 | 5.169053623e-02 |
| 1.0e-06 | 10 | 832 | 9.858689804e-02 | 6.298288519e-03 | 5.155711921e-04 |
| 1.0e-08 | 10 | 832 | 9.893629180e-04 | 6.320605322e-05 | 5.174005657e-06 |
| 1.0e-10 | 10 | 832 | 9.859264779e-06 | 6.298639242e-07 | 5.156030858e-08 |
| 1.0e-12 | 10 | 832 | 9.861244676e-08 | 6.299903731e-09 | 5.157063665e-10 |
| 1.0e-04 | 100 | 832 | 1.107272025e+00 | 6.425087774e-02 | 6.579226870e-03 |
| 1.0e-06 | 100 | 832 | 1.112634097e-02 | 6.455200732e-04 | 6.606214198e-05 |
| 1.0e-08 | 100 | 832 | 1.110997026e-04 | 6.445656286e-06 | 6.597015638e-07 |
| 1.0e-10 | 100 | 832 | 1.109043400e-06 | 6.434282066e-08 | 6.585509693e-09 |
| 1.0e-12 | 100 | 832 | 1.109659120e-08 | 6.437825764e-10 | 6.589173651e-11 |
| 1.0e-04 | 1000 | 832 | 2.329667238e-01 | 1.426777699e-02 | 6.594422679e-03 |
| 1.0e-06 | 1000 | 832 | 2.330761026e-03 | 1.427459686e-04 | 6.597917612e-05 |
| 1.0e-08 | 1000 | 832 | 2.323157876e-05 | 1.422686111e-06 | 6.576325066e-07 |
| 1.0e-10 | 1000 | 832 | 2.323581716e-07 | 1.422938097e-08 | 6.577493106e-09 |
| 1.0e-12 | 1000 | 832 | 2.336771776e-09 | 1.431038372e-10 | 6.614830905e-11 |
| 1.0e-04 | 10 | 3200 | 1.164981074e+02 | 4.283455070e+00 | 1.872483464e-01 |
| 1.0e-06 | 10 | 3200 | 3.874442818e+00 | 1.232166492e-01 | 4.789136036e-03 |
| 1.0e-08 | 10 | 3200 | 3.871197543e-02 | 1.231094755e-03 | 4.784391134e-05 |
| 1.0e-10 | 10 | 3200 | 3.872190741e-04 | 1.231410599e-05 | 4.785618359e-07 |
| 1.0e-12 | 10 | 3200 | 3.871205762e-06 | 1.231097395e-07 | 4.784402652e-09 |
| 1.0e-04 | 100 | 3200 | 3.694404718e+01 | 1.168013318e+00 | 4.772307133e-02 |
| 1.0e-06 | 100 | 3200 | 3.978421421e-01 | 1.234130747e-02 | 4.791231640e-04 |
| 1.0e-08 | 100 | 3200 | 3.971518635e-03 | 1.231989019e-04 | 4.782914630e-06 |
| 1.0e-10 | 100 | 3200 | 3.973984522e-05 | 1.232753913e-06 | 4.785884111e-08 |
| 1.0e-12 | 100 | 3200 | 3.977882972e-07 | 1.233962643e-08 | 4.790585706e-10 |
| 1.0e-04 | 1000 | 3200 | 4.988650843e+00 | 1.353984745e-01 | 2.611865994e-02 |
| 1.0e-06 | 1000 | 3200 | 4.987270973e-02 | 1.353516821e-03 | 2.610450832e-04 |
| 1.0e-08 | 1000 | 3200 | 4.989507036e-04 | 1.354120824e-05 | 2.611608031e-06 |
| 1.0e-10 | 1000 | 3200 | 4.989231290e-06 | 1.354049880e-07 | 2.611481575e-08 |
| 1.0e-12 | 1000 | 3200 | 5.018127136e-08 | 1.361876286e-09 | 2.626558970e-10 |
| 1.0e-04 | 10 | 12544 | 3.454733922e+03 | 5.389571360e+01 | 1.132605809e+00 |
| 1.0e-06 | 10 | 12544 | 1.370125895e+02 | 2.198148231e+00 | 4.334770767e-02 |
| 1.0e-08 | 10 | 12544 | 1.467735241e+00 | 2.322422691e-02 | 4.358239634e-04 |
| 1.0e-10 | 10 | 12544 | 1.466753011e-02 | 2.320868476e-04 | 4.355322571e-06 |
| 1.0e-12 | 10 | 12544 | 1.466564468e-04 | 2.320570142e-06 | 4.354762695e-08 |
| 1.0e-04 | 100 | 12544 | 3.827210483e+02 | 7.119045621e+00 | 1.564825238e-01 |
| 1.0e-06 | 100 | 12544 | 1.475298093e+01 | 2.321394976e-01 | 4.356208330e-03 |
| 1.0e-08 | 100 | 12544 | 1.475500882e-01 | 2.321699504e-03 | 4.356642703e-05 |
| 1.0e-10 | 100 | 12544 | 1.474753057e-03 | 2.320522820e-05 | 4.354434696e-07 |
| 1.0e-12 | 100 | 12544 | 1.476807294e-05 | 2.323755280e-07 | 4.360500627e-09 |
| 1.0e-04 | 1000 | 12544 | 1.456181827e+02 | 2.212642801e+00 | 1.086564140e-01 |
| 1.0e-06 | 1000 | 12544 | 1.558259731e+00 | 2.334010694e-02 | 1.037130729e-03 |
| 1.0e-08 | 1000 | 12544 | 1.557953173e-02 | 2.333550577e-04 | 1.036925339e-05 |
| 1.0e-10 | 1000 | 12544 | 1.557656452e-04 | 2.333106332e-06 | 1.036728224e-07 |
| 1.0e-12 | 1000 | 12544 | 1.565120743e-06 | 2.344286601e-08 | 1.041696174e-09 |

Πίνακας 4.7: Νόρμες σφαλμάτων για τη μέθοδο Uzawa

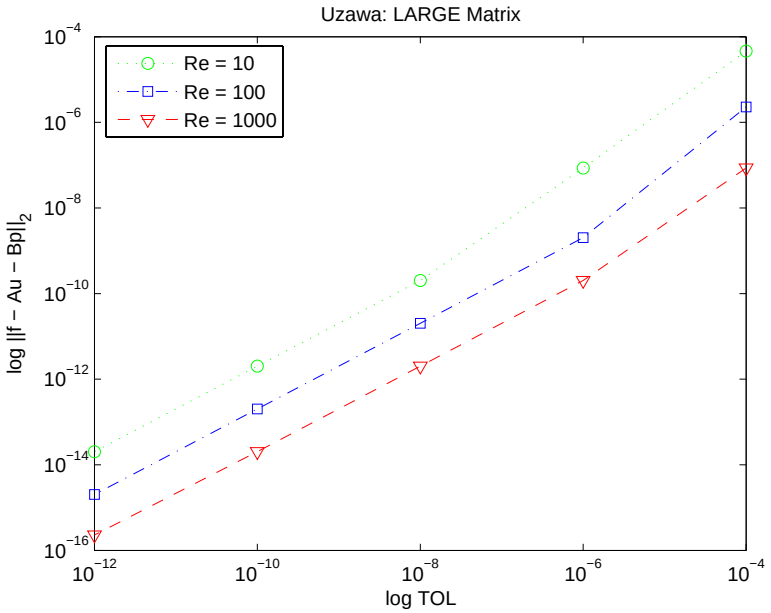
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο Uzawa:



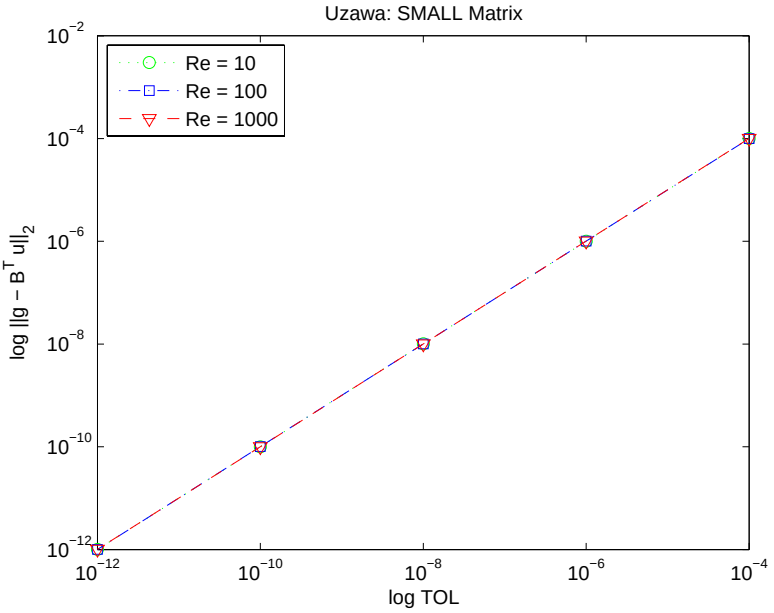

Σχήμα 4.16: Μέθοδος Uzawa: TOL vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



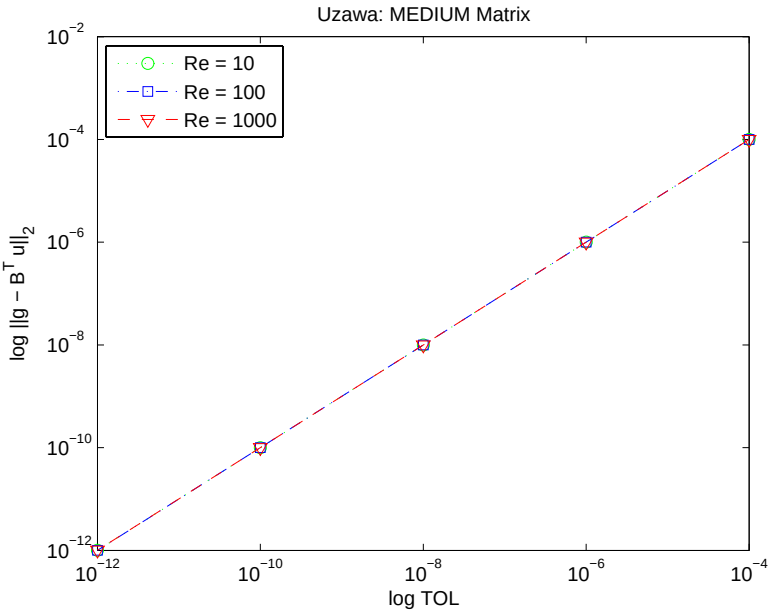

Σχήμα 4.17: Μέθοδος Uzawa: TOL vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



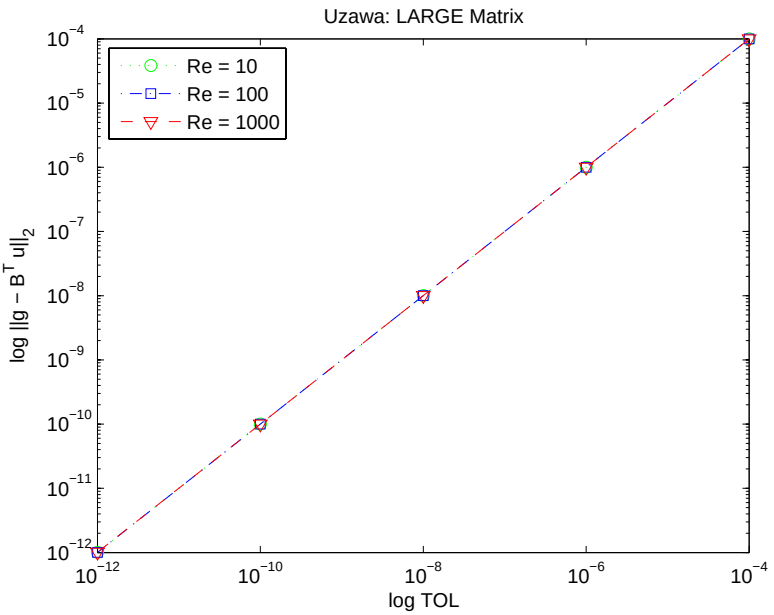

Σχήμα 4.18: Μέθοδος Uzawa: TOL vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



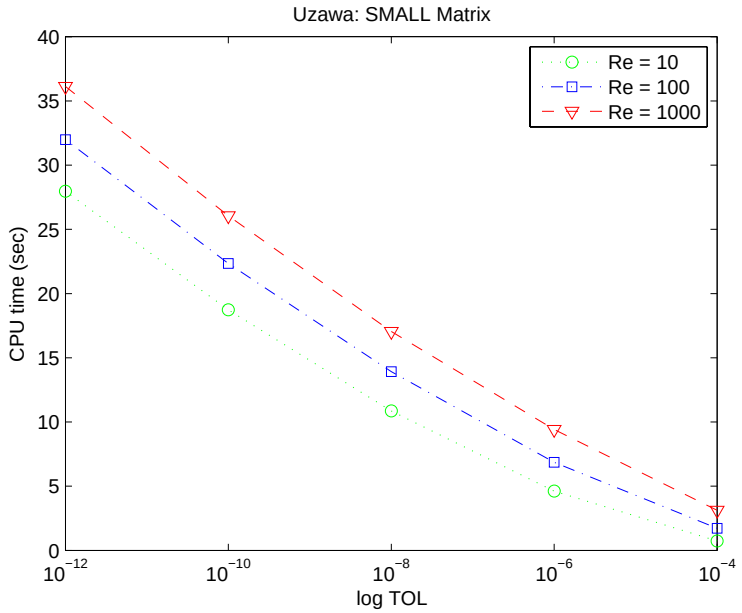

Σχήμα 4.19: Μέθοδος Uzawa: TOL vs $\|g - B^T u\|_2$ (SMALL Matrix).



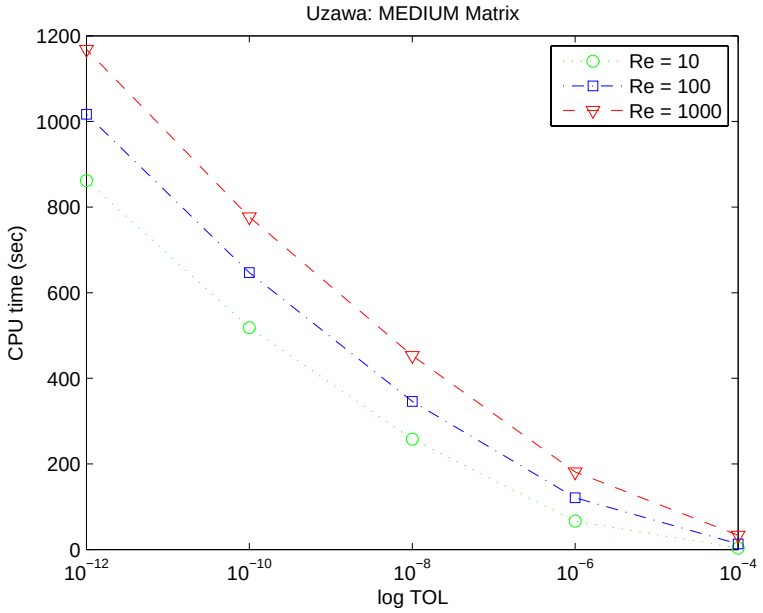

Σχήμα 4.20: Μέθοδος Uzawa: TOL vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



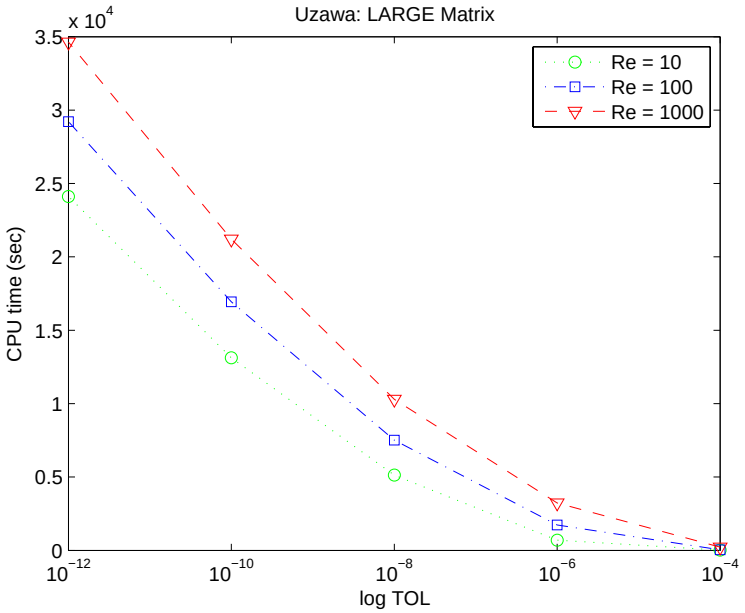

Σχήμα 4.21: Μέθοδος Uzawa: TOL vs $\|g - B^T u\|_2$ (LARGE Matrix).



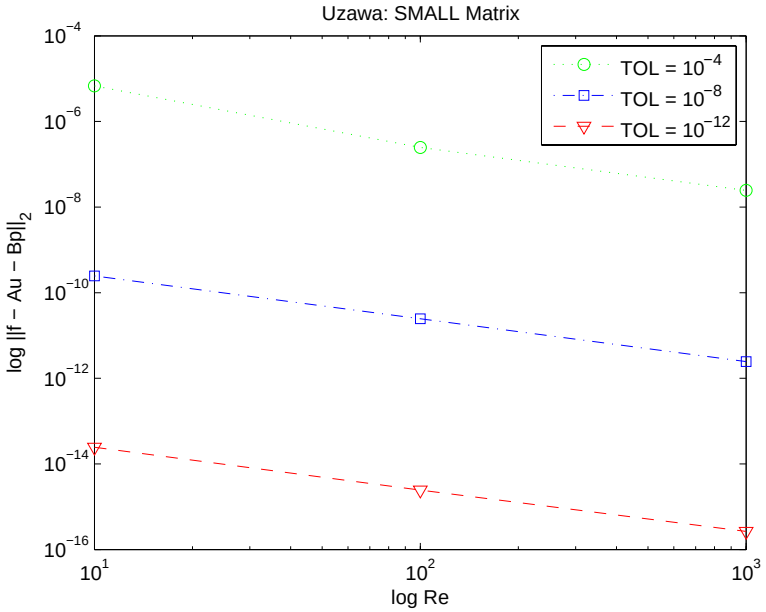

Σχήμα 4.22: Μέθοδος Uzawa: TOL vs CPU time, (SMALL Matrix).



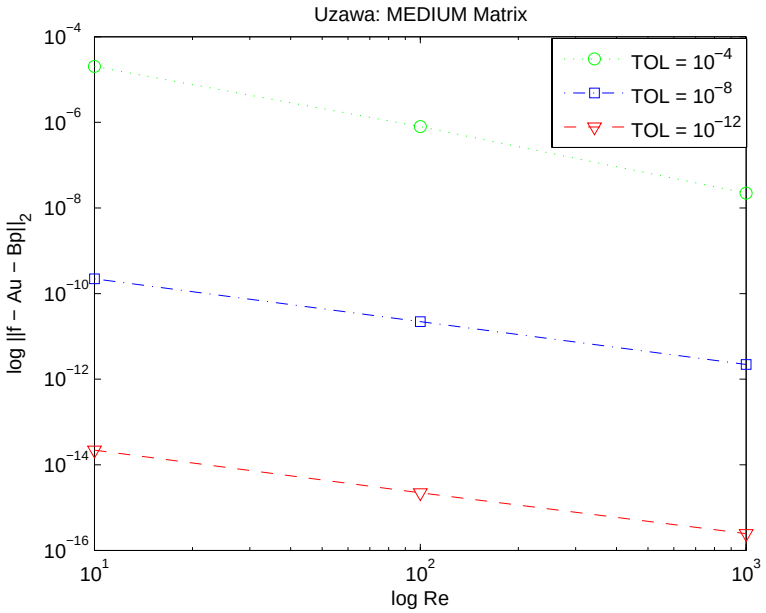

Σχήμα 4.23: Μέθοδος Uzawa: TOL vs CPU time, (MEDIUM Matrix).



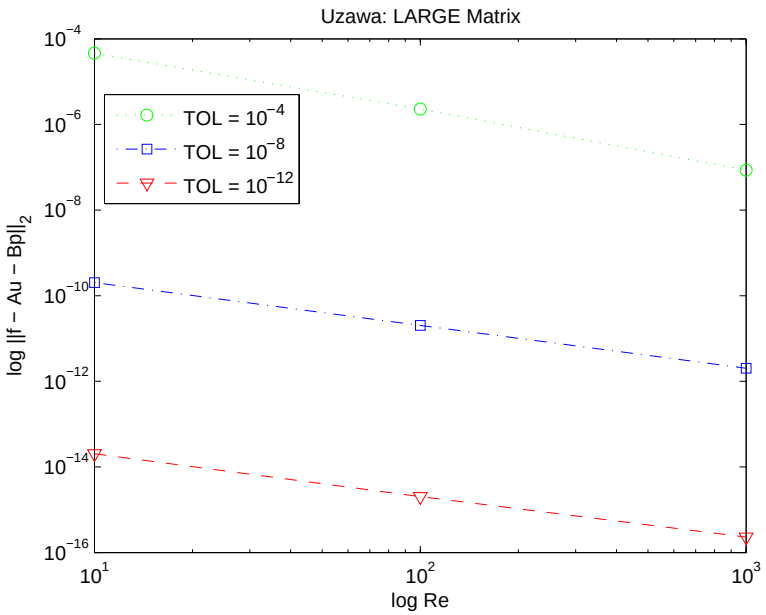

Σχήμα 4.24: Μέθοδος Uzawa: TOL vs CPU time, (LARGE Matrix).



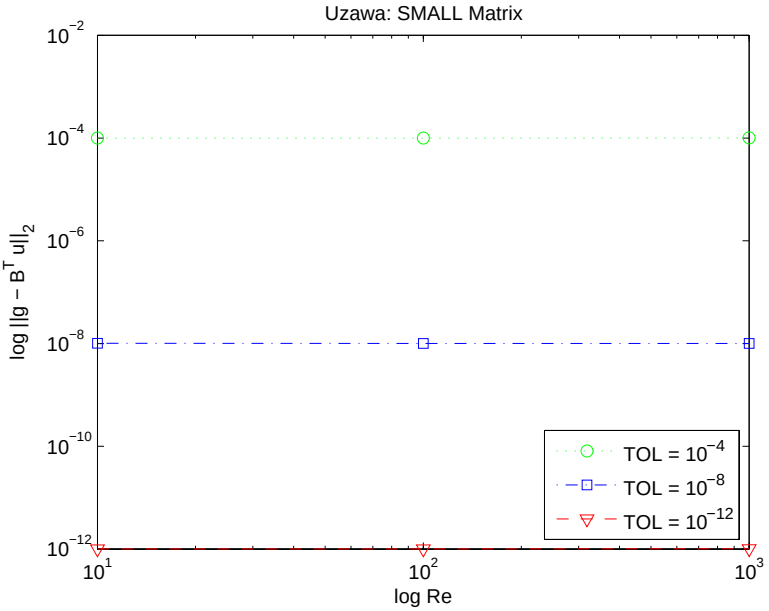

Σχήμα 4.25: Μέθοδος Uzawa: Re vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



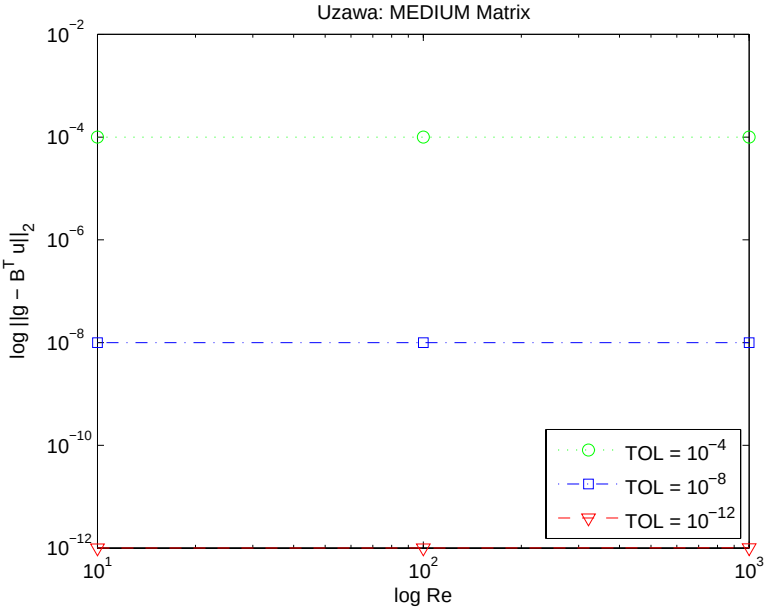

Σχήμα 4.26: Μέθοδος Uzawa: Re vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



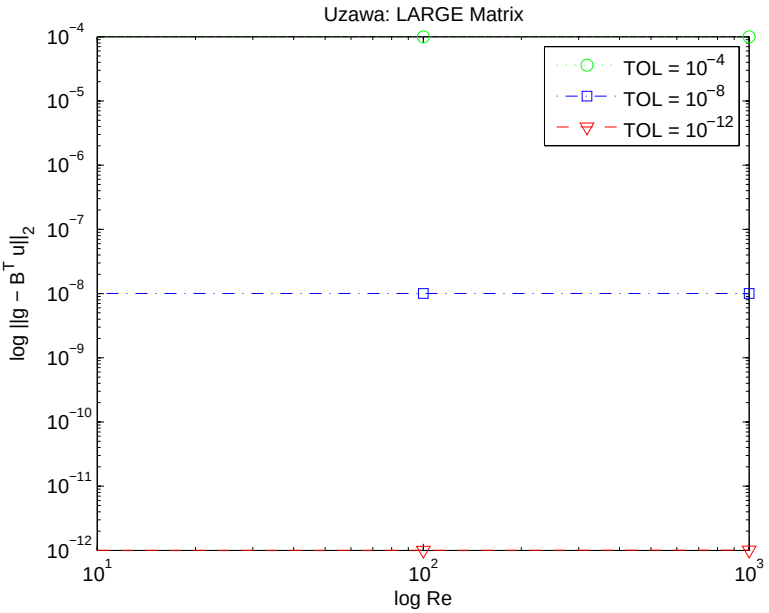

Σχήμα 4.27: Μέθοδος Uzawa: Re vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



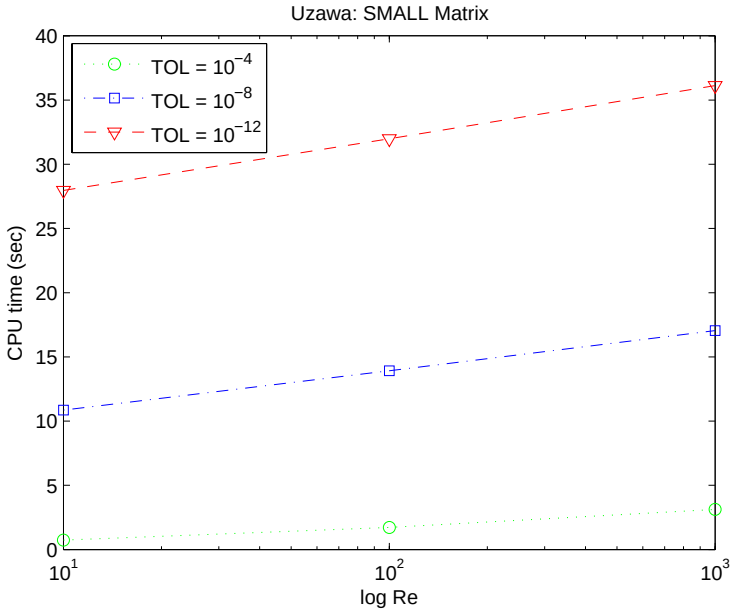

Σχήμα 4.28: Μέθοδος Uzawa: Re vs $\|g - B^T u\|_2$ (SMALL Matrix).



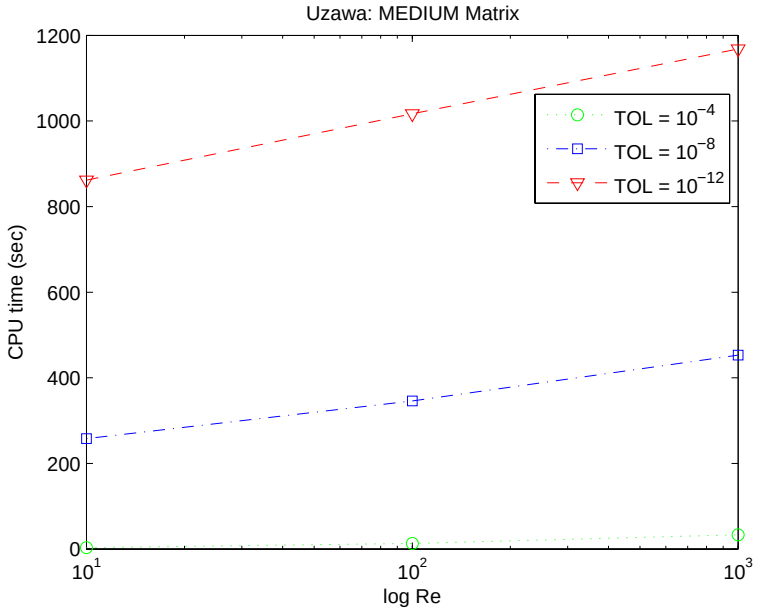

Σχήμα 4.29: Μέθοδος Uzawa: Re vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



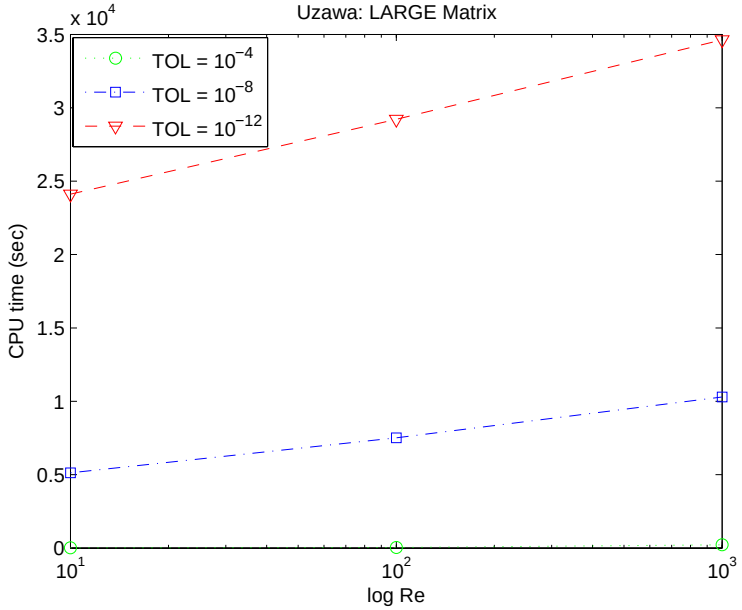

Σχήμα 4.30: Μέθοδος Uzawa: Re vs $\|g - B^T u\|_2$ (LARGE Matrix).



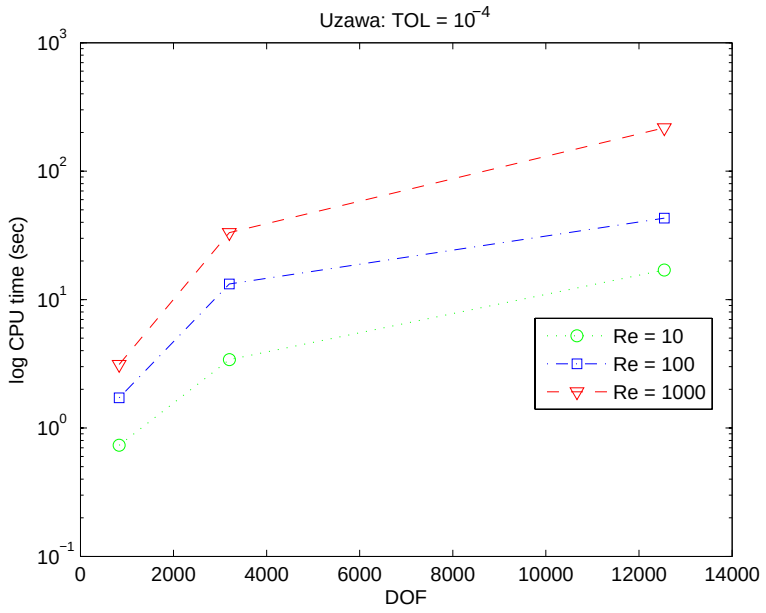

Σχήμα 4.31: Μέθοδος Uzawa: Re vs CPU time, (SMALL Matrix).



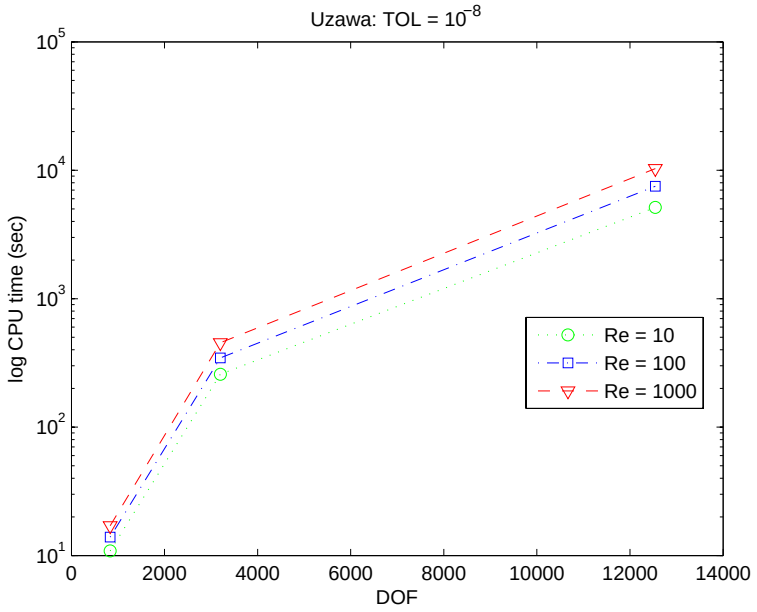

Σχήμα 4.32: Μέθοδος Uzawa: Re vs CPU time, (MEDIUM Matrix).



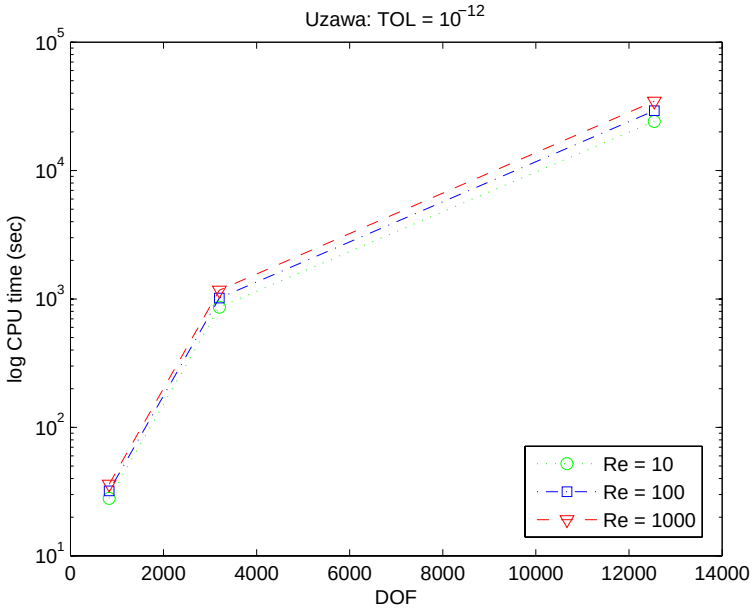

Σχήμα 4.33: Μέθοδος Uzawa: Re vs CPU time, (LARGE Matrix).




Σχήμα 4.34: Μέθοδος Uzawa: DOF vs CPU time, για TOL $= 10^{-4}$.




Σχήμα 4.35: Μέθοδος Uzawa: DOF vs CPU time, για TOL $= 10^{-8}$.




Σχήμα 4.36: Μέθοδος Uzawa: DOF vs CPU time, για TOL $= 10^{-12}$.

### 4.3.3 Αποτελέσματα Μεθόδου Augmented Lagrangian

Στους πίνακες (4.8-4.9), παραθέτουμε τα αποτελέσματα για τη μέθοδο Augmented Lagrangian. Στα σχήματα (4.37)–(4.57) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου Augmented Lagrangian. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου Augmented Lagrangian:

- Σε όλες τις κατηγορίες διαστάσεων των πινάκων και για όλες τις τιμές της ανοχής TOL που δοκιμάστηκαν (με εξαίρεση την τιμή TOL $= 10^{-6}$ στο μικρό πίνακα), όσο αυξάνει ο αριθμός Reynolds της ροής, μειώνεται ο απαιτούμενος υπολογιστικός χρόνος (βλέπε σχήματα (4.43) – (4.45) TOL vs CPU,(4.52) – (4.54) Re vs CPU,(4.55) – (4.57) DOF vs CPU).

- Καθώς αυξάνεται η διάσταση του γραμμικού συστήματος, ο απαιτούμενος υπολογιστικός χρόνος αυξάνεται θεαματικά, για όλες τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής TOL, που δοκιμάστηκαν (βλέπε σχήματα (4.55) – (4.57) DOF vs CPU).

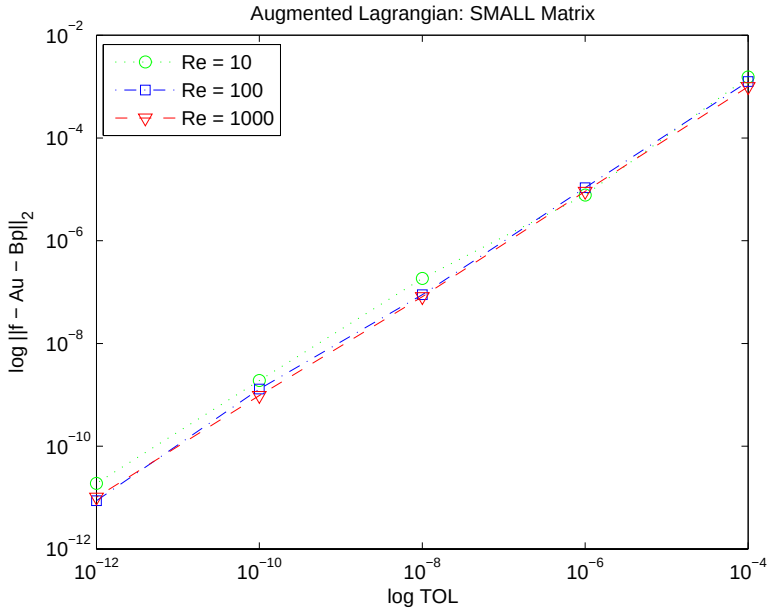
| TOL | Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | #iter | CPU time (sec) |
|---|---|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 1.536340585e-03 | 1.048607203e-04 | 1.539914986e-03 | 86 | 0.328000 |
| 1.0e-06 | 10 | 832 | 7.734857365e-06 | 6.160641425e-07 | 7.759352645e-06 | 274 | 1.528000 |
| 1.0e-08 | 10 | 832 | 1.853179028e-07 | 9.768797561e-09 | 1.855751990e-07 | 457 | 3.635000 |
| 1.0e-10 | 10 | 832 | 1.898461964e-09 | 9.911176522e-11 | 1.901047335e-09 | 647 | 6.739000 |
| 1.0e-12 | 10 | 832 | 1.895482066e-11 | 9.823360944e-13 | 1.898025844e-11 | 853 | 10.670000 |
| 1.0e-04 | 100 | 832 | 1.255508151e-03 | 9.333710013e-05 | 1.258972808e-03 | 28 | 0.266000 |
| 1.0e-06 | 100 | 832 | 1.077974158e-05 | 6.868375962e-07 | 1.080160055e-05 | 43 | 0.624000 |
| 1.0e-08 | 100 | 832 | 8.896344715e-08 | 7.419912886e-09 | 8.927233635e-08 | 63 | 1.185000 |
| 1.0e-10 | 100 | 832 | 1.301173575e-09 | 8.839910650e-11 | 1.304172947e-09 | 90 | 2.075000 |
| 1.0e-12 | 100 | 832 | 8.691576630e-12 | 7.628984051e-13 | 8.724993885e-12 | 106 | 2.964000 |
| 1.0e-04 | 1000 | 832 | 1.001104351e-03 | 7.672424851e-05 | 1.004040105e-03 | 22 | 0.265000 |
| 1.0e-06 | 1000 | 832 | 9.143040331e-06 | 8.196544160e-07 | 9.179706960e-06 | 52 | 0.921000 |
| 1.0e-08 | 1000 | 832 | 8.138028630e-08 | 7.110186275e-09 | 8.169030387e-08 | 41 | 0.936000 |
| 1.0e-10 | 1000 | 832 | 9.676390756e-10 | 9.189221562e-11 | 9.719925719e-10 | 23 | 0.795000 |
| 1.0e-12 | 1000 | 832 | 1.019792525e-11 | 7.768944682e-13 | 1.022747498e-11 | 45 | 1.701000 |
| 1.0e-04 | 10 | 3200 | 9.426386799e-04 | 9.943013937e-05 | 9.478681520e-04 | 465 | 6.349000 |
| 1.0e-06 | 10 | 3200 | 8.833105851e-06 | 9.980187237e-07 | 8.889308204e-06 | 2571 | 46.800000 |
| 1.0e-08 | 10 | 3200 | 8.831105375e-08 | 9.977633458e-09 | 8.887291705e-08 | 5589 | 171.789000 |
| 1.0e-10 | 10 | 3200 | 8.826093828e-10 | 9.971976740e-11 | 8.882248334e-10 | 8600 | 390.065000 |
| 1.0e-12 | 10 | 3200 | 8.834801594e-12 | 9.981531252e-13 | 8.891008315e-12 | 11612 | 647.919000 |
| 1.0e-04 | 100 | 3200 | 5.734865792e-04 | 9.176982235e-05 | 5.807827105e-04 | 156 | 5.553000 |
| 1.0e-06 | 100 | 3200 | 6.209024292e-06 | 9.552349986e-07 | 6.282074225e-06 | 537 | 30.670000 |
| 1.0e-08 | 100 | 3200 | 5.967129768e-08 | 9.752165726e-09 | 6.046295150e-08 | 929 | 78.625000 |
| 1.0e-10 | 100 | 3200 | 5.677229083e-10 | 9.557329269e-11 | 5.757113469e-10 | 1320 | 148.170000 |
| 1.0e-12 | 100 | 3200 | 5.332450628e-12 | 9.670488889e-13 | 5.419429237e-12 | 1697 | 244.547000 |
| 1.0e-04 | 1000 | 3200 | 4.151689315e-04 | 8.362197582e-05 | 4.235066428e-04 | 40 | 3.588000 |
| 1.0e-06 | 1000 | 3200 | 4.608912439e-06 | 8.616751201e-07 | 4.688769336e-06 | 77 | 11.778000 |
| 1.0e-08 | 1000 | 3200 | 3.182457381e-08 | 7.299792713e-09 | 3.265104090e-08 | 128 | 28.439000 |
| 1.0e-10 | 1000 | 3200 | 8.545602196e-10 | 9.866468844e-11 | 8.602371125e-10 | 169 | 47.502000 |
| 1.0e-12 | 1000 | 3200 | 4.405641244e-12 | 8.538148351e-13 | 4.487613457e-12 | 214 | 76.144000 |
| 1.0e-04 | 10 | 12544 | 4.538577036e-04 | 9.992932311e-05 | 4.647286141e-04 | 341 | 19.266000 |
| 1.0e-06 | 10 | 12544 | 4.424279947e-06 | 9.998108398e-07 | 4.535843336e-06 | 11045 | 964.554000 |
| 1.0e-08 | 10 | 12544 | 4.416955327e-08 | 9.998763533e-09 | 4.528713623e-08 | 47648 | 6690.244000 |
| 1.0e-10 | 10 | 12544 | 4.416838136e-10 | 9.998385529e-11 | 4.528590978e-10 | 84868 | 18402.720000 |
| 1.0e-12 | 10 | 12544 | 4.416737325e-12 | 9.998574371e-13 | 4.528496825e-12 | 122104 | 36440.976000 |
| 1.0e-04 | 100 | 12544 | 4.411399643e-04 | 1.000273287e-04 | 4.523382966e-04 | 906 | 108.857000 |
| 1.0e-06 | 100 | 12544 | 4.432481448e-06 | 9.990359398e-07 | 4.543673029e-06 | 5295 | 842.998000 |
| 1.0e-08 | 100 | 12544 | 4.433741912e-08 | 9.993192850e-09 | 4.544964948e-08 | 11668 | 3559.288000 |
| 1.0e-10 | 100 | 12544 | 4.431462135e-10 | 9.988046688e-11 | 4.542627810e-10 | 18111 | 7971.152000 |
| 1.0e-12 | 100 | 12544 | 4.433401634e-12 | 9.992338841e-13 | 4.544614219e-12 | 24497 | 14469.529000 |
| 1.0e-04 | 1000 | 12544 | 2.575198340e-04 | 9.809775359e-05 | 2.755714683e-04 | 260 | 92.587000 |
| 1.0e-06 | 1000 | 12544 | 2.857988542e-06 | 9.890004309e-07 | 3.024271873e-06 | 973 | 580.480000 |
| 1.0e-08 | 1000 | 12544 | 2.811456209e-08 | 9.430555343e-09 | 2.965407183e-08 | 1738 | 1631.021000 |
| 1.0e-10 | 1000 | 12544 | 3.158202371e-10 | 3.579603007e-10 | 4.773656869e-10 | 2467 | 3099.600000 |
| 1.0e-12 | 1000 | 12544 | 2.102885311e-12 | 9.615265579e-13 | 2.312284575e-12 | 3199 | 5131.309000 |

Πίνακας 4.8: Αποτελέσματα για τη μέθοδο Augmented Lagrangian

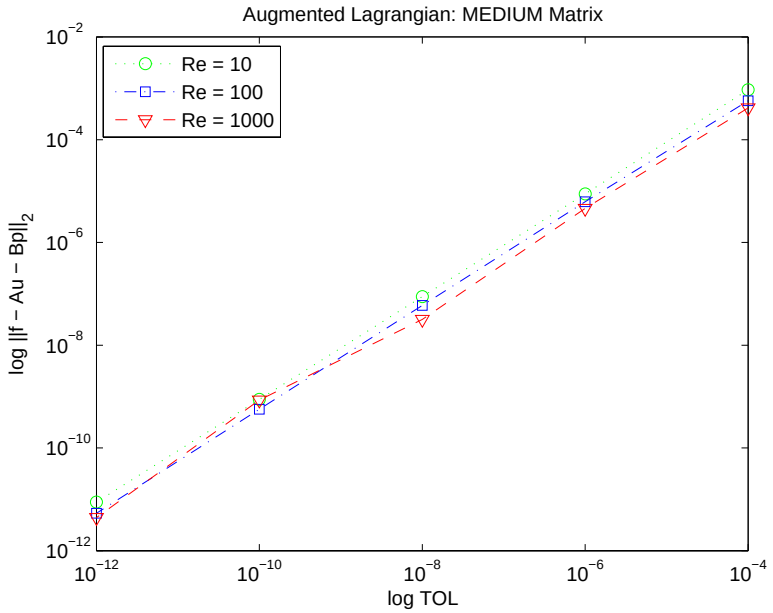
| TOL | Re | DOF | $\|x-x^*\|_1$ | $\|x-x^*\|_2$ | $\|x-x^*\|_\infty$ |
|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 7.857631839e+00 | 4.965156204e-01 | 4.049820576e-02 |
| 1.0e-06 | 10 | 832 | 6.810107286e-02 | 4.301016610e-03 | 3.360576621e-04 |
| 1.0e-08 | 10 | 832 | 7.999635357e-04 | 5.064275380e-05 | 3.985733490e-06 |
| 1.0e-10 | 10 | 832 | 8.074191860e-06 | 5.112472594e-07 | 4.046229374e-08 |
| 1.0e-12 | 10 | 832 | 5.492889787e-08 | 3.435344807e-09 | 2.627793538e-10 |
| 1.0e-04 | 100 | 832 | 7.726836675e-01 | 4.140127675e-02 | 5.099374267e-03 |
| 1.0e-06 | 100 | 832 | 3.433609903e-03 | 1.829319864e-04 | 3.660429003e-05 |
| 1.0e-08 | 100 | 832 | 4.101803746e-05 | 2.133065358e-06 | 3.012566951e-07 |
| 1.0e-10 | 100 | 832 | 6.445251390e-07 | 3.750075211e-08 | 5.531159752e-09 |
| 1.0e-12 | 100 | 832 | 3.449096120e-09 | 1.796687360e-10 | 2.582467573e-11 |
| 1.0e-04 | 1000 | 832 | 6.283572882e+00 | 3.730737079e-01 | 4.520854474e-02 |
| 1.0e-06 | 1000 | 832 | 3.425291257e-02 | 1.940272005e-03 | 2.358714186e-04 |
| 1.0e-08 | 1000 | 832 | 6.483597601e-04 | 3.955160196e-05 | 5.479228448e-06 |
| 1.0e-10 | 1000 | 832 | 5.996604108e-06 | 3.266271432e-07 | 4.322697222e-08 |
| 1.0e-12 | 1000 | 832 | 5.764462374e-08 | 3.467107215e-09 | 4.494220551e-10 |
| 1.0e-04 | 10 | 3200 | 9.909450525e+01 | 3.826796809e+00 | 1.693476582e-01 |
| 1.0e-06 | 10 | 3200 | 3.816982607e+00 | 1.213471688e-01 | 4.709770459e-03 |
| 1.0e-08 | 10 | 3200 | 3.815922071e-02 | 1.213122641e-03 | 4.708256139e-05 |
| 1.0e-10 | 10 | 3200 | 3.813762175e-04 | 1.212435815e-05 | 4.705587413e-07 |
| 1.0e-12 | 10 | 3200 | 3.817271774e-06 | 1.213551952e-07 | 4.709944101e-09 |
| 1.0e-04 | 100 | 3200 | 3.143482585e+01 | 9.727906724e-01 | 3.788287105e-02 |
| 1.0e-06 | 100 | 3200 | 3.131742743e-01 | 9.700851890e-03 | 3.860094869e-04 |
| 1.0e-08 | 100 | 3200 | 3.079909585e-03 | 9.540980250e-05 | 3.712080025e-06 |
| 1.0e-10 | 100 | 3200 | 2.691607063e-05 | 8.319642135e-07 | 3.255197023e-08 |
| 1.0e-12 | 100 | 3200 | 2.919915448e-07 | 9.032290823e-09 | 3.498683565e-10 |
| 1.0e-04 | 1000 | 3200 | 9.401467146e+00 | 2.451602774e-01 | 1.455920194e-02 |
| 1.0e-06 | 1000 | 3200 | 4.414162811e-02 | 9.904206696e-04 | 1.272868991e-04 |
| 1.0e-08 | 1000 | 3200 | 3.389981899e-04 | 8.524729074e-06 | 5.037460358e-07 |
| 1.0e-10 | 1000 | 3200 | 5.209882127e-06 | 1.247964701e-07 | 2.561301471e-08 |
| 1.0e-12 | 1000 | 3200 | 5.174107864e-08 | 1.202306134e-09 | 9.390244138e-11 |
| 1.0e-04 | 10 | 12544 | 3.438255814e+03 | 5.363937259e+01 | 1.127788367e+00 |
| 1.0e-06 | 10 | 12544 | 1.370349633e+02 | 2.198055556e+00 | 4.332039094e-02 |
| 1.0e-08 | 10 | 12544 | 1.465851013e+00 | 2.319428071e-02 | 4.352480384e-04 |
| 1.0e-10 | 10 | 12544 | 1.465778745e-02 | 2.319313720e-04 | 4.352266002e-06 |
| 1.0e-12 | 10 | 12544 | 1.465877959e-04 | 2.319470691e-06 | 4.352559735e-08 |
| 1.0e-04 | 100 | 12544 | 3.637687282e+02 | 6.870174997e+00 | 1.523522247e-01 |
| 1.0e-06 | 100 | 12544 | 1.452728642e+01 | 2.285230619e-01 | 4.282662307e-03 |
| 1.0e-08 | 100 | 12544 | 1.453137010e-01 | 2.285874031e-03 | 4.284081509e-05 |
| 1.0e-10 | 100 | 12544 | 1.452390627e-03 | 2.284704395e-05 | 4.281373800e-07 |
| 1.0e-12 | 100 | 12544 | 1.453000760e-05 | 2.285660260e-07 | 4.283671751e-09 |
| 1.0e-04 | 1000 | 12544 | 1.350614750e+02 | 1.995993511e+00 | 9.548668972e-02 |
| 1.0e-06 | 1000 | 12544 | 1.455729090e+00 | 2.166797861e-02 | 9.858883330e-04 |
| 1.0e-08 | 1000 | 12544 | 1.346546831e-02 | 2.010291309e-04 | 9.053823119e-06 |
| 1.0e-10 | 1000 | 12544 | 1.420383921e-04 | 2.053315892e-06 | 9.218187480e-08 |
| 1.0e-12 | 1000 | 12544 | 1.374671217e-06 | 2.051165236e-08 | 9.254671474e-10 |

Πίνακας 4.9: Νόρμες σφαλμάτων για τη μέθοδο Augmented Lagrangian

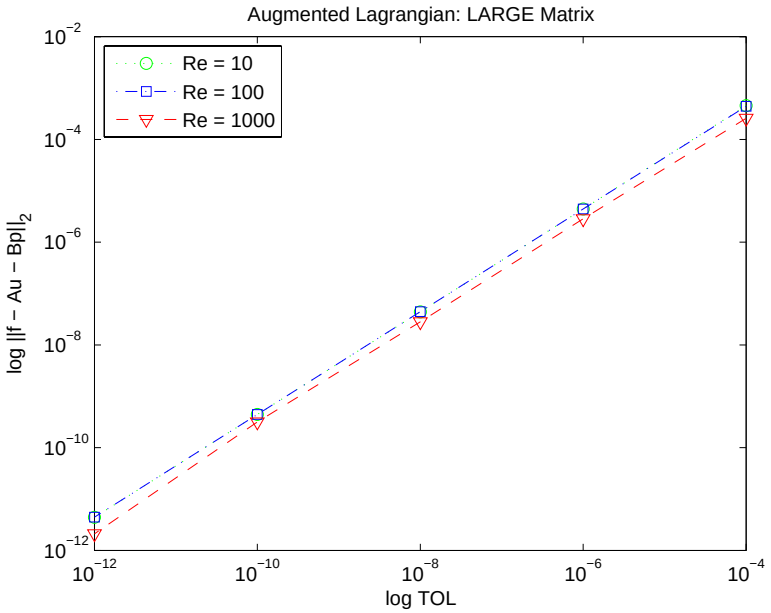
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο Augmented Lagrangian:



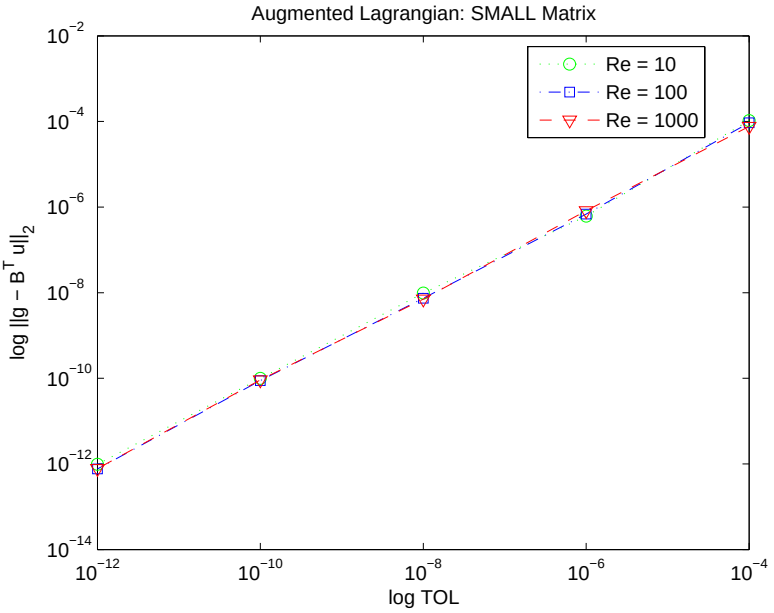

Σχήμα 4.37: Μέθοδος Augmented Lagrangian: TOL vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



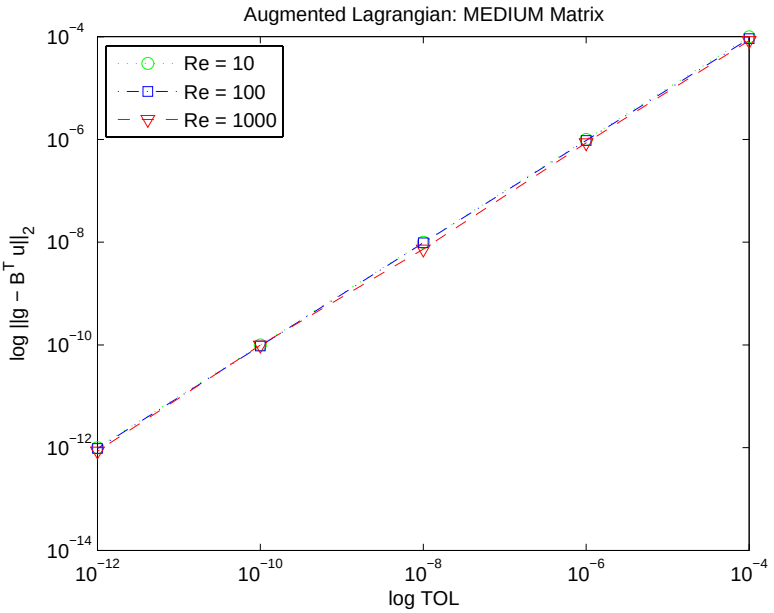

Σχήμα 4.38: Μέθοδος Augmented Lagrangian: TOL vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



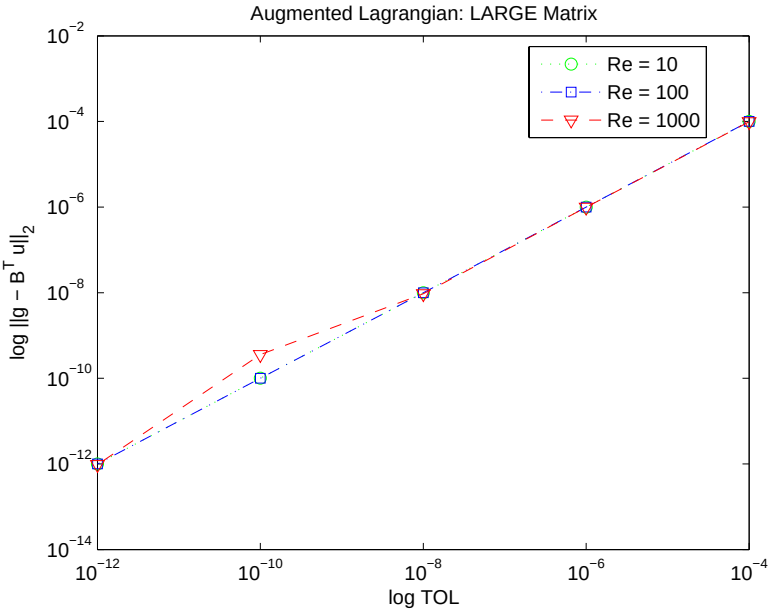

Σχήμα 4.39: Μέθοδος Augmented Lagrangian: TOL vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



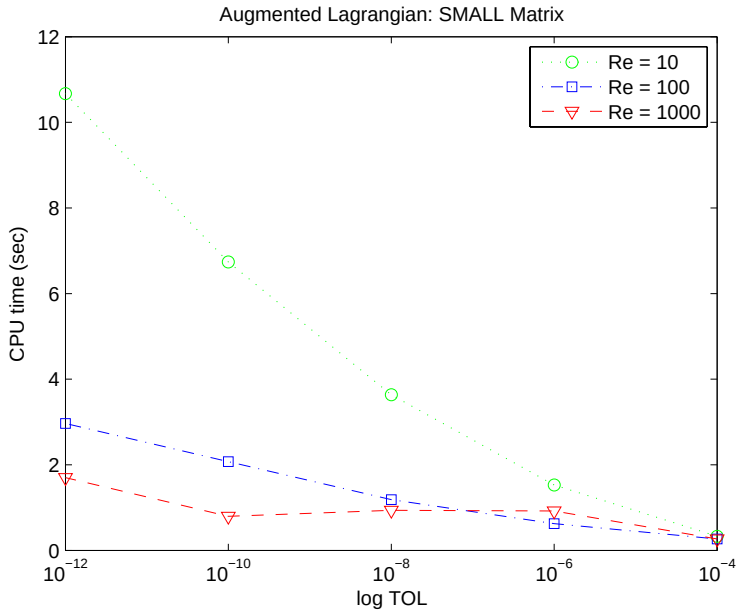

Σχήμα 4.40: Μέθοδος Augmented Lagrangian: TOL vs $\|g - B^T u\|_2$ (SMALL Matrix).



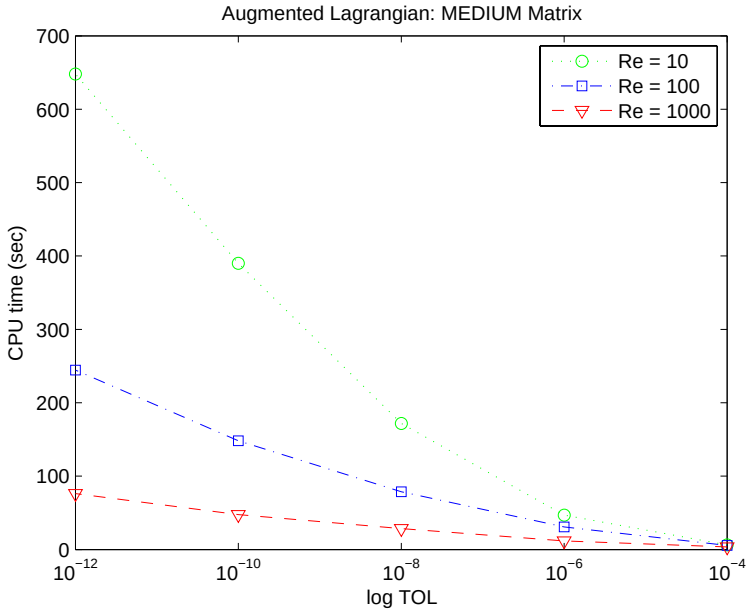

Σχήμα 4.41: Μέθοδος Augmented Lagrangian: TOL vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



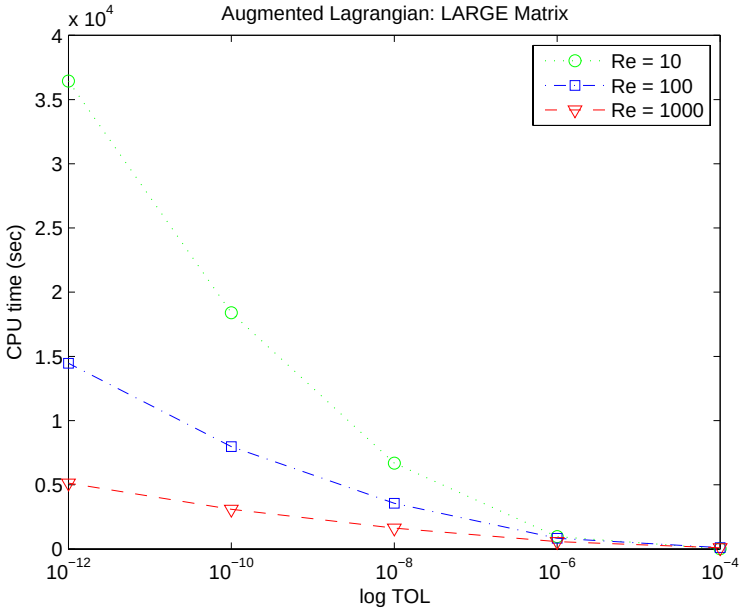

Σχήμα 4.42: Μέθοδος Augmented Lagrangian: TOL vs $\|g - B^T u\|_2$ (LARGE Matrix).



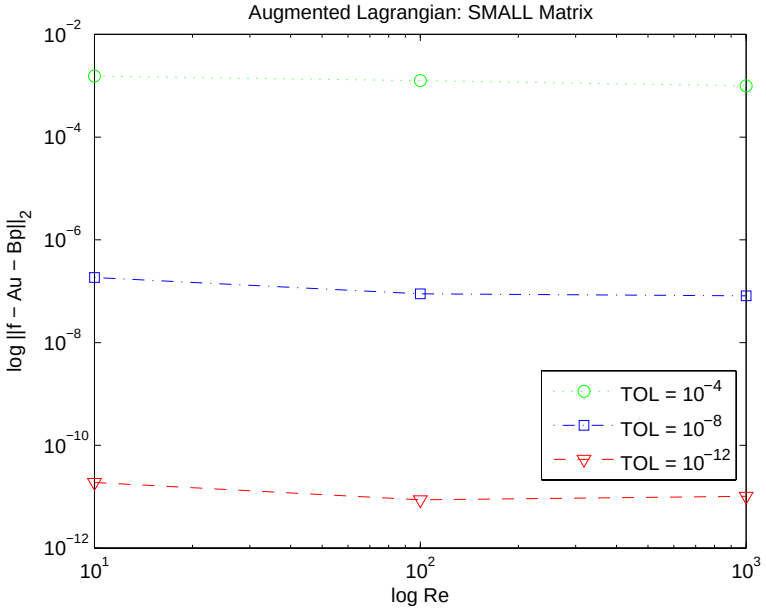

Σχήμα 4.43: Μέθοδος Augmented Lagrangian: TOL vs CPU time, (SMALL Matrix).



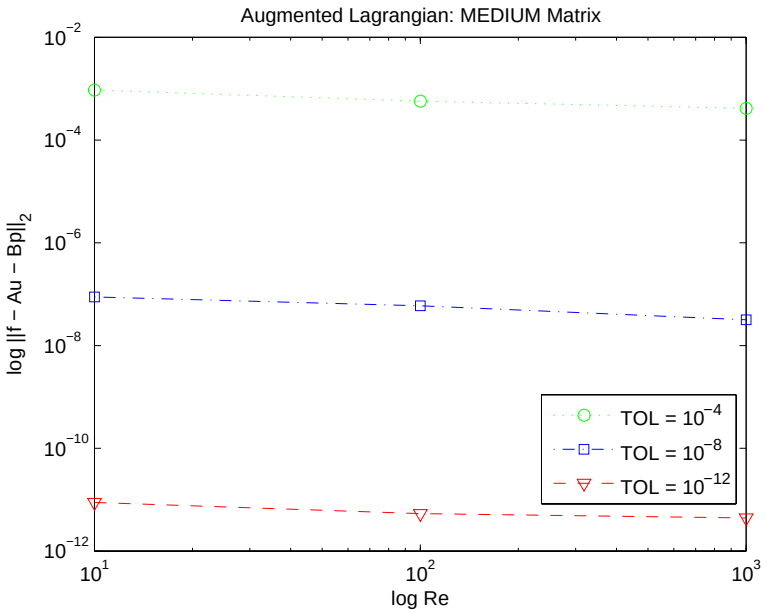

Σχήμα 4.44: Μέθοδος Augmented Lagrangian: TOL vs CPU time, (MEDIUM Matrix).



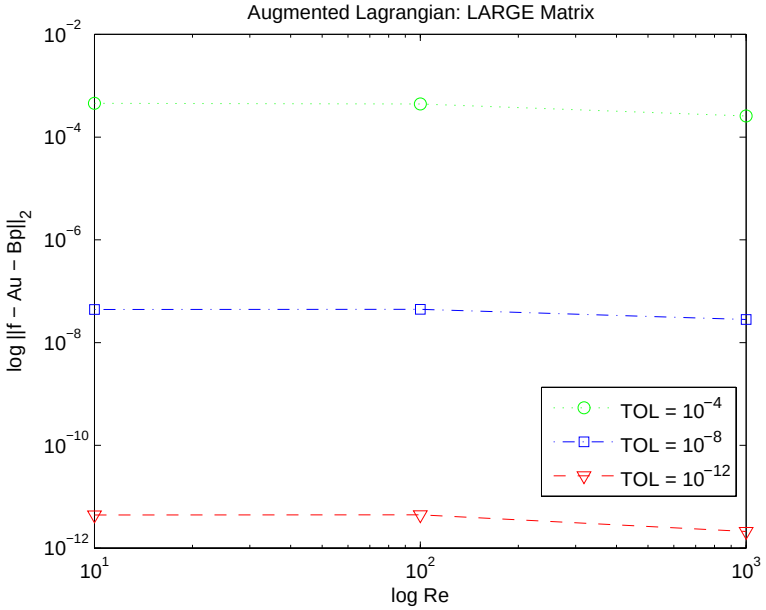

Σχήμα 4.45: Μέθοδος Augmented Lagrangian: TOL vs CPU time, (LARGE Matrix).



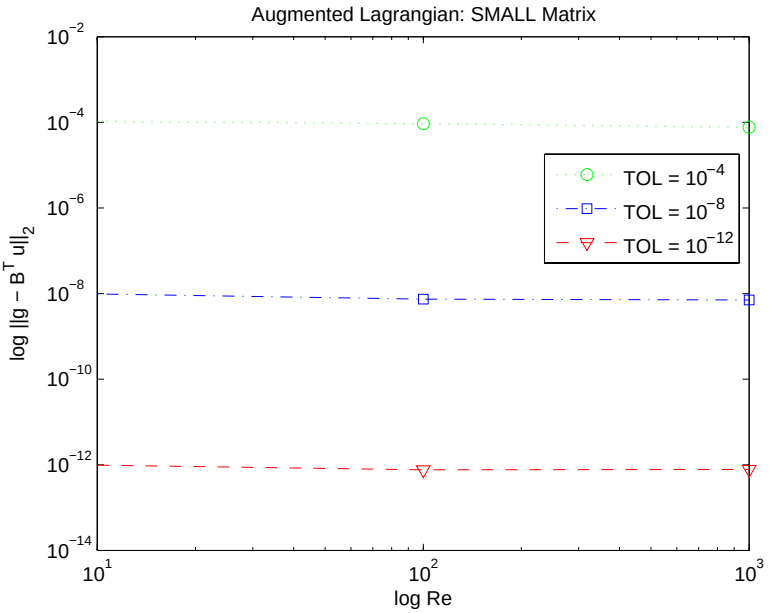

Σχήμα 4.46: Μέθοδος Augmented Lagrangian: Re vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



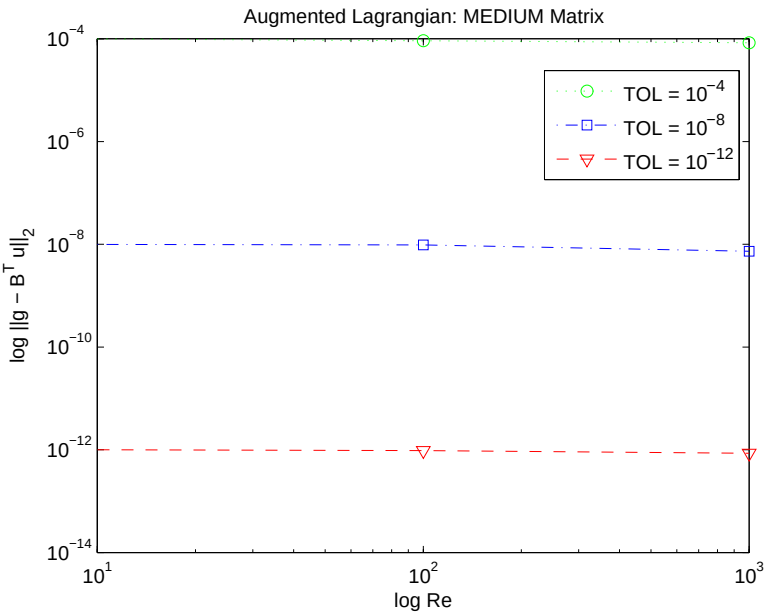

Σχήμα 4.47: Μέθοδος Augmented Lagrangian: Re vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



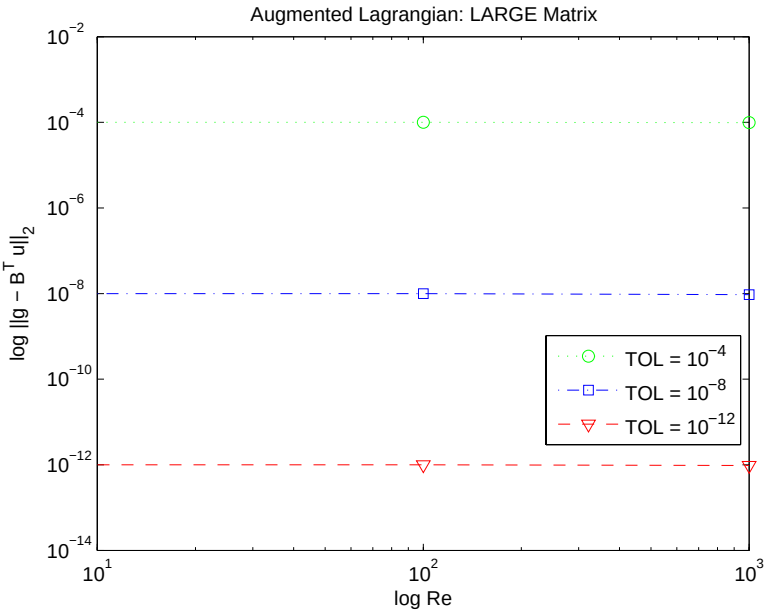

Σχήμα 4.48: Μέθοδος Augmented Lagrangian: Re vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



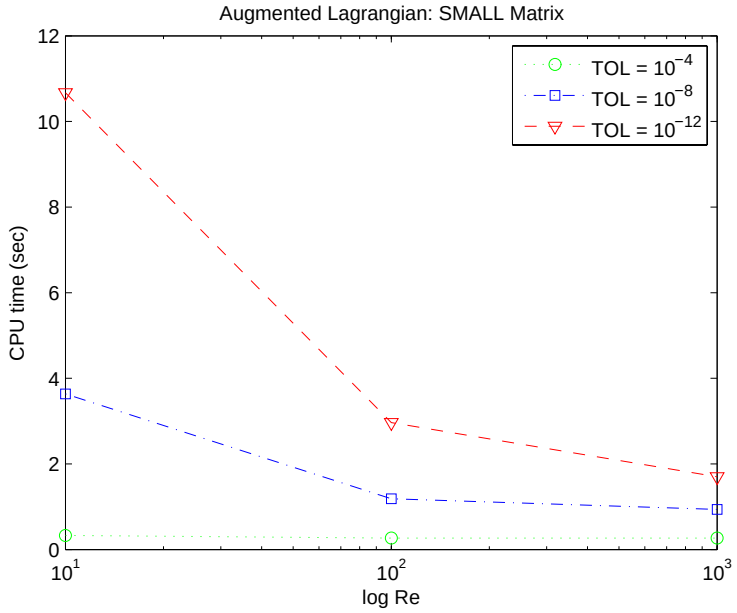

Σχήμα 4.49: Μέθοδος Augmented Lagrangian: Re vs $\|g - B^T u\|_2$ (SMALL Matrix).



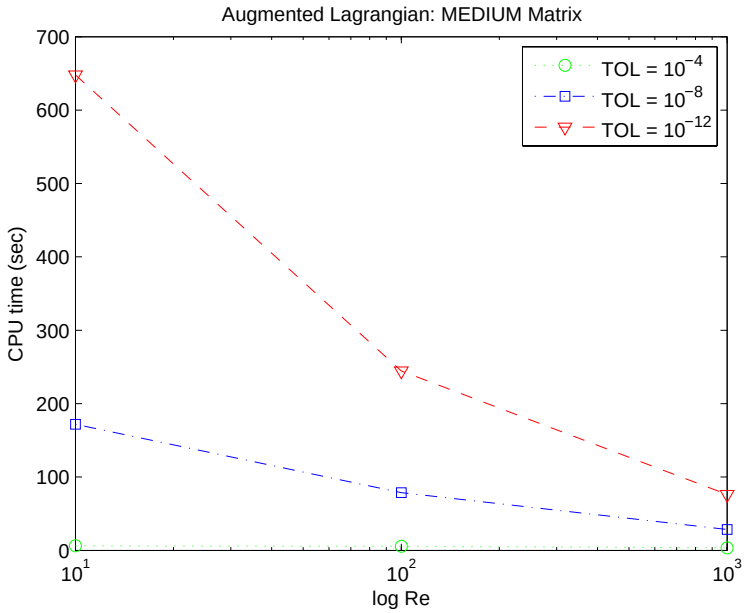

Σχήμα 4.50: Μέθοδος Augmented Lagrangian: Re vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



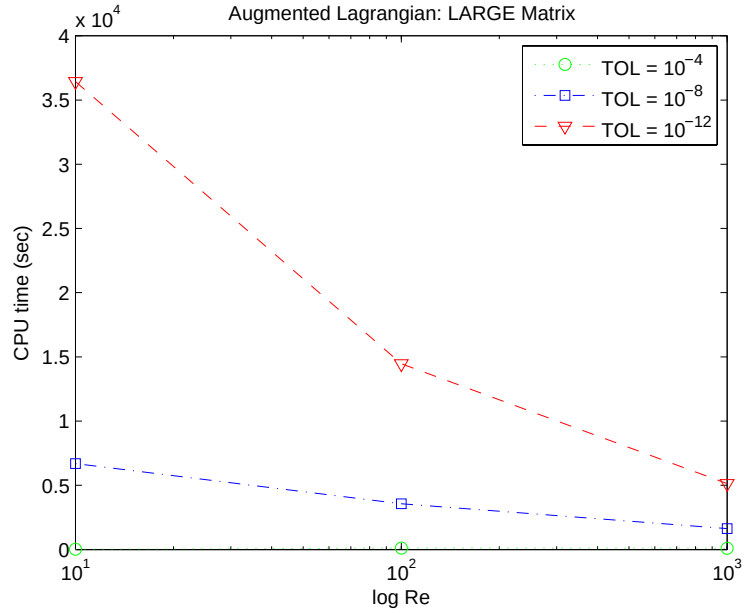

Σχήμα 4.51: Μέθοδος Augmented Lagrangian: Re vs $\|g - B^T u\|_2$ (LARGE Matrix).



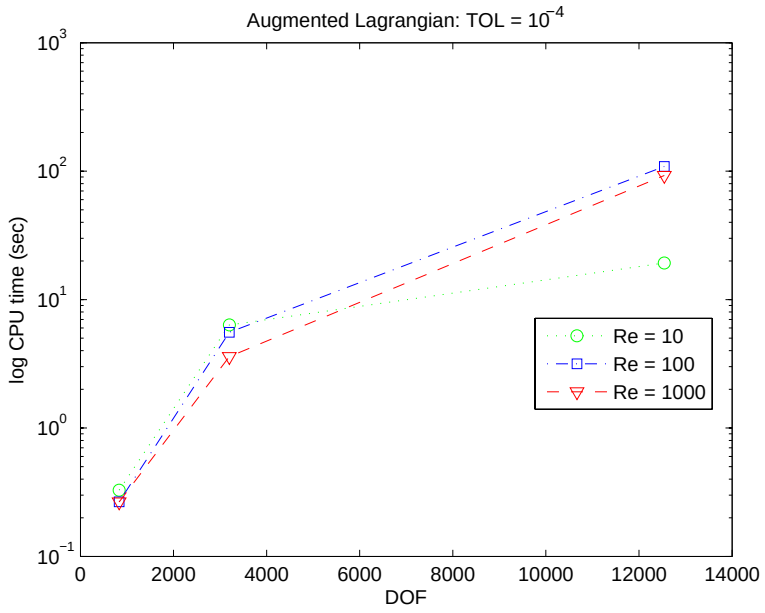

Σχήμα 4.52: Μέθοδος Augmented Lagrangian: Re vs CPU time, (SMALL Matrix).



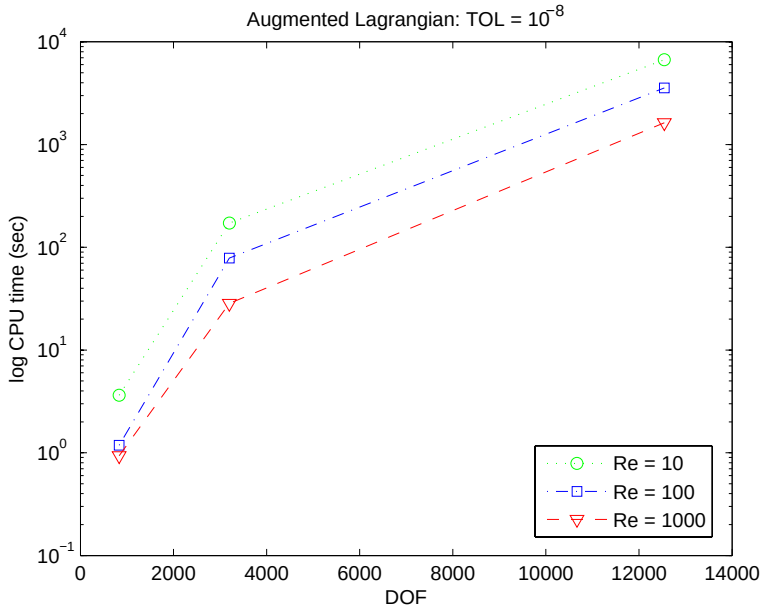

Σχήμα 4.53: Μέθοδος Augmented Lagrangian: Re vs CPU time, (MEDIUM Matrix).



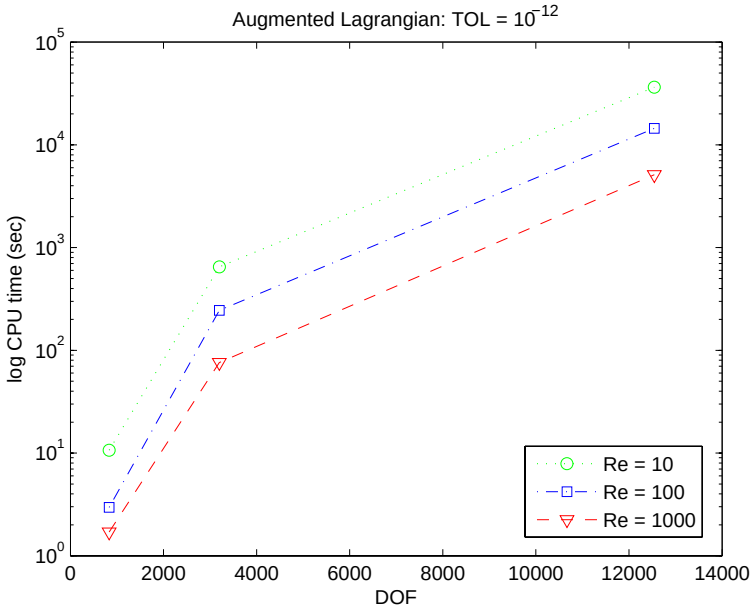

Σχήμα 4.54: Μέθοδος Augmented Lagrangian: Re vs CPU time, (LARGE Matrix).




Σχήμα 4.55: Μέθοδος Augmented Lagrangian: DOF vs CPU time, για TOL $= 10^{-4}$.




Σχήμα 4.56: Μέθοδος Augmented Lagrangian: DOF vs CPU time, για TOL $= 10^{-8}$.




Σχήμα 4.57: Μέθοδος Augmented Lagrangian: DOF vs CPU time, για TOL $= 10^{-12}$.

### 4.3.4 Αποτελέσματα Μεθόδου Simplified Augmented Lagrangian

Στους πίνακες (4.10-4.11), παραθέτουμε τα αποτελέσματα για τη μέθοδο Simplified Augmented Lagrangian. Στα σχήματα (4.58)–(4.78) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου Simplified Augmented Lagrangian. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου Simplified Augmented Lagrangian:

- Σε όλες τις κατηγορίες διαστάσεων των πινάκων και για όλες τις τιμές της ανοχής TOL που δοκιμάστηκαν, όσο αυξάνει ο αριθμός Reynolds της ροής, μειώνεται ο απαιτούμενος υπολογιστικός χρόνος (εξαίρεση αποτελεί ο μεγάλος πίνακας, για TOL= $10^{-4}$, όπου έχουμε αναμοιόμορφη συμπεριφορά)(βλέπε σχήματα (4.64) – (4.66) TOL vs CPU,(4.73) – (4.75) Re vs CPU,(4.76) DOF vs CPU).

- Καθώς αυξάνεται η διάσταση του γραμμικού συστήματος, ο απαιτούμενος υπολογιστικός χρόνος αυξάνεται,για όλες τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής TOL, που δοκιμάστηκαν (βλέπε σχήματα (4.76) – (4.78) DOF vs CPU).

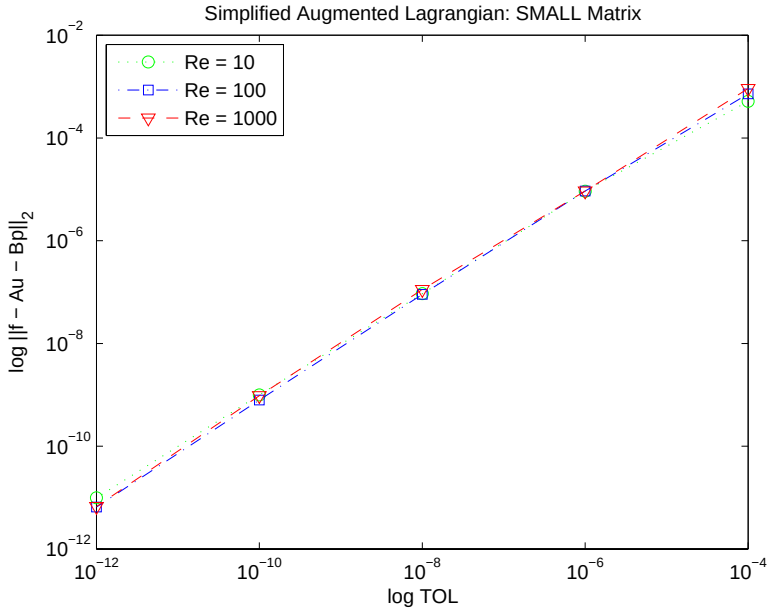
| TOL | Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | #iter | CPU time (sec) |
|---|---|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 5.153625854e-04 | 1.120893950e-04 | 5.274112502e-04 | 152 | 0.078000 |
| 1.0e-06 | 10 | 832 | 9.226277294e-06 | 9.811682052e-07 | 9.278301771e-06 | 548 | 0.390000 |
| 1.0e-08 | 10 | 832 | 9.286979661e-08 | 9.815971728e-09 | 9.338711058e-08 | 961 | 1.123000 |
| 1.0e-10 | 10 | 832 | 1.002514001e-09 | 9.838343953e-11 | 1.007329948e-09 | 1375 | 2.480000 |
| 1.0e-12 | 10 | 832 | 9.844467997e-12 | 9.773862206e-13 | 9.892867833e-12 | 1788 | 4.743000 |
| 1.0e-04 | 100 | 832 | 7.233452259e-04 | 8.453422033e-05 | 7.282680483e-04 | 44 | 0.062000 |
| 1.0e-06 | 100 | 832 | 9.319230409e-06 | 8.429795320e-07 | 9.357278980e-06 | 83 | 0.218000 |
| 1.0e-08 | 100 | 832 | 9.027413269e-08 | 8.961105173e-09 | 9.071780663e-08 | 123 | 0.500000 |
| 1.0e-10 | 100 | 832 | 7.927265870e-10 | 7.122869021e-11 | 7.959202020e-10 | 175 | 0.967000 |
| 1.0e-12 | 100 | 832 | 6.533338787e-12 | 8.390314236e-13 | 6.586993961e-12 | 211 | 1.560000 |
| 1.0e-04 | 1000 | 832 | 9.168353996e-04 | 9.209560019e-05 | 9.214492658e-04 | 86 | 0.109000 |
| 1.0e-06 | 1000 | 832 | 9.289590122e-06 | 9.034193985e-07 | 9.333415840e-06 | 18 | 0.109000 |
| 1.0e-08 | 1000 | 832 | 1.126842680e-07 | 9.209103260e-09 | 1.130599480e-07 | 41 | 0.203000 |
| 1.0e-10 | 1000 | 832 | 9.632080146e-10 | 8.750715113e-11 | 9.671748451e-10 | 31 | 0.296000 |
| 1.0e-12 | 1000 | 832 | 6.685205232e-12 | 7.687066104e-13 | 6.729255446e-12 | 44 | 0.452000 |
| 1.0e-04 | 10 | 3200 | 2.245873994e-04 | 9.964279615e-05 | 2.456993829e-04 | 910 | 0.999000 |
| 1.0e-06 | 10 | 3200 | 3.955398960e-07 | 9.991631527e-07 | 1.074606354e-06 | 5943 | 8.127000 |
| 1.0e-08 | 10 | 3200 | 3.947792332e-09 | 9.994190426e-09 | 1.074564594e-08 | 13961 | 38.579000 |
| 1.0e-10 | 10 | 3200 | 3.946164429e-11 | 9.990224619e-11 | 1.074135939e-10 | 21983 | 104.599000 |
| 1.0e-12 | 10 | 3200 | 3.946486913e-13 | 9.992185357e-13 | 1.074330150e-12 | 30004 | 239.025000 |
| 1.0e-04 | 100 | 3200 | 3.701769078e-04 | 9.631555858e-05 | 3.825018038e-04 | 249 | 0.920000 |
| 1.0e-06 | 100 | 3200 | 3.186545803e-06 | 9.957491662e-07 | 3.338501244e-06 | 1018 | 5.226000 |
| 1.0e-08 | 100 | 3200 | 2.425355932e-08 | 9.712422457e-09 | 2.612596964e-08 | 1812 | 16.723000 |
| 1.0e-10 | 100 | 3200 | 3.714259194e-10 | 9.554750291e-11 | 3.835186291e-10 | 2639 | 42.792000 |
| 1.0e-12 | 100 | 3200 | 3.319535896e-12 | 9.932777979e-13 | 3.464955894e-12 | 3424 | 88.889000 |
| 1.0e-04 | 1000 | 3200 | 3.906597863e-04 | 8.790608810e-05 | 4.004279572e-04 | 69 | 0.920000 |
| 1.0e-06 | 1000 | 3200 | 3.102575081e-06 | 8.504377268e-07 | 3.217019810e-06 | 154 | 3.136000 |
| 1.0e-08 | 1000 | 3200 | 4.054668578e-08 | 8.177254718e-09 | 4.136304175e-08 | 242 | 7.956000 |
| 1.0e-10 | 1000 | 3200 | 4.005480733e-10 | 7.344289276e-11 | 4.072255119e-10 | 318 | 17.207000 |
| 1.0e-12 | 1000 | 3200 | 3.772335565e-12 | 9.046687146e-13 | 3.879296469e-12 | 401 | 31.247000 |
| 1.0e-04 | 10 | 12544 | 1.298132560e-04 | 9.968608180e-05 | 1.636728332e-04 | 88 | 4.602000 |
| 1.0e-06 | 10 | 12544 | 1.175424419e-07 | 9.999223308e-07 | 1.006807277e-06 | 37888 | 113.585000 |
| 1.0e-08 | 10 | 12544 | 1.414457252e-10 | 9.999599166e-09 | 1.000059950e-08 | 182567 | 924.914000 |
| 1.0e-10 | 10 | 12544 | 1.414385318e-12 | 9.999497309e-11 | 1.000049755e-10 | 333700 | 4034.014000 |
| 1.0e-12 | 10 | 12544 | 1.434795308e-14 | 9.999587592e-13 | 1.000061690e-12 | 484833 | 10802.508000 |
| 1.0e-04 | 100 | 12544 | 1.031914850e-04 | 9.990870576e-05 | 1.436322807e-04 | 1658 | 13.011000 |
| 1.0e-06 | 100 | 12544 | 3.007310389e-07 | 9.995477165e-07 | 1.043807834e-06 | 10700 | 89.685000 |
| 1.0e-08 | 100 | 12544 | 3.005375912e-09 | 9.994184004e-09 | 1.043628278e-08 | 25812 | 509.124000 |
| 1.0e-10 | 100 | 12544 | 3.005678284e-11 | 9.995197622e-11 | 1.043734053e-10 | 40927 | 1482.711000 |
| 1.0e-12 | 100 | 12544 | 3.006036466e-13 | 9.996804104e-13 | 1.043898211e-12 | 56042 | 3764.164000 |
| 1.0e-04 | 1000 | 12544 | 1.312457076e-04 | 9.840236982e-05 | 1.640379900e-04 | 437 | 11.263000 |
| 1.0e-06 | 1000 | 12544 | 1.226939614e-06 | 9.151888393e-07 | 1.530670254e-06 | 1877 | 69.702000 |
| 1.0e-08 | 1000 | 12544 | 1.232271433e-08 | 9.720437121e-09 | 1.569510071e-08 | 3359 | 257.058000 |
| 1.0e-10 | 1000 | 12544 | 1.342704675e-10 | 9.952731958e-11 | 1.671354115e-10 | 4876 | 684.236000 |
| 1.0e-12 | 1000 | 12544 | 1.416833129e-12 | 9.871098884e-13 | 1.726789521e-12 | 6415 | 1545.736000 |

Πίνακας 4.10: Αποτελέσματα για τη μέθοδο Simplified Augmented Lagrangian

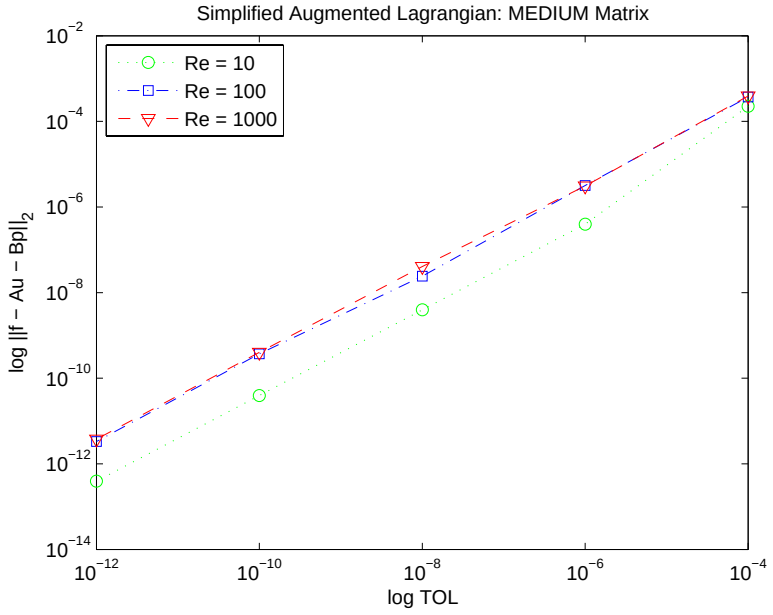
| TOL | Re | DOF | $\|x-x^*\|_1$ | $\|x-x^*\|_2$ | $\|x-x^*\|_\infty$ |
|---|---|---|---|---|---|
| 1.0e-04 | 10 | 832 | 9.393372955e+00 | 5.983367146e-01 | 4.783724183e-02 |
| 1.0e-06 | 10 | 832 | 1.035994650e-01 | 6.552477064e-03 | 5.253641163e-04 |
| 1.0e-08 | 10 | 832 | 1.041084216e-03 | 6.594757853e-05 | 5.123459696e-06 |
| 1.0e-10 | 10 | 832 | 1.003461611e-05 | 6.338637295e-07 | 5.010152992e-08 |
| 1.0e-12 | 10 | 832 | 1.001951589e-07 | 6.332251651e-09 | 5.029390238e-10 |
| 1.0e-04 | 100 | 832 | 5.971495008e-01 | 2.738105488e-02 | 3.293404423e-03 |
| 1.0e-06 | 100 | 832 | 5.203762556e-03 | 2.800627115e-04 | 6.570244033e-05 |
| 1.0e-08 | 100 | 832 | 5.953221529e-05 | 3.429745595e-06 | 7.306506447e-07 |
| 1.0e-10 | 100 | 832 | 6.435222036e-07 | 4.208014275e-08 | 5.126203240e-09 |
| 1.0e-12 | 100 | 832 | 3.748988231e-09 | 2.118828587e-10 | 5.395384139e-11 |
| 1.0e-04 | 1000 | 832 | 1.949006560e+00 | 1.044334966e-01 | 1.165360428e-02 |
| 1.0e-06 | 1000 | 832 | 6.741889250e-02 | 3.864358180e-03 | 4.314464569e-04 |
| 1.0e-08 | 1000 | 832 | 5.634366434e-04 | 2.975777658e-05 | 3.865442891e-06 |
| 1.0e-10 | 1000 | 832 | 6.716697117e-06 | 3.944167019e-07 | 5.394590885e-08 |
| 1.0e-12 | 1000 | 832 | 6.018195053e-08 | 3.431899001e-09 | 4.710289936e-10 |
| 1.0e-04 | 10 | 3200 | 1.547706248e+02 | 5.349243748e+00 | 2.316124459e-01 |
| 1.0e-06 | 10 | 3200 | 5.457583294e+00 | 1.735565781e-01 | 6.744380935e-03 |
| 1.0e-08 | 10 | 3200 | 5.459372479e-02 | 1.736018250e-03 | 6.744443096e-05 |
| 1.0e-10 | 10 | 3200 | 5.457206093e-04 | 1.735329372e-05 | 6.741767657e-07 |
| 1.0e-12 | 10 | 3200 | 5.458272019e-06 | 1.735668311e-07 | 6.743085201e-09 |
| 1.0e-04 | 100 | 3200 | 4.000471928e+01 | 1.252587658e+00 | 5.108427633e-02 |
| 1.0e-06 | 100 | 3200 | 4.640902174e-01 | 1.430948079e-02 | 5.484386794e-04 |
| 1.0e-08 | 100 | 3200 | 5.057501630e-03 | 1.566418026e-04 | 6.033865178e-06 |
| 1.0e-10 | 100 | 3200 | 4.337572045e-05 | 1.343899305e-06 | 5.210675846e-08 |
| 1.0e-12 | 100 | 3200 | 4.850148493e-07 | 1.499815058e-08 | 5.861231500e-10 |
| 1.0e-04 | 1000 | 3200 | 7.684976612e+00 | 1.952134275e-01 | 1.192877694e-02 |
| 1.0e-06 | 1000 | 3200 | 3.030477160e-02 | 7.443302109e-04 | 4.223098583e-05 |
| 1.0e-08 | 1000 | 3200 | 2.969141038e-04 | 7.032914065e-06 | 5.135608010e-07 |
| 1.0e-10 | 1000 | 3200 | 3.161993708e-06 | 7.421203701e-08 | 4.452874625e-09 |
| 1.0e-12 | 1000 | 3200 | 3.656225867e-08 | 8.380774394e-10 | 1.590579890e-10 |
| 1.0e-04 | 10 | 12544 | 4.057338810e+03 | 6.323882359e+01 | 1.269150425e+00 |
| 1.0e-06 | 10 | 12544 | 1.668050940e+02 | 2.730309686e+00 | 5.606331777e-02 |
| 1.0e-08 | 10 | 12544 | 2.073948463e+00 | 3.281646884e-02 | 6.158339094e-04 |
| 1.0e-10 | 10 | 12544 | 2.073927310e-02 | 3.281613381e-04 | 6.158275929e-06 |
| 1.0e-12 | 10 | 12544 | 2.073949532e-04 | 3.281648541e-06 | 6.158341748e-08 |
| 1.0e-04 | 100 | 12544 | 5.059952229e+02 | 8.769621862e+00 | 1.866607661e-01 |
| 1.0e-06 | 100 | 12544 | 2.076250429e+01 | 3.266726242e-01 | 6.127776244e-03 |
| 1.0e-08 | 100 | 12544 | 2.076010444e-01 | 3.266310340e-03 | 6.126644048e-05 |
| 1.0e-10 | 100 | 12544 | 2.076221022e-03 | 3.266641622e-05 | 6.127265204e-07 |
| 1.0e-12 | 100 | 12544 | 2.076550894e-05 | 3.267160678e-07 | 6.128239360e-09 |
| 1.0e-04 | 1000 | 12544 | 1.534362235e+02 | 2.260989788e+00 | 1.098026244e-01 |
| 1.0e-06 | 1000 | 12544 | 1.808167565e+00 | 2.703307143e-02 | 1.225193379e-03 |
| 1.0e-08 | 1000 | 12544 | 2.005839638e-02 | 2.997500641e-04 | 1.317792077e-05 |
| 1.0e-10 | 1000 | 12544 | 2.011438759e-04 | 3.008800620e-06 | 1.376788742e-07 |
| 1.0e-12 | 1000 | 12544 | 1.852570870e-06 | 2.767964954e-08 | 1.248232517e-09 |

Πίνακας 4.11: Νόρμες σφαλμάτων για τη μέθοδο Simplified Augmented Lagrangian

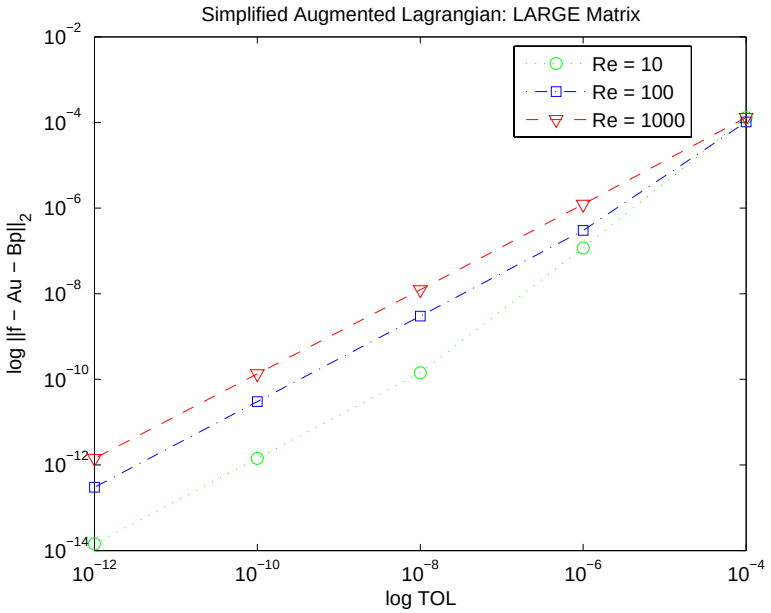
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο Simplified Augmented Lagrangian:



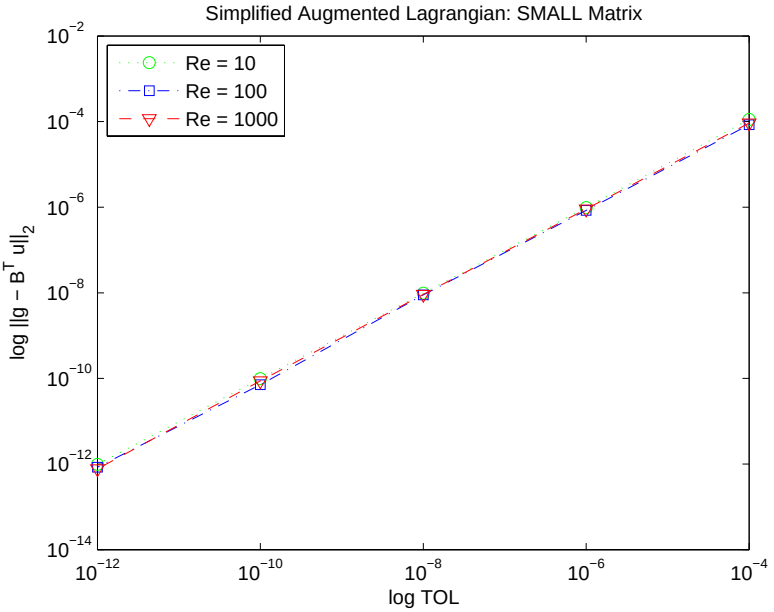

Σχήμα 4.58: Μέθοδος Simplified Augmented Lagrangian: TOL vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



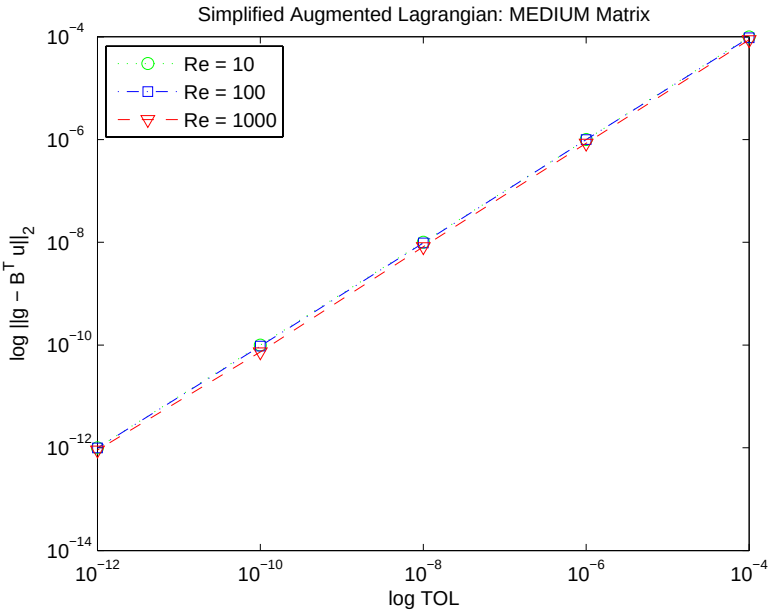

Σχήμα 4.59: Μέθοδος Simplified Augmented Lagrangian: TOL vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



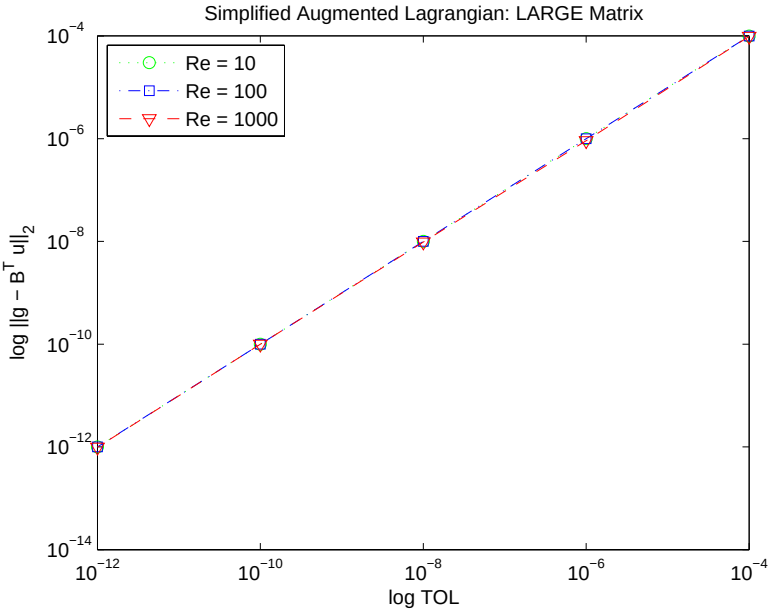

Σχήμα 4.60: Μέθοδος Simplified Augmented Lagrangian: TOL vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



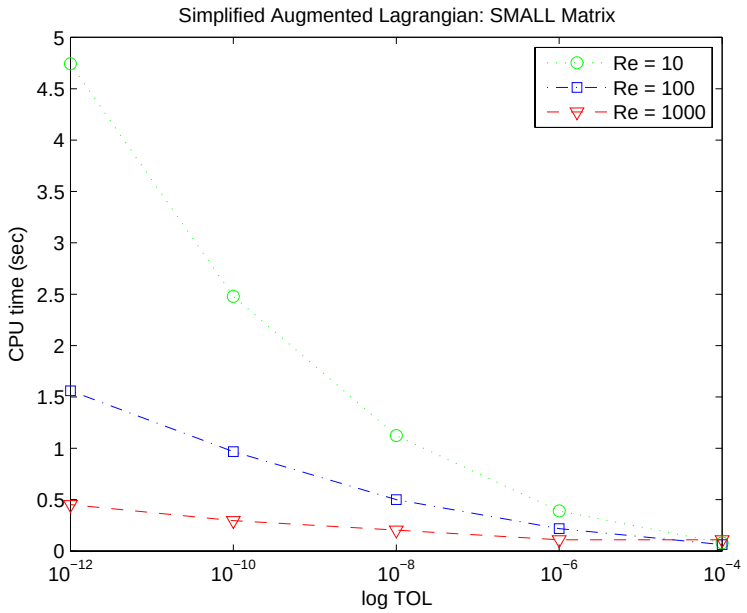

Σχήμα 4.61: Μέθοδος Simplified Augmented Lagrangian: TOL vs $\|g - B^T u\|_2$ (SMALL Matrix).



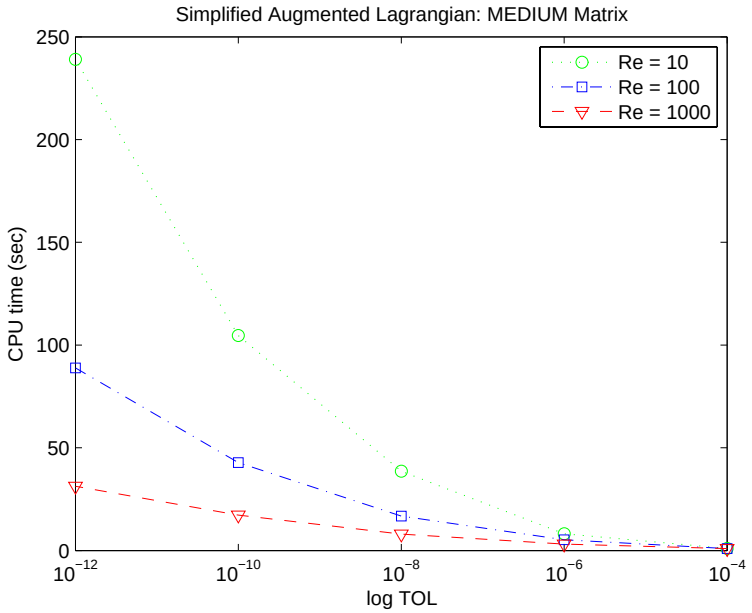

Σχήμα 4.62: Μέθοδος Simplified Augmented Lagrangian: TOL vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



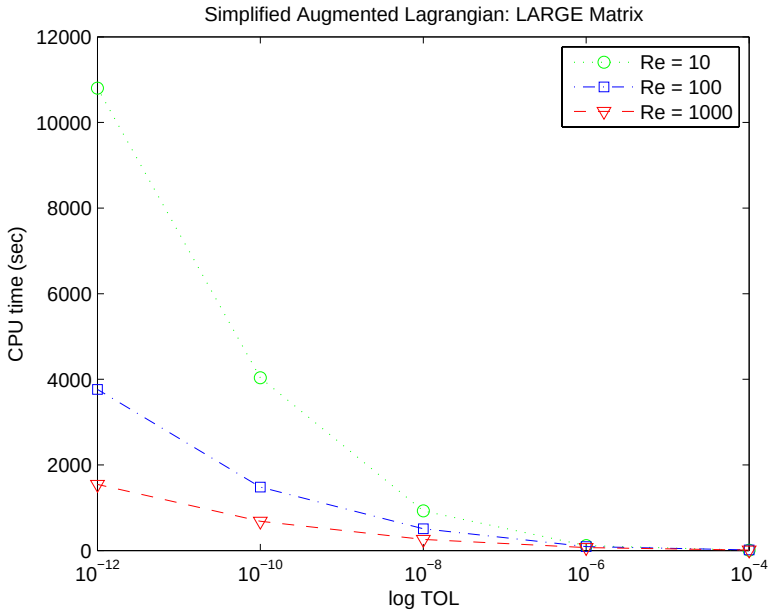

Σχήμα 4.63: Μέθοδος Simplified Augmented Lagrangian: TOL vs $\|g - B^T u\|_2$ (LARGE Matrix).



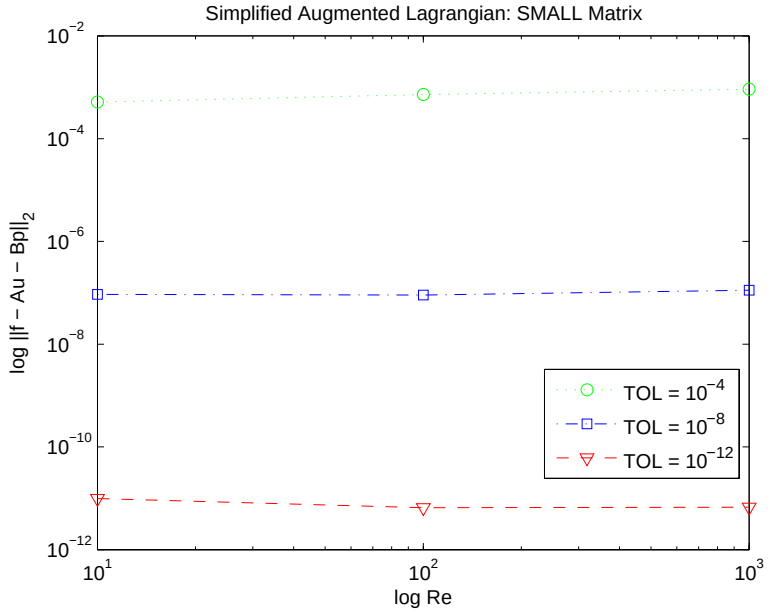

Σχήμα 4.64: Μέθοδος Simplified Augmented Lagrangian: TOL vs CPU time, (SMALL Matrix).



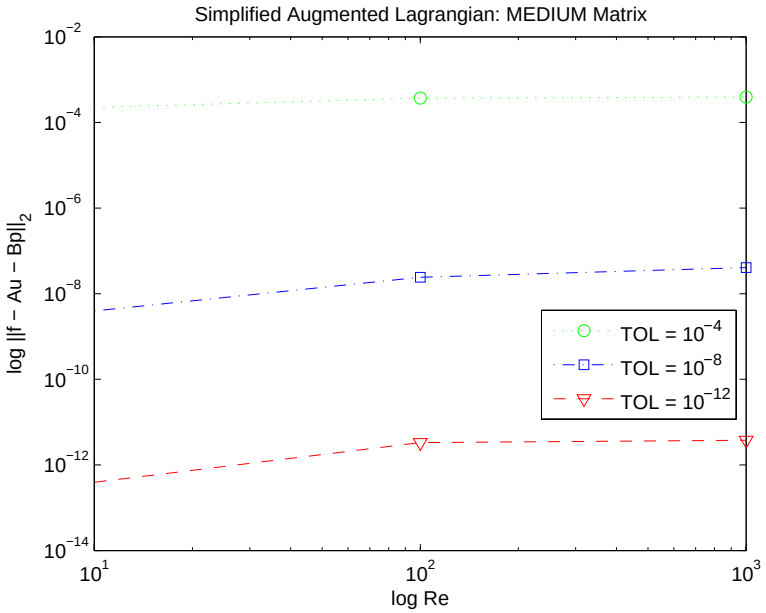

Σχήμα 4.65: Μέθοδος Simplified Augmented Lagrangian: TOL vs CPU time, (MEDIUM Matrix).



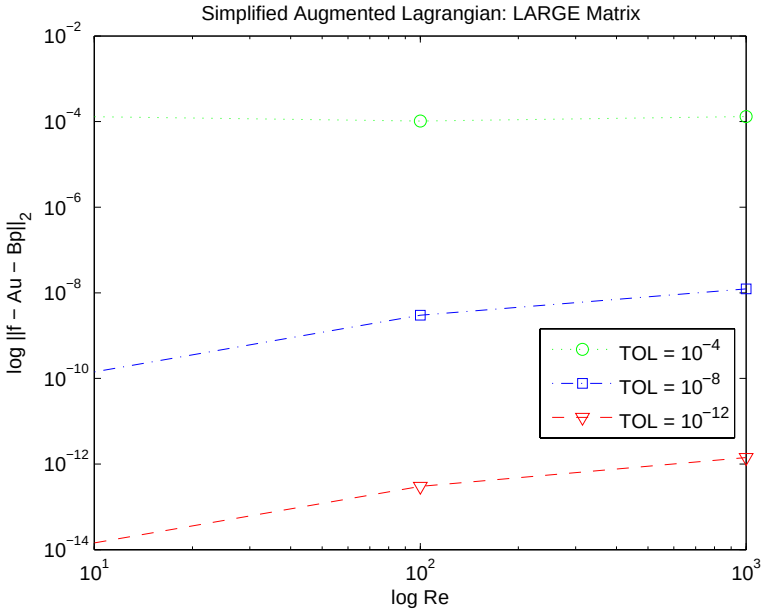

Σχήμα 4.66: Μέθοδος Simplified Augmented Lagrangian: TOL vs CPU time, (LARGE Matrix).



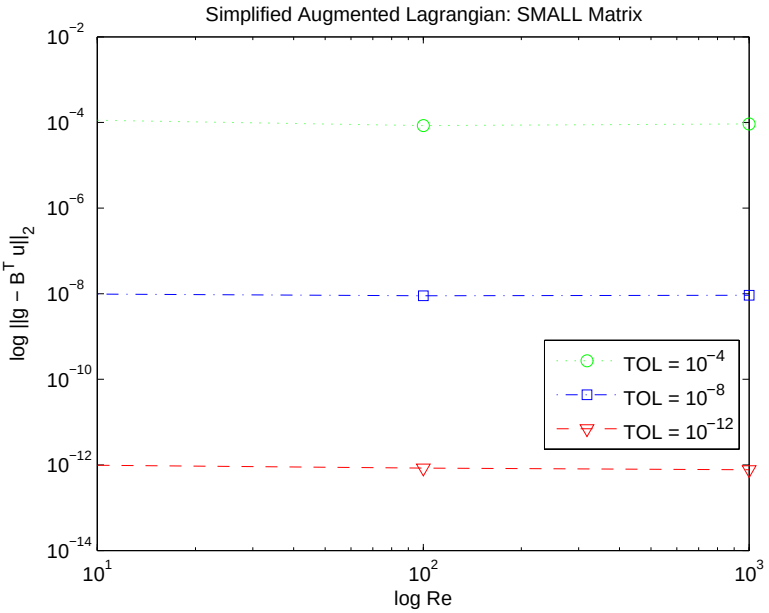

Σχήμα 4.67: Μέθοδος Simplified Augmented Lagrangian: Re vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



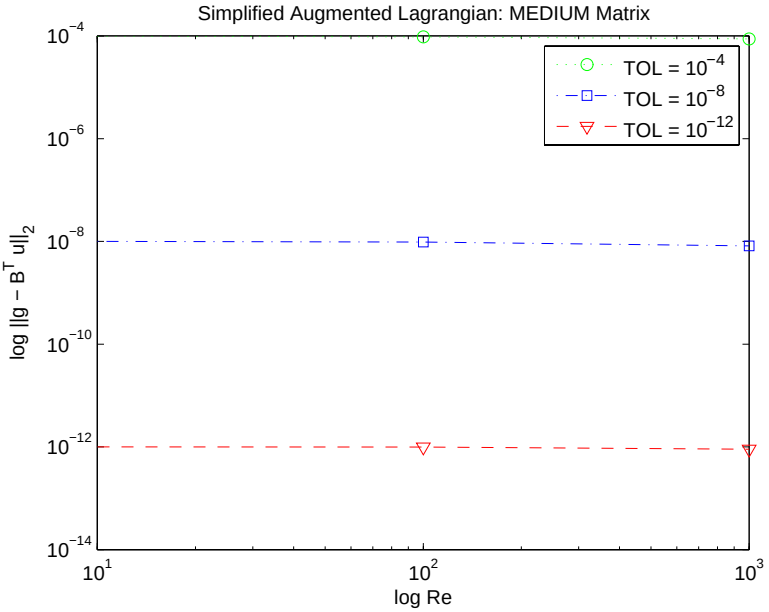

Σχήμα 4.68: Μέθοδος Simplified Augmented Lagrangian: Re vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



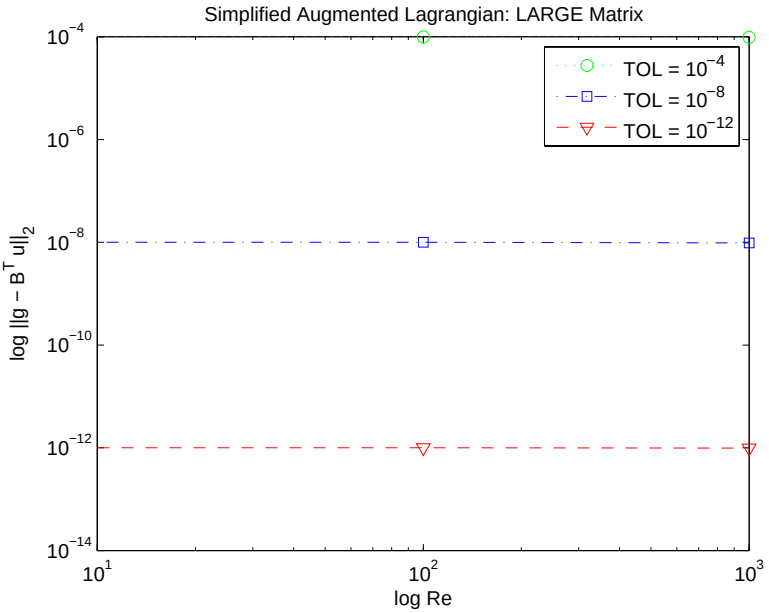

Σχήμα 4.69: Μέθοδος Simplified Augmented Lagrangian: Re vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



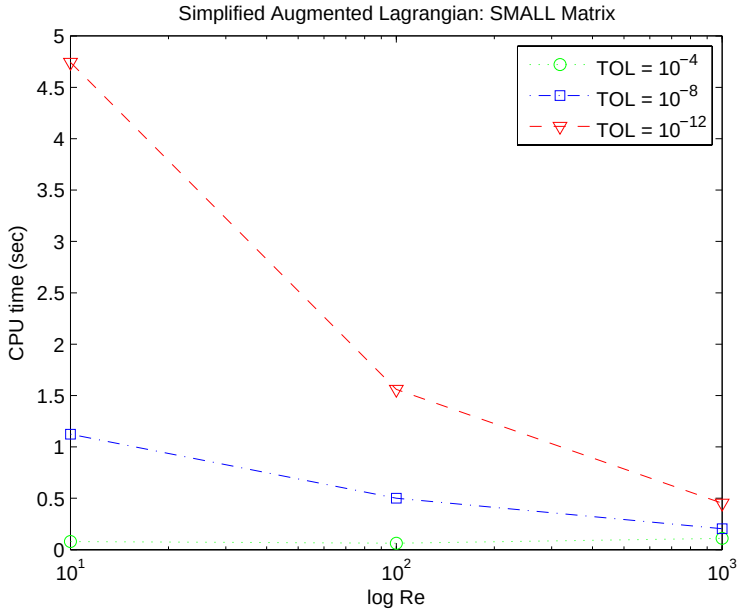

Σχήμα 4.70: Μέθοδος Simplified Augmented Lagrangian: Re vs $\|g - B^T u\|_2$ (SMALL Matrix).



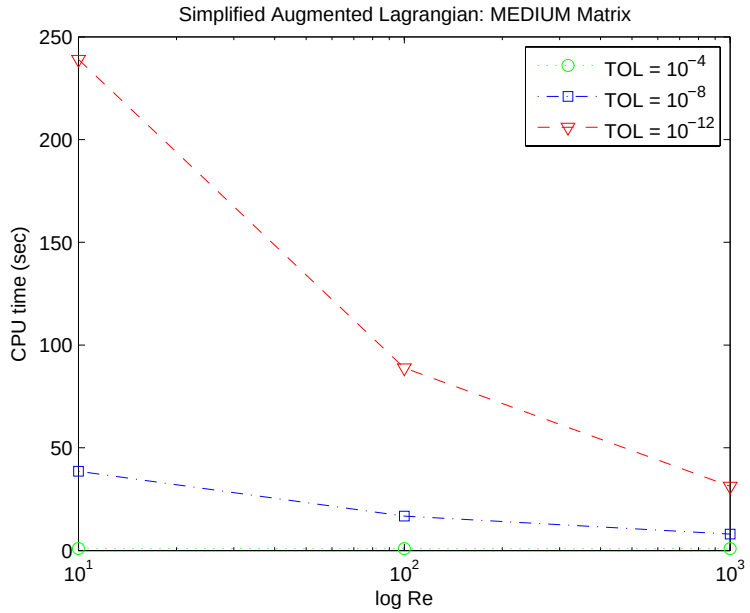

Σχήμα 4.71: Μέθοδος Simplified Augmented Lagrangian: Re vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



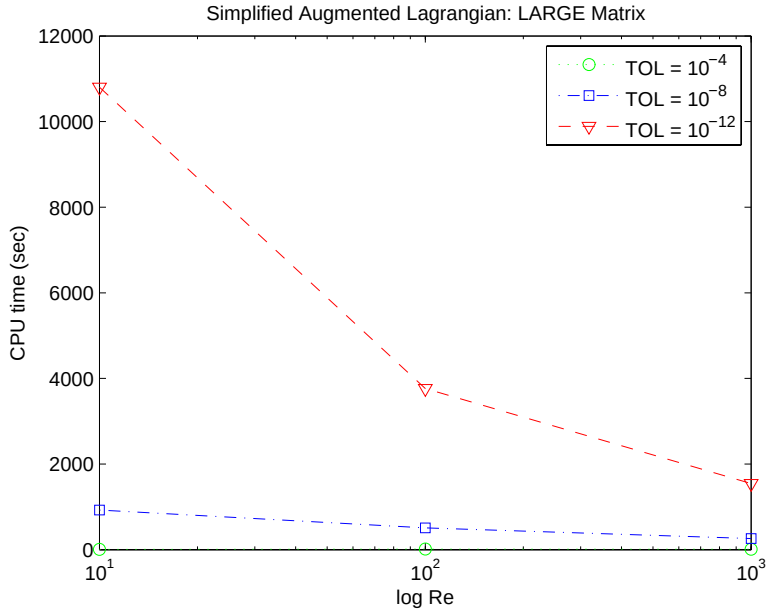

Σχήμα 4.72: Μέθοδος Simplified Augmented Lagrangian: Re vs $\|g - B^T u\|_2$ (LARGE Matrix).



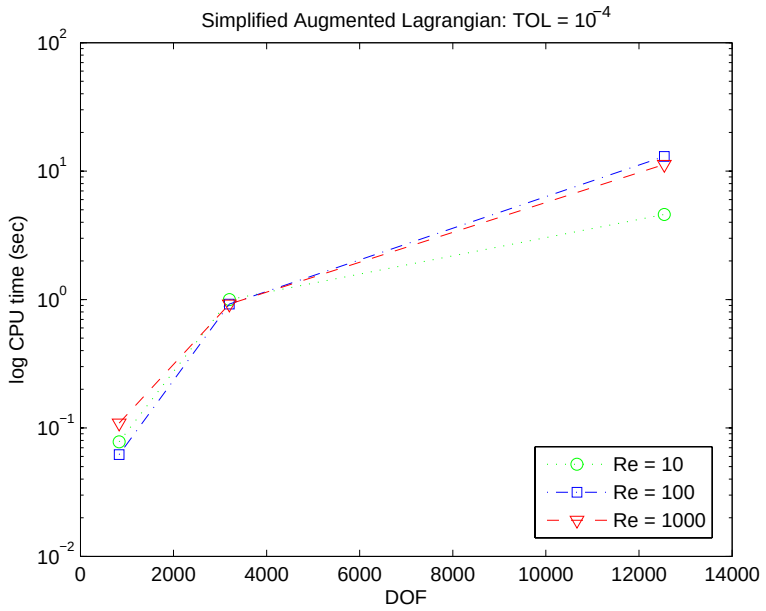

Σχήμα 4.73: Μέθοδος Simplified Augmented Lagrangian: Re vs CPU time, (SMALL Matrix).



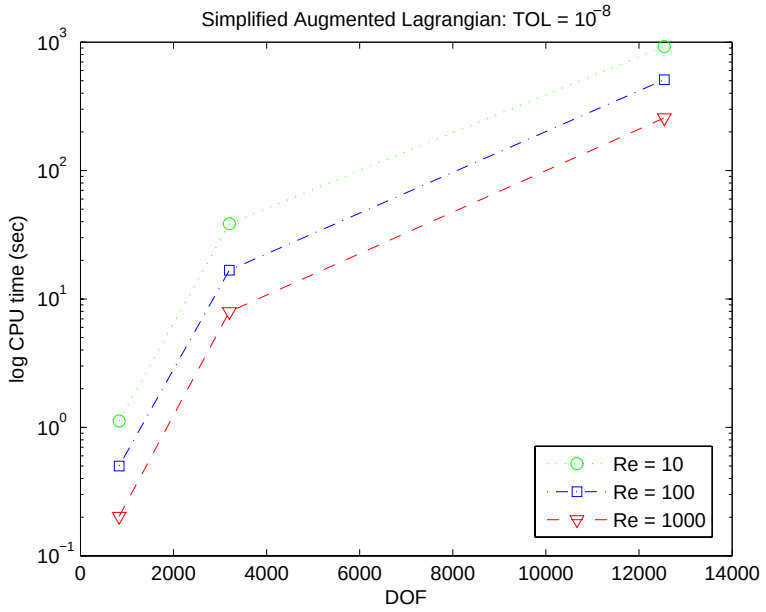

Σχήμα 4.74: Μέθοδος Simplified Augmented Lagrangian: Re vs CPU time, (MEDIUM Matrix).



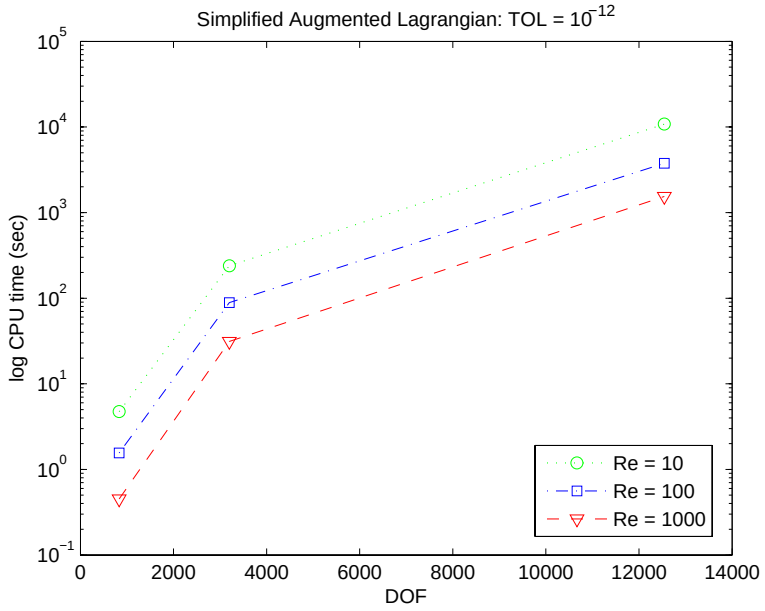

Σχήμα 4.75: Μέθοδος Simplified Augmented Lagrangian: Re vs CPU time, (LARGE Matrix).




Σχήμα 4.76: Μέθοδος Simplified Augmented Lagrangian: DOF vs CPU time, για TOL $= 10^{-4}$.




Σχήμα 4.77: Μέθοδος Simplified Augmented Lagrangian: DOF vs CPU time, για TOL $= 10^{-8}$.




Σχήμα 4.78: Μέθοδος Simplified Augmented Lagrangian: DOF vs CPU time, για TOL $= 10^{-12}$.

### 4.3.5 Αποτελέσματα Μεθόδου MA57

Στους πίνακες (4.12-4.13), παραθέτουμε τα αποτελέσματα για τη μέθοδο MA57 με χρήση MeTiS Ordering (βλέπε [20]). Στους πίνακες (4.14-4.15),φαίνονται τα αποτελέσματα για τη μέθοδο MA57 με χρήση AMD Ordering. (Approximate Minimum Degree Ordering). Στα σχήματα (4.79)–(4.82) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου MA57. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου MA57:

- Σε όλες τις κατηγορίες διαστάσεων των πινάκων που δοκιμάστηκαν, όσο αυξάνει ο αριθμός Reynolds της ροής, **ο απαιτούμενος υπολογιστικός χρόνος παραμένει σταθερός** (βλέπε σχήμα (4.81) Re vs CPU).

- Καθώς αυξάνεται η διάσταση του γραμμικού συστήματος, ο απαιτούμενος υπολογιστικός χρόνος αυξάνεται, ομοιόμορφα ως προς τις τιμές του αριθμού Reynolds της ροής, που δοκιμάστηκαν (βλέπε σχήμα (4.82) DOF vs CPU).

| Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | CPU time (sec) |
|---|---|---|---|---|---|
| 10 | 832 | 7.158252447e-15 | 7.277573323e-16 | 7.195151759e-15 | 0.031000 |
| 100 | 832 | 8.895083896e-16 | 1.524615689e-15 | 1.765128373e-15 | 0.032000 |
| 1000 | 832 | 6.181741231e-16 | 1.464449901e-14 | 1.465754039e-14 | 0.046000 |
| 10 | 3200 | 2.553327950e-14 | 8.416037359e-16 | 2.554714580e-14 | 0.359000 |
| 100 | 3200 | 2.298193339e-15 | 1.076020880e-15 | 2.537619664e-15 | 0.359000 |
| 1000 | 3200 | 6.770477532e-16 | 7.473169406e-15 | 7.503776025e-15 | 0.359000 |
| 10 | 12544 | 5.980413670e-14 | 8.529707832e-16 | 5.981021924e-14 | 4.508000 |
| 100 | 12544 | 6.807948077e-15 | 1.159686971e-15 | 6.906014110e-15 | 4.508000 |
| 1000 | 12544 | 9.233647077e-16 | 4.044067941e-15 | 4.148142704e-15 | 4.509000 |

Πίνακας 4.12: Αποτελέσματα MA57 (MeTiS ordering used)

| Re | DOF | $\|x - x^*\|_1$ | $\|x - x^*\|_2$ | $\|x - x^*\|_\infty$ |
|---|---|---|---|---|
| 10 | 832 | 5.828337812e-12 | 3.700567200e-13 | 4.207745263e-14 |
| 100 | 832 | 3.035238727e-12 | 1.533826449e-13 | 1.509903313e-14 |
| 1000 | 832 | 6.477485215e-12 | 3.507132152e-13 | 6.195044477e-14 |
| 10 | 3200 | 3.838973583e-10 | 1.203624776e-11 | 5.193623309e-13 |
| 100 | 3200 | 3.537881099e-11 | 9.233270557e-13 | 3.419486916e-14 |
| 1000 | 3200 | 2.137523492e-11 | 4.754166723e-13 | 6.405986852e-14 |
| 10 | 12544 | 9.502616583e-09 | 1.485057182e-10 | 2.738476113e-12 |
| 100 | 12544 | 3.323820108e-10 | 4.528505070e-12 | 1.213473766e-13 |
| 1000 | 12544 | 6.639047090e-10 | 1.093690097e-11 | 1.126210236e-12 |

Πίνακας 4.13: Νόρμες σφαλμάτων MA57 (MeTiS ordering used)

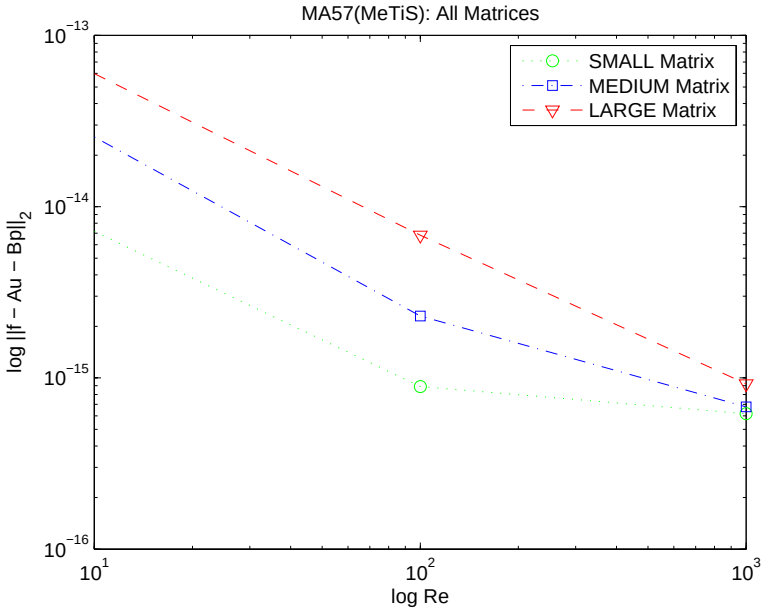
| Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | CPU time (sec) |
|---|---|---|---|---|---|
| 10 | 832 | 7.092991841e-15 | 7.199081525e-16 | 7.129432026e-15 | 0.031000 |
| 100 | 832 | 8.897276174e-16 | 1.263621925e-15 | 1.545430555e-15 | 0.047000 |
| 1000 | 832 | 5.571868293e-16 | 1.150738726e-14 | 1.152086883e-14 | 0.031000 |
| 10 | 3200 | 3.469883898e-14 | 8.386737684e-16 | 3.470897291e-14 | 0.344000 |
| 100 | 3200 | 2.889537128e-15 | 1.177088814e-15 | 3.120090205e-15 | 0.358000 |
| 1000 | 3200 | 6.858316951e-16 | 5.936309913e-15 | 5.975796223e-15 | 0.344000 |
| 10 | 12544 | 1.183663127e-13 | 9.985545094e-16 | 1.183705246e-13 | 4.446000 |
| 100 | 12544 | 1.098536760e-14 | 1.110214893e-15 | 1.104132593e-14 | 4.430000 |
| 1000 | 12544 | 1.366797946e-15 | 3.348224379e-15 | 3.616454496e-15 | 4.493000 |

Πίνακας 4.14: Αποτελέσματα MA57 (AMD ordering used)

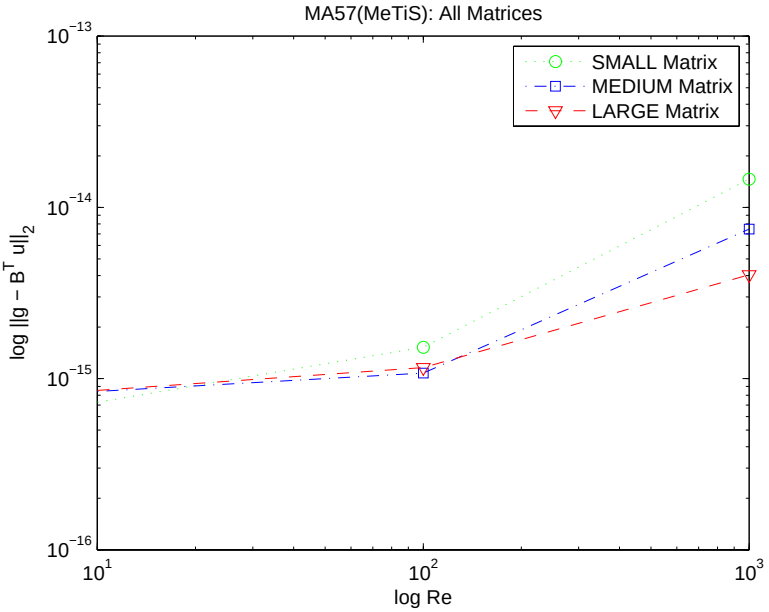
| Re | DOF | $\|x - x^*\|_1$ | $\|x - x^*\|_2$ | $\|x - x^*\|_\infty$ |
|---|---|---|---|---|
| 10 | 832 | 4.429123734e-12 | 3.015239422e-13 | 3.885780586e-14 |
| 100 | 832 | 2.328803816e-12 | 1.221606601e-13 | 2.708944180e-14 |
| 1000 | 832 | 4.945377441e-12 | 2.480480248e-13 | 6.727951529e-14 |
| 10 | 3200 | 5.956202198e-11 | 1.964621600e-12 | 1.936228955e-13 |
| 100 | 3200 | 4.287237232e-11 | 1.099282595e-12 | 6.339373471e-14 |
| 1000 | 3200 | 2.276923095e-11 | 5.794190360e-13 | 1.101341240e-13 |
| 10 | 12544 | 1.009146999e-08 | 1.553307740e-10 | 3.150257832e-12 |
| 100 | 12544 | 1.320733301e-09 | 2.489612498e-11 | 5.920819390e-13 |
| 1000 | 12544 | 6.902972638e-10 | 8.494027702e-12 | 1.277977724e-12 |

Πίνακας 4.15: Νόρμες σφαλμάτων MA57 (AMD ordering used)

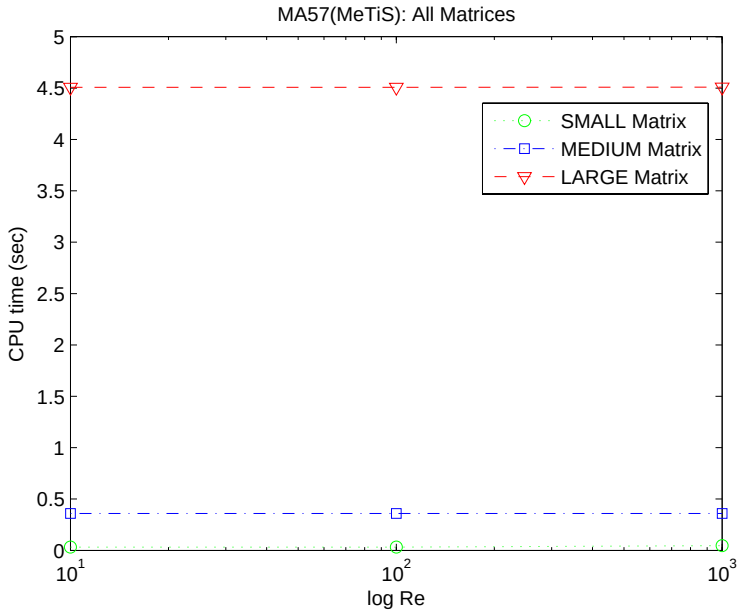
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο MA57:



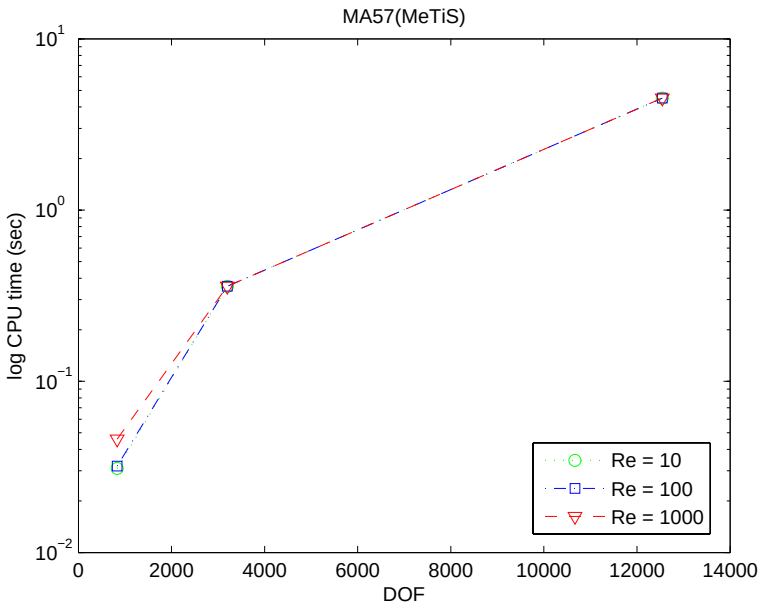

Σχήμα 4.79: Μέθοδος MA57(MeTiS): Re vs $\|f - Au - Bp\|_2$.




Σχήμα 4.80: Μέθοδος MA57(MeTiS): Re vs $\|g - B^T u\|_2$.




Σχήμα 4.81: Μέθοδος MA57(MeTiS): Re vs CPU time.




Σχήμα 4.82: Μέθοδος MA57(MeTiS): DOF vs CPU time.

### 4.3.6 Αποτελέσματα Μεθόδου Preconditioned MINRES

Στους πίνακες (4.16-4.17), παραθέτουμε τα αποτελέσματα για τη μέθοδο Preconditioned MINRES με $(r = 0.22, band = 2)$. Στα σχήματα (4.83)–(4.103) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου Preconditioned MINRES. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου Preconditioned MINRES:

- Για το μεσαίο και το μεγάλο πίνακα και για τιμές της ανοχής TOL= $10^{-4}, 10^{-6}, 10^{-8}$, όσο αυξάνει ο αριθμός Reynolds της ροής, **αυξάνεται** ο απαιτούμενος υπολογιστικός χρόνος, ενώ για τιμές της ανοχής TOL$< 10^{-8}$,όσο αυξάνει ο αριθμός Reynolds της ροής, **μειώνεται** ο απαιτούμενος υπολογιστικός χρόνος (βλέπε σχήματα (4.89) – (4.91) TOL vs CPU,(4.98) – (4.100) Re vs CPU,(4.101) – (4.103) DOF vs CPU).

- Καθώς αυξάνεται η διάσταση του γραμμικού συστήματος, ο απαιτούμενος υπολογιστικός χρόνος αυξάνεται, για όλες τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής TOL, που δοκιμάστηκαν (βλέπε σχήματα (4.101) – (4.103) DOF vs CPU).

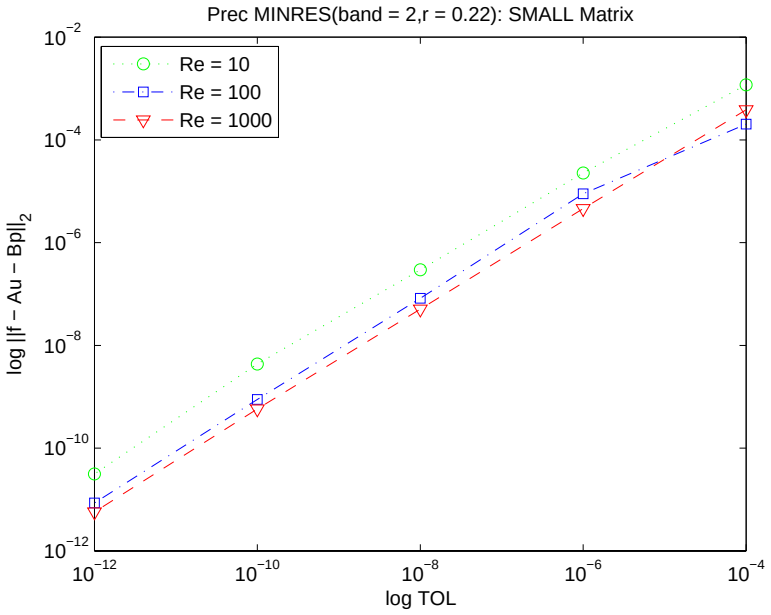
| TOL | Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | #iter | CPU time (sec) |
|---|---|---|---|---|---|---|---|
| 0.1E-03 | 10 | 832 | 0.118404458E-02 | 0.329546700E-02 | 0.350172304E-02 | 144 | 0.0468003 |
| 0.1E-05 | 10 | 832 | 0.227058724E-04 | 0.341573320E-04 | 0.410156064E-04 | 285 | 0.0624004 |
| 0.1E-07 | 10 | 832 | 0.295455975E-06 | 0.340925931E-06 | 0.451137145E-06 | 319 | 0.0780005 |
| 0.1E-09 | 10 | 832 | 0.434209493E-08 | 0.307610875E-08 | 0.532129998E-08 | 357 | 0.0780005 |
| 0.1E-11 | 10 | 832 | 0.314975368E-10 | 0.318310857E-10 | 0.447807195E-10 | 402 | 0.0780005 |
| 0.1E-03 | 100 | 832 | 0.203915871E-03 | 0.249751803E-02 | 0.250582880E-02 | 212 | 0.0780005 |
| 0.1E-05 | 100 | 832 | 0.892344500E-05 | 0.137714911E-04 | 0.164098092E-04 | 314 | 0.0624004 |
| 0.1E-07 | 100 | 832 | 0.828230361E-07 | 0.148355684E-06 | 0.169908988E-06 | 353 | 0.0780005 |
| 0.1E-09 | 100 | 832 | 0.885874475E-09 | 0.188678463E-08 | 0.208440156E-08 | 386 | 0.0780005 |
| 0.1E-11 | 100 | 832 | 0.854760774E-11 | 0.216734959E-10 | 0.232981120E-10 | 413 | 0.0936006 |
| 0.1E-03 | 1000 | 832 | 0.387611595E-03 | 0.192078921E-02 | 0.195950860E-02 | 236 | 0.0468003 |
| 0.1E-05 | 1000 | 832 | 0.460155773E-05 | 0.137909511E-04 | 0.145383860E-04 | 340 | 0.0780005 |
| 0.1E-07 | 1000 | 832 | 0.508400079E-07 | 0.205889855E-06 | 0.212073899E-06 | 415 | 0.0780005 |
| 0.1E-09 | 1000 | 832 | 0.580145557E-09 | 0.262413078E-08 | 0.268749534E-08 | 465 | 0.0936006 |
| 0.1E-11 | 1000 | 832 | 0.573338368E-11 | 0.288255163E-10 | 0.293901698E-10 | 538 | 0.1092007 |
| 0.1E-03 | 10 | 3200 | 0.215956140E-02 | 0.305852861E-02 | 0.374410239E-02 | 88 | 0.3432022 |
| 0.1E-05 | 10 | 3200 | 0.259734866E-04 | 0.842797934E-04 | 0.881913011E-04 | 599 | 0.6240039 |
| 0.1E-07 | 10 | 3200 | 0.404059467E-06 | 0.112777347E-05 | 0.119797206E-05 | 1074 | 0.8580055 |
| 0.1E-09 | 10 | 3200 | 0.540188307E-08 | 0.108829747E-07 | 0.121498757E-07 | 1222 | 0.9516063 |
| 0.1E-11 | 10 | 3200 | 0.611901161E-10 | 0.121147482E-09 | 0.135723774E-09 | 1387 | 1.0140066 |
| 0.1E-03 | 100 | 3200 | 0.397029435E-03 | 0.329554690E-02 | 0.331937671E-02 | 422 | 0.5304031 |
| 0.1E-05 | 100 | 3200 | 0.416432123E-05 | 0.503958957E-04 | 0.505676564E-04 | 924 | 0.7800050 |
| 0.1E-07 | 100 | 3200 | 0.107469429E-06 | 0.394506544E-06 | 0.408882735E-06 | 1031 | 0.8424053 |
| 0.1E-09 | 100 | 3200 | 0.122799669E-08 | 0.388442537E-08 | 0.407390922E-08 | 1085 | 0.8736057 |
| 0.1E-11 | 100 | 3200 | 0.134550853E-10 | 0.378894801E-10 | 0.402076115E-10 | 1137 | 0.9048061 |
| 0.1E-03 | 1000 | 3200 | 0.227072841E-03 | 0.268971990E-02 | 0.269928791E-02 | 452 | 0.5460033 |
| 0.1E-05 | 1000 | 3200 | 0.468464205E-05 | 0.143669351E-04 | 0.151114094E-04 | 896 | 0.7644043 |
| 0.1E-07 | 1000 | 3200 | 0.474188275E-07 | 0.196658768E-06 | 0.202294874E-06 | 1051 | 0.8580055 |
| 0.1E-09 | 1000 | 3200 | 0.500988090E-09 | 0.264628037E-08 | 0.269328589E-08 | 1159 | 0.9048061 |
| 0.1E-11 | 1000 | 3200 | 0.520124960E-11 | 0.240109796E-10 | 0.245678680E-10 | 1310 | 0.9828062 |
| 0.1E-03 | 10 | 12544 | 0.460568494E-02 | 0.509714475E-02 | 0.686973204E-02 | 144 | 4.4928293 |
| 0.1E-05 | 10 | 12544 | 0.185989576E-04 | 0.109659257E-03 | 0.111225330E-03 | 444 | 5.1480331 |
| 0.1E-07 | 10 | 12544 | 0.534822823E-06 | 0.248804306E-05 | 0.254487597E-05 | 2527 | 9.5316620 |
| 0.1E-09 | 10 | 12544 | 0.114309039E-07 | 0.301850873E-07 | 0.322770051E-07 | 4233 | 13.1040840 |
| 0.1E-11 | 10 | 12544 | 0.818587919E-10 | 0.339236697E-09 | 0.348973350E-09 | 4879 | 14.5080910 |
| 0.1E-03 | 100 | 12544 | 0.543855408E-03 | 0.200557689E-02 | 0.207800803E-02 | 207 | 4.6332283 |
| 0.1E-05 | 100 | 12544 | 0.674710093E-05 | 0.812792303E-04 | 0.815587926E-04 | 2069 | 8.5644531 |
| 0.1E-07 | 100 | 12544 | 0.924364595E-07 | 0.983559586E-06 | 0.987893698E-06 | 3099 | 10.7484741 |
| 0.1E-09 | 100 | 12544 | 0.135811843E-08 | 0.106510838E-07 | 0.107373214E-07 | 3959 | 12.5424805 |
| 0.1E-11 | 100 | 12544 | 0.190904836E-10 | 0.102713649E-09 | 0.104472677E-09 | 4303 | 13.2756805 |
| 0.1E-03 | 1000 | 12544 | 0.170236932E-03 | 0.389340448E-02 | 0.389712446E-02 | 1356 | 7.0356445 |
| 0.1E-05 | 1000 | 12544 | 0.454798389E-05 | 0.443292703E-04 | 0.445619609E-04 | 3145 | 10.8420715 |
| 0.1E-07 | 1000 | 12544 | 0.584252562E-07 | 0.245071962E-06 | 0.251940026E-06 | 3396 | 11.3568802 |
| 0.1E-09 | 1000 | 12544 | 0.611885241E-09 | 0.290103235E-08 | 0.296485956E-08 | 3552 | 11.6844635 |
| 0.1E-11 | 1000 | 12544 | 0.564063383E-11 | 0.359582845E-10 | 0.363980078E-10 | 3749 | 12.1212769 |

Πίνακας 4.16: Αποτελέσματα για τη μέθοδο Preconditioned MINRES($r = 0.22, band = 2$)

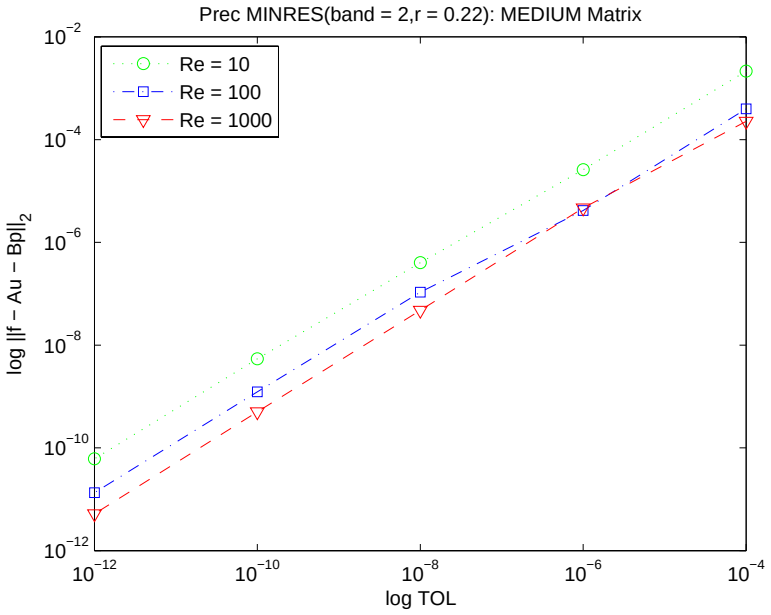
| TOL | Re | DOF | $\|x-x^*\|_1$ | $\|x-x^*\|_2$ | $\|x-x^*\|_\infty$ |
|---|---|---|---|---|---|
| 0.1E-03 | 10 | 832 | 0.128523080E+03 | 0.790390550E+01 | 0.699391296E+00 |
| 0.1E-05 | 10 | 832 | 0.106004875E+01 | 0.676798921E-01 | 0.630215945E-02 |
| 0.1E-07 | 10 | 832 | 0.433923342E-03 | 0.263758168E-04 | 0.416858597E-05 |
| 0.1E-09 | 10 | 832 | 0.549153136E-05 | 0.261174269E-06 | 0.521103692E-07 |
| 0.1E-11 | 10 | 832 | 0.228367202E-07 | 0.116770967E-08 | 0.169366743E-09 |
| 0.1E-03 | 100 | 832 | 0.392223798E+02 | 0.226858952E+01 | 0.229352046E+00 |
| 0.1E-05 | 100 | 832 | 0.370859218E-01 | 0.198494360E-02 | 0.280286416E-03 |
| 0.1E-07 | 100 | 832 | 0.167886785E-03 | 0.918347591E-05 | 0.123427569E-05 |
| 0.1E-09 | 100 | 832 | 0.106849027E-05 | 0.561621035E-07 | 0.796896882E-08 |
| 0.1E-11 | 100 | 832 | 0.932334687E-08 | 0.497486269E-09 | 0.792549359E-10 |
| 0.1E-03 | 1000 | 832 | 0.104861317E+02 | 0.643941177E+00 | 0.114756505E+00 |
| 0.1E-05 | 1000 | 832 | 0.162204438E+00 | 0.110444884E-01 | 0.152876758E-02 |
| 0.1E-07 | 1000 | 832 | 0.118468280E-02 | 0.785009365E-04 | 0.118804410E-04 |
| 0.1E-09 | 1000 | 832 | 0.722561609E-05 | 0.411935369E-06 | 0.539067274E-07 |
| 0.1E-11 | 1000 | 832 | 0.995766776E-07 | 0.577133305E-08 | 0.746958939E-09 |
| 0.1E-03 | 10 | 3200 | 0.104158898E+04 | 0.321776500E+02 | 0.127852575E+01 |
| 0.1E-05 | 10 | 3200 | 0.110827682E+03 | 0.370165062E+01 | 0.167938696E+00 |
| 0.1E-07 | 10 | 3200 | 0.427117758E+00 | 0.142675856E-01 | 0.808885831E-03 |
| 0.1E-09 | 10 | 3200 | 0.161607706E-03 | 0.568440022E-05 | 0.541510151E-06 |
| 0.1E-11 | 10 | 3200 | 0.167154753E-05 | 0.589608953E-07 | 0.931269095E-08 |
| 0.1E-03 | 100 | 3200 | 0.394663495E+03 | 0.115814537E+02 | 0.769508600E+00 |
| 0.1E-05 | 100 | 3200 | 0.795387261E+01 | 0.243817753E+00 | 0.111008785E-01 |
| 0.1E-07 | 100 | 3200 | 0.146896256E-02 | 0.353940978E-04 | 0.264519509E-05 |
| 0.1E-09 | 100 | 3200 | 0.920330366E-05 | 0.221135421E-06 | 0.150033883E-07 |
| 0.1E-11 | 100 | 3200 | 0.954020849E-07 | 0.231127047E-08 | 0.200774508E-09 |
| 0.1E-03 | 1000 | 3200 | 0.201345510E+03 | 0.474562217E+01 | 0.865845674E+00 |
| 0.1E-05 | 1000 | 3200 | 0.162019586E+01 | 0.504033659E-01 | 0.319493368E-02 |
| 0.1E-07 | 1000 | 3200 | 0.741948909E-02 | 0.244122052E-03 | 0.205692706E-04 |
| 0.1E-09 | 1000 | 3200 | 0.672037904E-04 | 0.194127416E-05 | 0.127560940E-06 |
| 0.1E-11 | 1000 | 3200 | 0.357084179E-06 | 0.101926227E-07 | 0.745486117E-09 |
| 0.1E-03 | 10 | 12544 | 0.419909972E+04 | 0.641661560E+02 | 0.180801178E+01 |
| 0.1E-05 | 10 | 12544 | 0.408217262E+04 | 0.636331771E+02 | 0.126695731E+01 |
| 0.1E-07 | 10 | 12544 | 0.380308091E+03 | 0.602069886E+01 | 0.113313490E+00 |
| 0.1E-09 | 10 | 12544 | 0.146651282E+00 | 0.277557570E-02 | 0.934363456E-04 |
| 0.1E-11 | 10 | 12544 | 0.193589057E-03 | 0.348458693E-05 | 0.119741679E-06 |
| 0.1E-03 | 100 | 12544 | 0.434113614E+04 | 0.643268718E+02 | 0.120833221E+01 |
| 0.1E-05 | 100 | 12544 | 0.387298412E+03 | 0.624651226E+01 | 0.134963038E+00 |
| 0.1E-07 | 100 | 12544 | 0.131573781E+01 | 0.226969246E-01 | 0.694272378E-03 |
| 0.1E-09 | 100 | 12544 | 0.747227868E-03 | 0.108062738E-04 | 0.488381928E-06 |
| 0.1E-11 | 100 | 12544 | 0.310921829E-05 | 0.435748405E-07 | 0.228709673E-08 |
| 0.1E-03 | 1000 | 12544 | 0.183080674E+04 | 0.234633038E+02 | 0.353483476E+01 |
| 0.1E-05 | 1000 | 12544 | 0.566372268E+02 | 0.729480708E+00 | 0.312950813E-01 |
| 0.1E-07 | 1000 | 12544 | 0.420465454E-01 | 0.559111962E-03 | 0.248972199E-04 |
| 0.1E-09 | 1000 | 12544 | 0.363325725E-03 | 0.548425765E-05 | 0.246620208E-06 |
| 0.1E-11 | 1000 | 12544 | 0.338065856E-05 | 0.460640920E-07 | 0.144601220E-08 |

Πίνακας 4.17: Νόρμες σφαλμάτων για τη μέθοδο Preconditioned MINRES($r = 0.22, band = 2$)

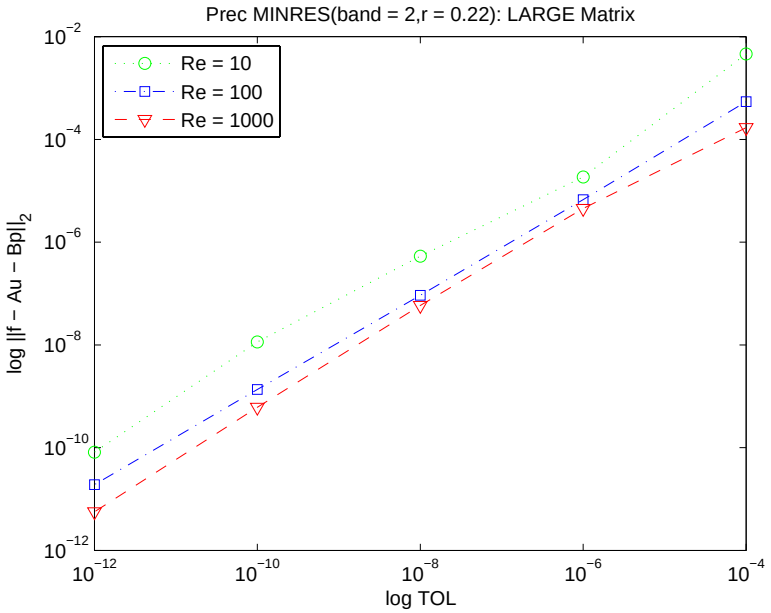
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο Preconditioned MINRES:



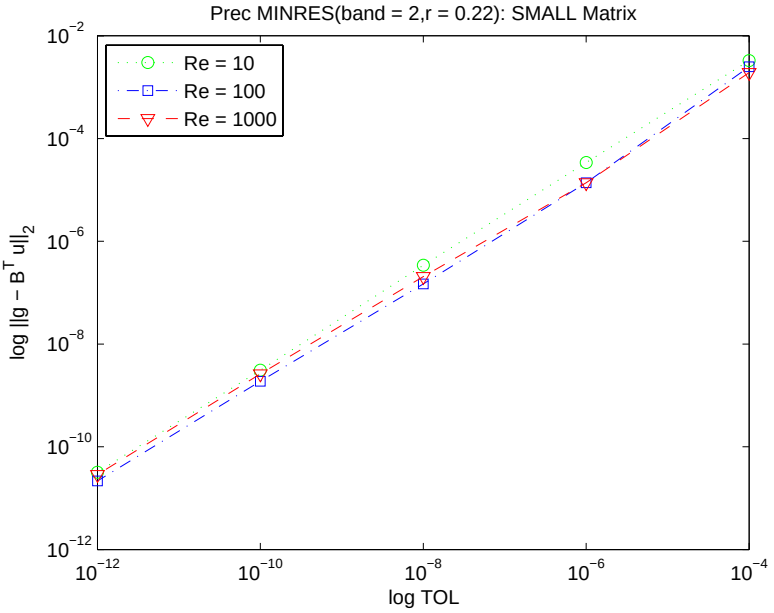

Σχήμα 4.83: Μέθοδος Preconditioned MINRES: TOL vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



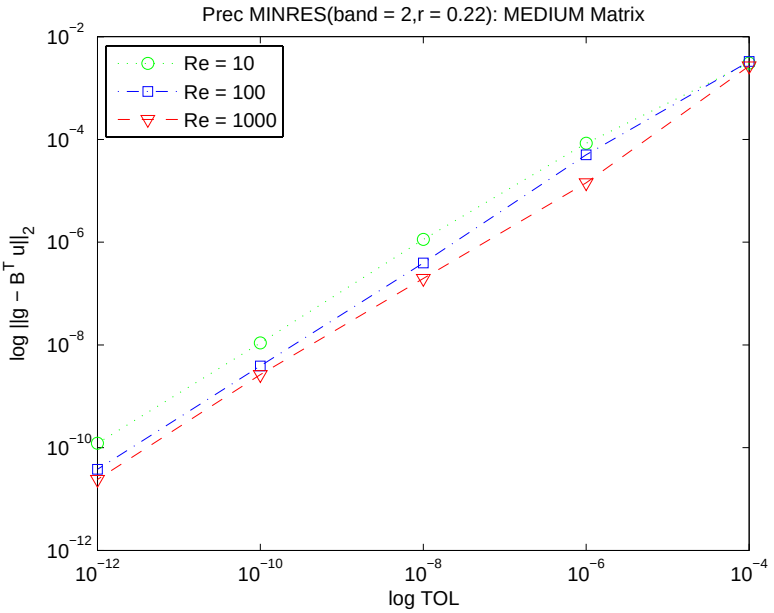

Σχήμα 4.84: Μέθοδος Preconditioned MINRES: TOL vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



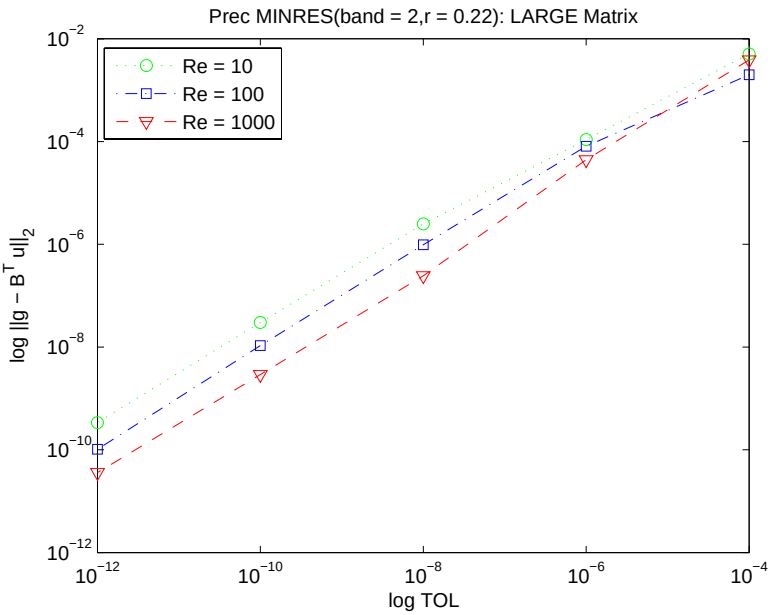

Σχήμα 4.85: Μέθοδος Preconditioned MINRES: TOL vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



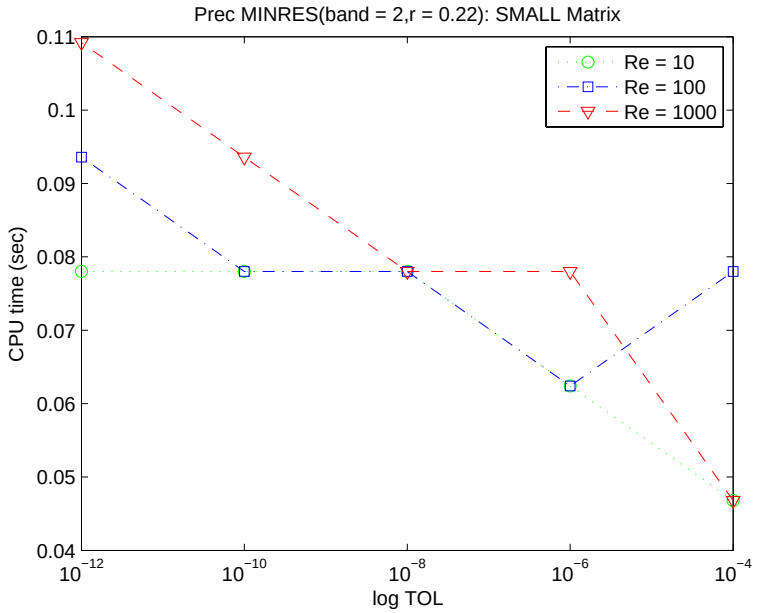

Σχήμα 4.86: Μέθοδος Preconditioned MINRES: TOL vs $\|g - B^T u\|_2$ (SMALL Matrix).



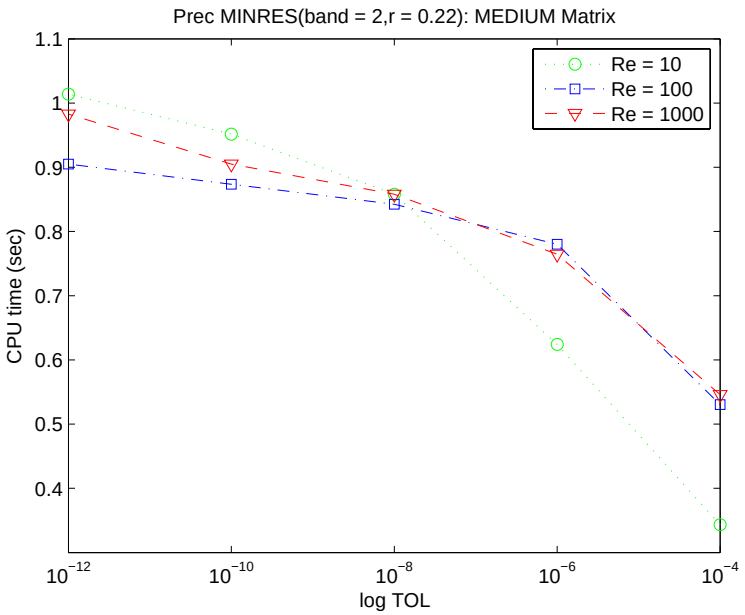

Σχήμα 4.87: Μέθοδος Preconditioned MINRES: TOL vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



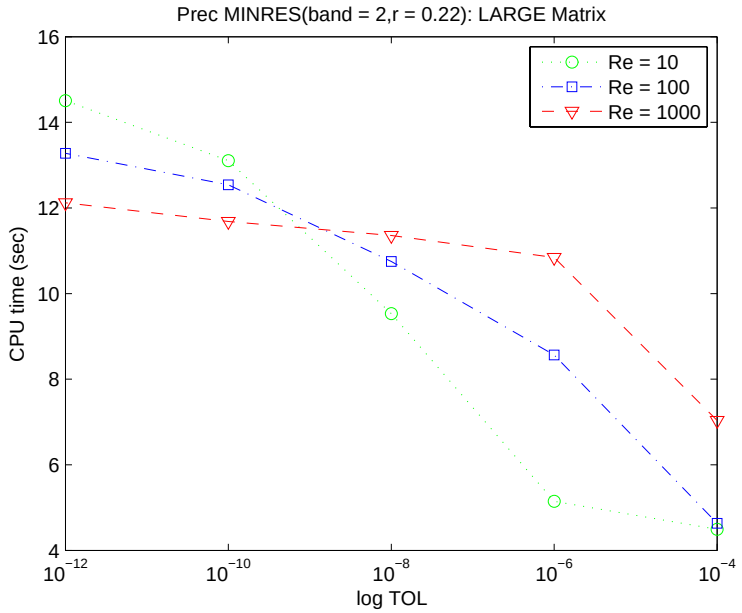

Σχήμα 4.88: Μέθοδος Preconditioned MINRES: TOL vs $\|g - B^T u\|_2$ (LARGE Matrix).



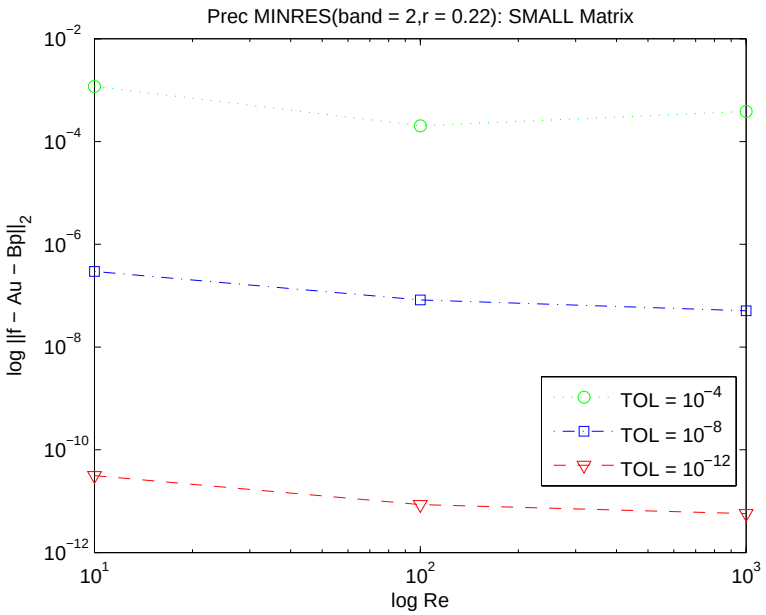

Σχήμα 4.89: Μέθοδος Preconditioned MINRES: TOL vs CPU time, (SMALL Matrix).



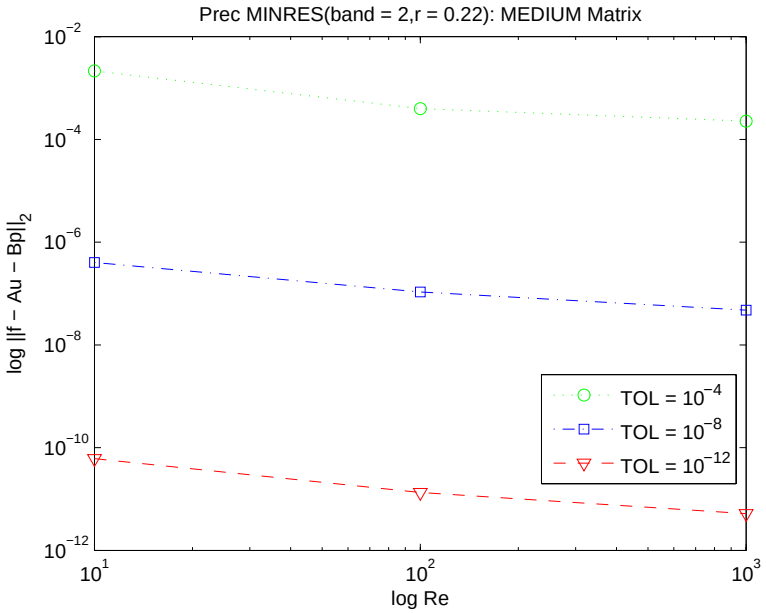

Σχήμα 4.90: Μέθοδος Preconditioned MINRES: TOL vs CPU time, (MEDIUM Matrix).



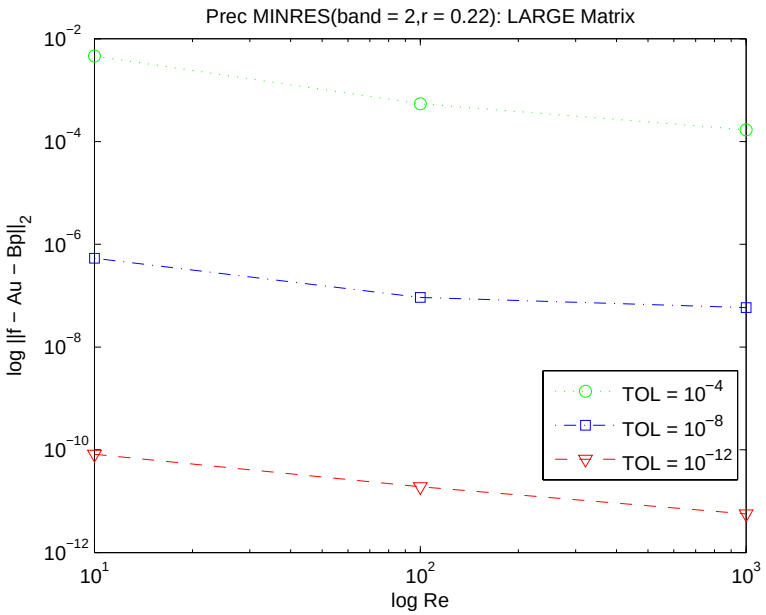

Σχήμα 4.91: Μέθοδος Preconditioned MINRES: TOL vs CPU time, (LARGE Matrix).



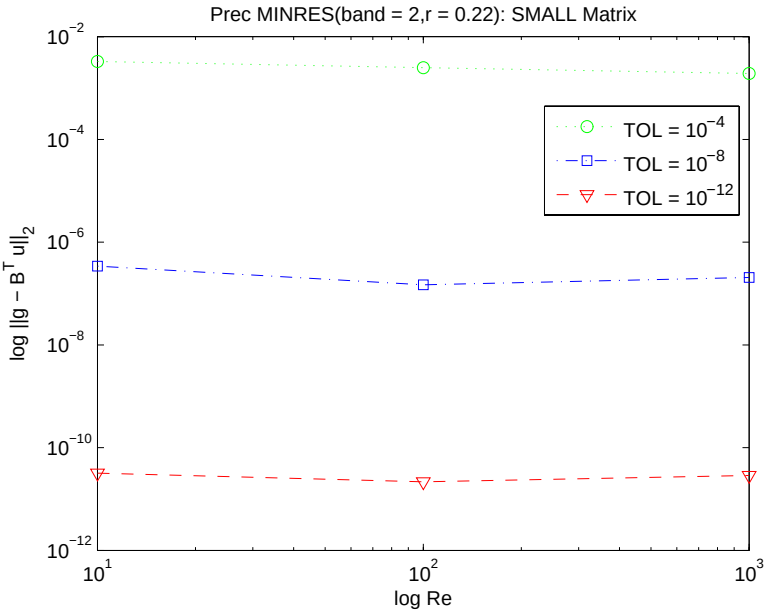

Σχήμα 4.92: Μέθοδος Preconditioned MINRES: Re vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



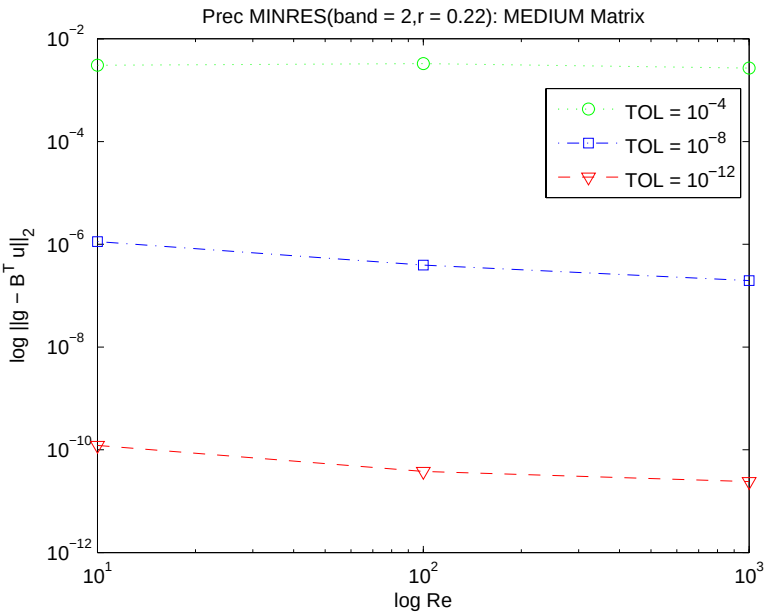

Σχήμα 4.93: Μέθοδος Preconditioned MINRES: Re vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



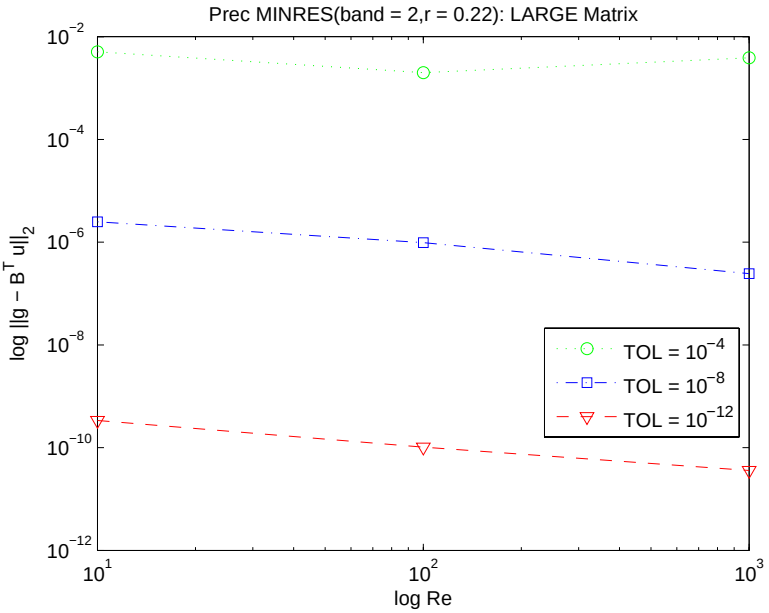

Σχήμα 4.94: Μέθοδος Preconditioned MINRES: Re vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



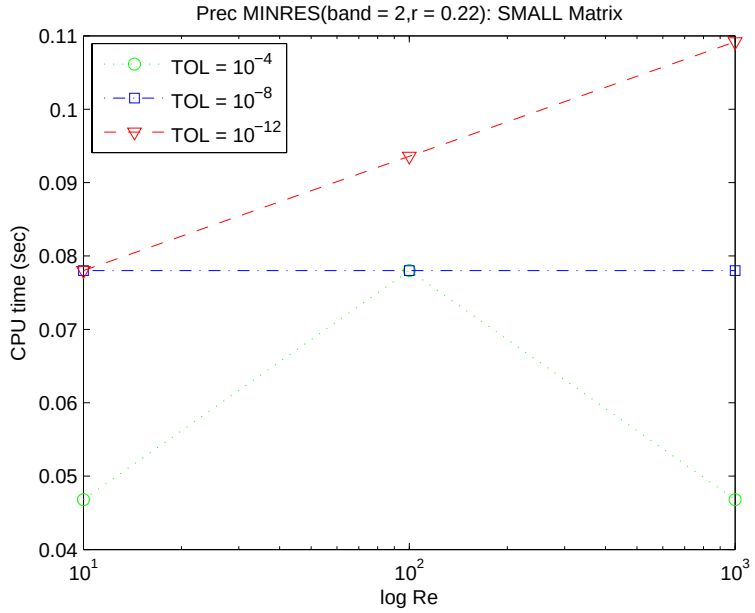

Σχήμα 4.95: Μέθοδος Preconditioned MINRES: Re vs $\|g - B^T u\|_2$ (SMALL Matrix).



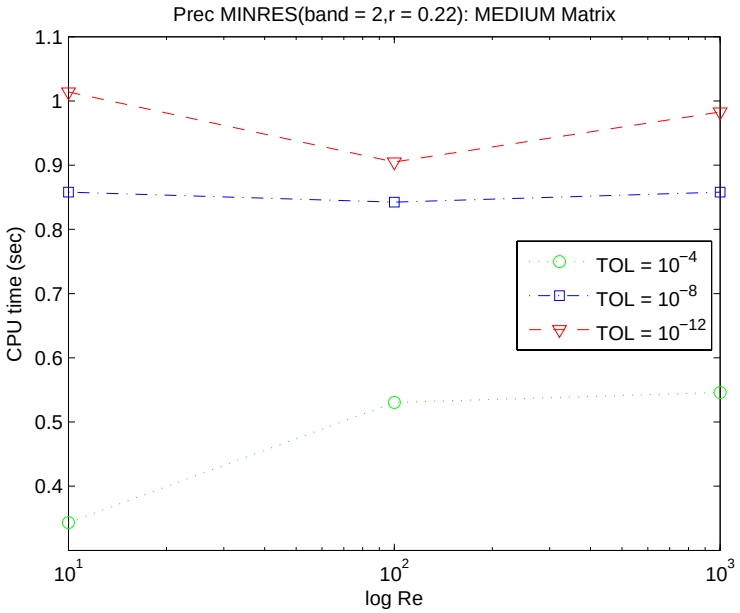

Σχήμα 4.96: Μέθοδος Preconditioned MINRES: Re vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



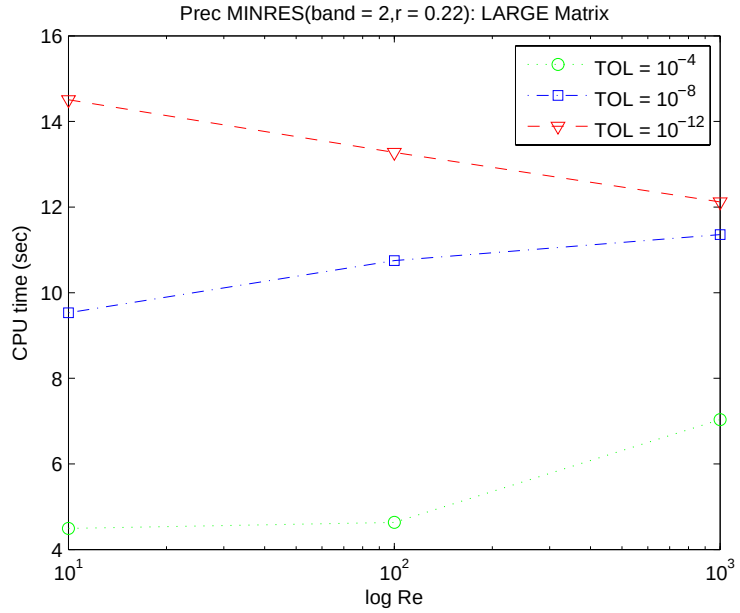

Σχήμα 4.97: Μέθοδος Preconditioned MINRES: Re vs $\|g - B^T u\|_2$ (LARGE Matrix).



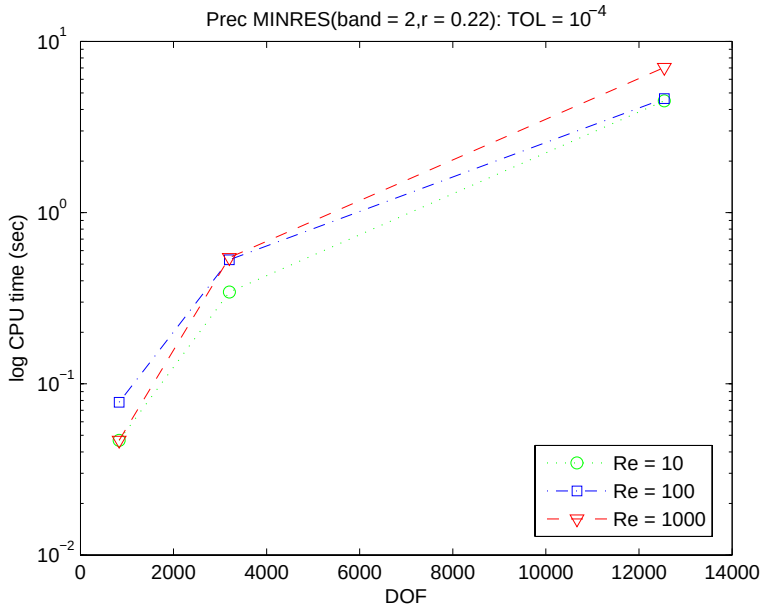

Σχήμα 4.98: Μέθοδος Preconditioned MINRES: Re vs CPU time, (SMALL Matrix).



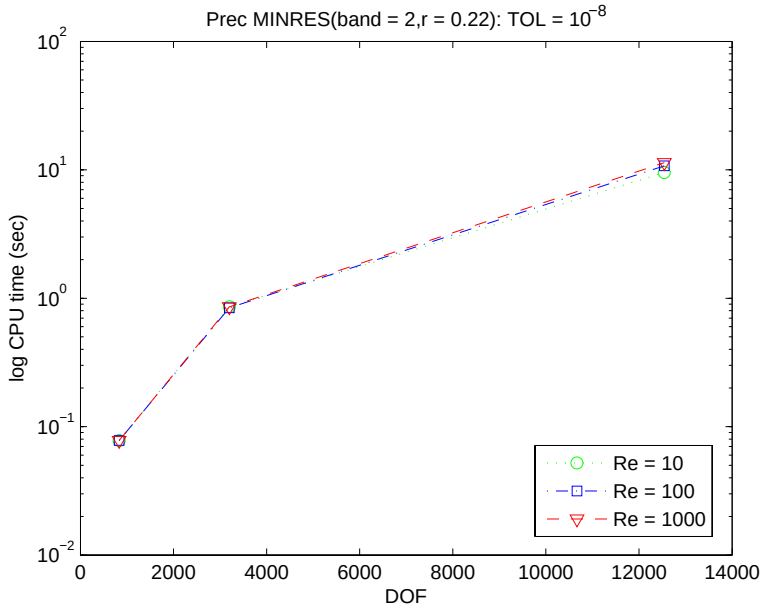

Σχήμα 4.99: Μέθοδος Preconditioned MINRES: Re vs CPU time, (MEDIUM Matrix).



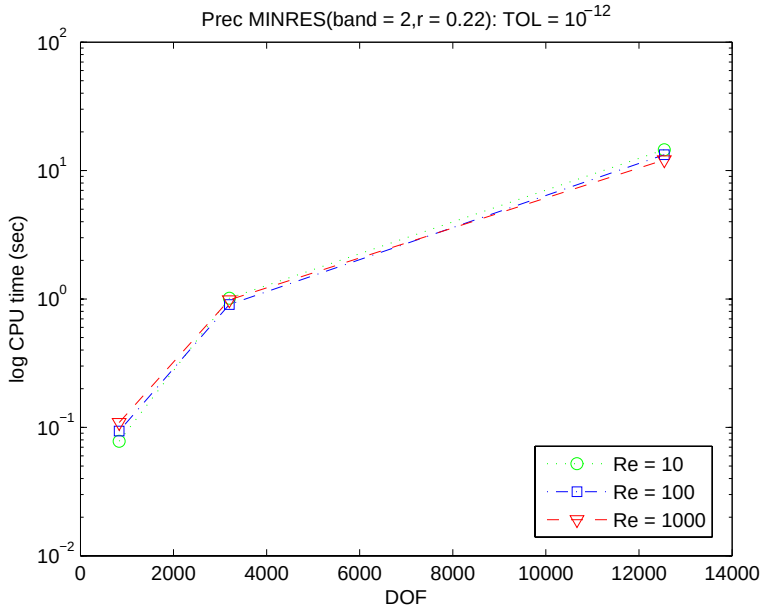

Σχήμα 4.100: Μέθοδος Preconditioned MINRES: Re vs CPU time, (LARGE Matrix).




Σχήμα 4.101: Μέθοδος Preconditioned MINRES: DOF vs CPU time, για TOL $= 10^{-4}$.




Σχήμα 4.102: Μέθοδος Preconditioned MINRES: DOF vs CPU time, για TOL $= 10^{-8}$.




Σχήμα 4.103: Μέθοδος Preconditioned MINRES: DOF vs CPU time, για TOL $= 10^{-12}$.

### 4.3.7 Αποτελέσματα Μεθόδου MINRES

Στους πίνακες (4.18-4.19), παραθέτουμε τα αποτελέσματα για τη μέθοδο MINRES. Στα σχήματα (4.104)–(4.124) βλέπουμε διάφορες χαρακτηριστικές συμπεριφορές της μεθόδου MINRES. Είναι αξιοσημείωτες οι παρακάτω συμπεριφορές της μεθόδου MINRES:

- Η εξάρτηση του αριθμού Reynolds της ροής με τον απαιτούμενο υπολογιστικό χρόνο είναι ανώμαλη (βλέπε σχήματα (4.110) – (4.112) TOL vs CPU,(4.119) – (4.121) Re vs CPU).
- Καθώς αυξάνεται η διάσταση του γραμμικού συστήματος, ο απαιτούμενος υπολογιστικός χρόνος αυξάνεται, για όλες τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής TOL, που δοκιμάστηκαν (βλέπε σχήματα (4.122) – (4.124) DOF vs CPU).

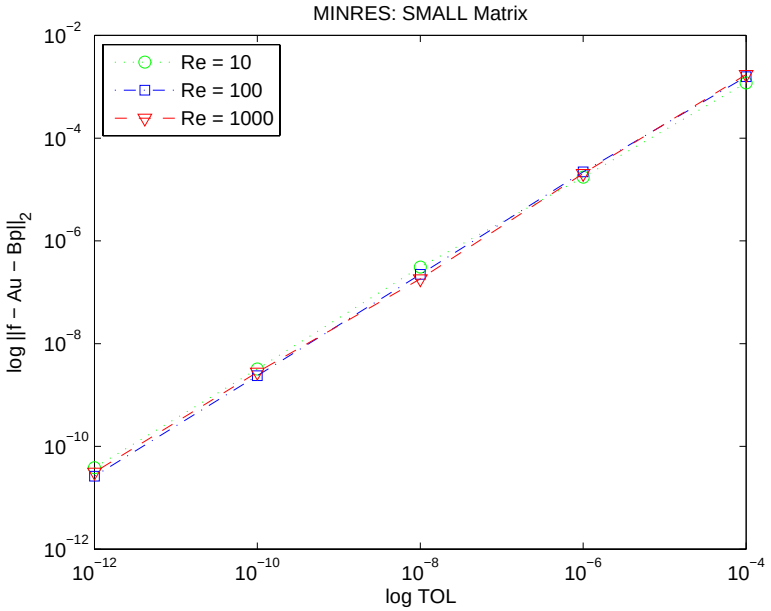
| TOL | Re | DOF | $\|f - Au - Bp\|_2$ | $\|g - B^T u\|_2$ | $\|r\|_2$ | #iter | CPU time (sec) |
|---|---|---|---|---|---|---|---|
| 0.1E-03 | 10 | 832 | 0.119316776E-02 | 0.328010475E-02 | 0.349037770E-02 | 143 | 0.0468003 |
| 0.1E-05 | 10 | 832 | 0.173325220E-04 | 0.430915839E-04 | 0.464467536E-04 | 286 | 0.0624004 |
| 0.1E-07 | 10 | 832 | 0.309056174E-06 | 0.300731527E-06 | 0.431225197E-06 | 322 | 0.0624004 |
| 0.1E-09 | 10 | 832 | 0.320229188E-08 | 0.344797090E-08 | 0.470565369E-08 | 354 | 0.0624004 |
| 0.1E-11 | 10 | 832 | 0.385502093E-10 | 0.361720491E-10 | 0.528633689E-10 | 395 | 0.0624004 |
| 0.1E-03 | 100 | 832 | 0.157219267E-02 | 0.108062581E-02 | 0.190775835E-02 | 197 | 0.0312002 |
| 0.1E-05 | 100 | 832 | 0.221467351E-04 | 0.531432189E-05 | 0.227754230E-04 | 291 | 0.0624004 |
| 0.1E-07 | 100 | 832 | 0.222752708E-06 | 0.107033358E-06 | 0.247133382E-06 | 368 | 0.0624004 |
| 0.1E-09 | 100 | 832 | 0.237199064E-08 | 0.998601104E-09 | 0.257362464E-08 | 409 | 0.0624004 |
| 0.1E-11 | 100 | 832 | 0.262484674E-10 | 0.135981480E-10 | 0.295616587E-10 | 493 | 0.0780005 |
| 0.1E-03 | 1000 | 832 | 0.173145250E-02 | 0.556108668E-03 | 0.181856664E-02 | 176 | 0.0624004 |
| 0.1E-05 | 1000 | 832 | 0.204709001E-04 | 0.582315377E-05 | 0.212830184E-04 | 255 | 0.0468003 |
| 0.1E-07 | 1000 | 832 | 0.182911787E-06 | 0.102179784E-06 | 0.209517136E-06 | 395 | 0.0780005 |
| 0.1E-09 | 1000 | 832 | 0.277400031E-08 | 0.956502585E-09 | 0.293427588E-08 | 603 | 0.0780005 |
| 0.1E-11 | 1000 | 832 | 0.313509057E-10 | 0.125683319E-10 | 0.337763564E-10 | 733 | 0.0936006 |
| 0.1E-03 | 10 | 3200 | 0.199665174E-02 | 0.345862917E-02 | 0.399358660E-02 | 85 | 0.3432022 |
| 0.1E-05 | 10 | 3200 | 0.151607490E-04 | 0.855399839E-04 | 0.868731095E-04 | 598 | 0.5148033 |
| 0.1E-07 | 10 | 3200 | 0.371548312E-06 | 0.102669348E-05 | 0.109185514E-05 | 1072 | 0.7020044 |
| 0.1E-09 | 10 | 3200 | 0.489220946E-08 | 0.106993596E-07 | 0.117647783E-07 | 1215 | 0.7332048 |
| 0.1E-11 | 10 | 3200 | 0.579224880E-10 | 0.100041994E-09 | 0.115600238E-09 | 1338 | 0.7800047 |
| 0.1E-03 | 100 | 3200 | 0.161662739E-02 | 0.113475670E-02 | 0.197513465E-02 | 333 | 0.4368029 |
| 0.1E-05 | 100 | 3200 | 0.263420792E-04 | 0.987534549E-05 | 0.281323228E-04 | 751 | 0.5772038 |
| 0.1E-07 | 100 | 3200 | 0.266087704E-06 | 0.139160841E-06 | 0.300280545E-06 | 921 | 0.6240039 |
| 0.1E-09 | 100 | 3200 | 0.316255289E-08 | 0.102929693E-08 | 0.332583718E-08 | 1009 | 0.6708045 |
| 0.1E-11 | 100 | 3200 | 0.306599881E-10 | 0.195357402E-10 | 0.363549174E-10 | 1219 | 0.7332048 |
| 0.1E-03 | 1000 | 3200 | 0.142263199E-02 | 0.118189296E-02 | 0.184952771E-02 | 312 | 0.4368029 |
| 0.1E-05 | 1000 | 3200 | 0.232779360E-04 | 0.617864014E-05 | 0.240839759E-04 | 581 | 0.5148034 |
| 0.1E-07 | 1000 | 3200 | 0.299794218E-06 | 0.104336837E-06 | 0.317431486E-06 | 1028 | 0.6708040 |
| 0.1E-09 | 1000 | 3200 | 0.375825840E-08 | 0.458970425E-09 | 0.378618014E-08 | 1486 | 0.8268051 |
| 0.1E-11 | 1000 | 3200 | 0.416636560E-10 | 0.116652290E-10 | 0.432658965E-10 | 1959 | 0.9984064 |
| 0.1E-03 | 10 | 12544 | 0.457300363E-02 | 0.650520411E-02 | 0.795173206E-02 | 140 | 4.4148283 |
| 0.1E-05 | 10 | 12544 | 0.164240802E-04 | 0.107535409E-03 | 0.108782419E-03 | 399 | 4.7892313 |
| 0.1E-07 | 10 | 12544 | 0.576048678E-06 | 0.228603318E-05 | 0.235749438E-05 | 2496 | 7.8468494 |
| 0.1E-09 | 10 | 12544 | 0.108406104E-07 | 0.278139776E-07 | 0.298519042E-07 | 4076 | 10.1712646 |
| 0.1E-11 | 10 | 12544 | 0.950328523E-10 | 0.306241881E-09 | 0.320648301E-09 | 4802 | 11.2320747 |
| 0.1E-03 | 100 | 12544 | 0.818122987E-03 | 0.142546015E-02 | 0.164355160E-02 | 251 | 4.5708275 |
| 0.1E-05 | 100 | 12544 | 0.249446555E-04 | 0.271847849E-04 | 0.368950995E-04 | 1759 | 6.7704430 |
| 0.1E-07 | 100 | 12544 | 0.425274735E-06 | 0.194450709E-06 | 0.467621298E-06 | 3043 | 8.6580544 |
| 0.1E-09 | 100 | 12544 | 0.447347780E-08 | 0.186255637E-08 | 0.484573213E-08 | 3299 | 9.0324631 |
| 0.1E-11 | 100 | 12544 | 0.413855705E-10 | 0.294208378E-10 | 0.507774668E-10 | 3484 | 9.2976532 |
| 0.1E-03 | 1000 | 12544 | 0.144902867E-02 | 0.865760423E-03 | 0.168796481E-02 | 467 | 4.8828354 |
| 0.1E-05 | 1000 | 12544 | 0.264549575E-04 | 0.582719797E-05 | 0.270891309E-04 | 1402 | 6.2400360 |
| 0.1E-07 | 1000 | 12544 | 0.345649036E-06 | 0.784130531E-07 | 0.354431746E-06 | 2477 | 7.8312531 |
| 0.1E-09 | 1000 | 12544 | 0.389078216E-08 | 0.105456554E-08 | 0.403116538E-08 | 3254 | 8.9544601 |
| 0.1E-11 | 1000 | 12544 | 0.462025073E-10 | 0.111332105E-10 | 0.475249414E-10 | 4557 | 10.8732681 |

Πίνακας 4.18: Αποτελέσματα για τη μέθοδο MINRES

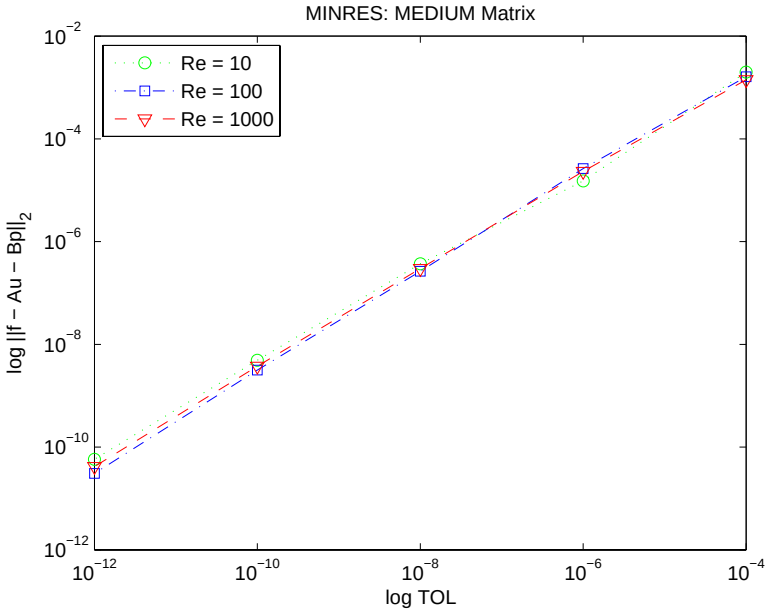
| TOL | Re | DOF | $\|x-x^*\|_1$ | $\|x-x^*\|_2$ | $\|x-x^*\|_\infty$ |
|---|---|---|---|---|---|
| 0.1E-03 | 10 | 832 | 0.127865210E+03 | 0.786552226E+01 | 0.694924877E+00 |
| 0.1E-05 | 10 | 832 | 0.140405228E+01 | 0.895793709E-01 | 0.824451440E-02 |
| 0.1E-07 | 10 | 832 | 0.338879598E-03 | 0.204909530E-04 | 0.378753303E-05 |
| 0.1E-09 | 10 | 832 | 0.425050565E-05 | 0.204317238E-06 | 0.373837514E-07 |
| 0.1E-11 | 10 | 832 | 0.353234763E-07 | 0.181492644E-08 | 0.332650352E-09 |
| 0.1E-03 | 100 | 832 | 0.186008321E+02 | 0.874927801E+00 | 0.788860417E-01 |
| 0.1E-05 | 100 | 832 | 0.167387348E+00 | 0.994617150E-02 | 0.103398522E-02 |
| 0.1E-07 | 100 | 832 | 0.589581247E-03 | 0.351235787E-04 | 0.540983195E-05 |
| 0.1E-09 | 100 | 832 | 0.547883069E-05 | 0.317682296E-06 | 0.431249174E-07 |
| 0.1E-11 | 100 | 832 | 0.621333727E-07 | 0.342835744E-08 | 0.436559455E-09 |
| 0.1E-03 | 1000 | 832 | 0.963659453E+01 | 0.610101460E+00 | 0.140165438E+00 |
| 0.1E-05 | 1000 | 832 | 0.657816905E+00 | 0.418773700E-01 | 0.550952547E-02 |
| 0.1E-07 | 1000 | 832 | 0.700469467E-02 | 0.441635884E-03 | 0.575374967E-04 |
| 0.1E-09 | 1000 | 832 | 0.684357170E-04 | 0.396151111E-05 | 0.438947060E-06 |
| 0.1E-11 | 1000 | 832 | 0.491371477E-06 | 0.290402806E-07 | 0.401603473E-08 |
| 0.1E-03 | 10 | 3200 | 0.104220224E+04 | 0.322084531E+02 | 0.129629750E+01 |
| 0.1E-05 | 10 | 3200 | 0.110862393E+03 | 0.369528961E+01 | 0.168000635E+00 |
| 0.1E-07 | 10 | 3200 | 0.354454566E+00 | 0.118515755E-01 | 0.689762056E-03 |
| 0.1E-09 | 10 | 3200 | 0.158764229E-03 | 0.581929306E-05 | 0.510046182E-06 |
| 0.1E-11 | 10 | 3200 | 0.140081034E-05 | 0.479997975E-07 | 0.705888858E-08 |
| 0.1E-03 | 100 | 3200 | 0.133081242E+03 | 0.487244063E+01 | 0.228840289E+00 |
| 0.1E-05 | 100 | 3200 | 0.264283708E+01 | 0.600843291E-01 | 0.206278609E-02 |
| 0.1E-07 | 100 | 3200 | 0.346507800E-02 | 0.964674048E-04 | 0.690400488E-05 |
| 0.1E-09 | 100 | 3200 | 0.562047539E-04 | 0.167407736E-05 | 0.106986116E-06 |
| 0.1E-11 | 100 | 3200 | 0.503038811E-06 | 0.125327684E-07 | 0.888491059E-09 |
| 0.1E-03 | 1000 | 3200 | 0.816405668E+02 | 0.207879325E+01 | 0.347796371E+00 |
| 0.1E-05 | 1000 | 3200 | 0.433008375E+01 | 0.139714636E+00 | 0.104692198E-01 |
| 0.1E-07 | 1000 | 3200 | 0.501646416E-01 | 0.160206920E-02 | 0.125449731E-03 |
| 0.1E-09 | 1000 | 3200 | 0.214931754E-02 | 0.639717606E-04 | 0.333989662E-05 |
| 0.1E-11 | 1000 | 3200 | 0.657421397E-05 | 0.192669549E-06 | 0.130895939E-07 |
| 0.1E-03 | 10 | 12544 | 0.421546747E+04 | 0.642154837E+02 | 0.191726021E+01 |
| 0.1E-05 | 10 | 12544 | 0.407975843E+04 | 0.635982422E+02 | 0.126896793E+01 |
| 0.1E-07 | 10 | 12544 | 0.380027937E+03 | 0.601481563E+01 | 0.113128600E+00 |
| 0.1E-09 | 10 | 12544 | 0.125454631E+00 | 0.236881658E-02 | 0.812673020E-04 |
| 0.1E-11 | 10 | 12544 | 0.175464609E-03 | 0.316572891E-05 | 0.108924664E-06 |
| 0.1E-03 | 100 | 12544 | 0.437463550E+04 | 0.660196871E+02 | 0.127590432E+01 |
| 0.1E-05 | 100 | 12544 | 0.408262936E+03 | 0.641253550E+01 | 0.121504941E+00 |
| 0.1E-07 | 100 | 12544 | 0.282831826E+00 | 0.373742794E-02 | 0.732080787E-04 |
| 0.1E-09 | 100 | 12544 | 0.674530970E-03 | 0.977242902E-05 | 0.286657552E-06 |
| 0.1E-11 | 100 | 12544 | 0.628733012E-05 | 0.812039961E-07 | 0.276910495E-08 |
| 0.1E-03 | 1000 | 12544 | 0.574167299E+03 | 0.831017140E+01 | 0.583788894E+00 |
| 0.1E-05 | 1000 | 12544 | 0.329781817E+02 | 0.517578860E+00 | 0.202440208E-01 |
| 0.1E-07 | 1000 | 12544 | 0.550969895E+00 | 0.867246585E-02 | 0.294683903E-03 |
| 0.1E-09 | 1000 | 12544 | 0.373185054E-02 | 0.580520062E-04 | 0.231482525E-05 |
| 0.1E-11 | 1000 | 12544 | 0.159711054E-03 | 0.236135139E-05 | 0.638879977E-07 |

Πίνακας 4.19: Νόρμες σφαλμάτων για τη μέθοδο MINRES

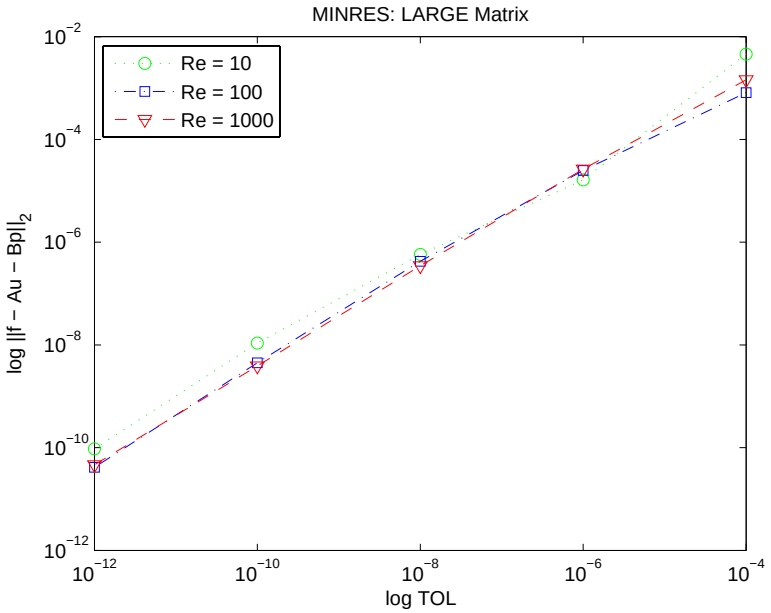
Στη συνέχεια παραθέτουμε μια σειρά γραφικών παραστάσεων για τη μέθοδο MINRES:



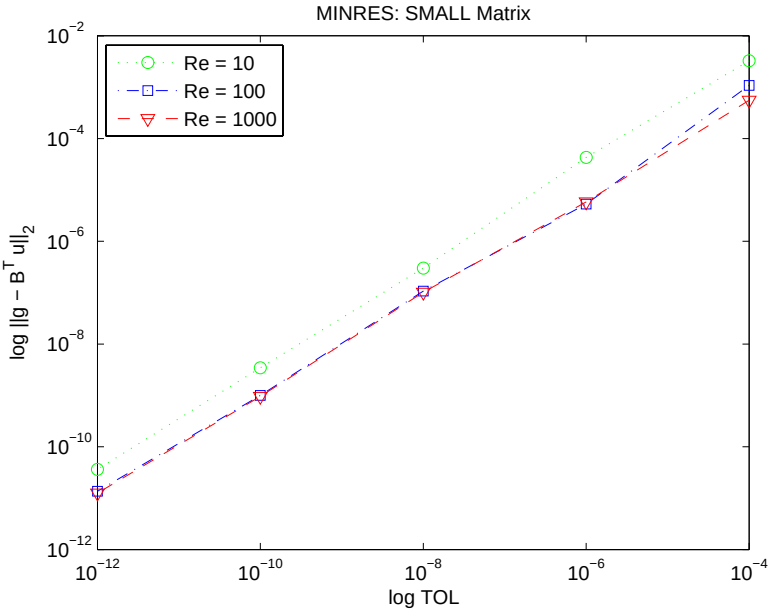

Σχήμα 4.104: Μέθοδος MINRES: TOL vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



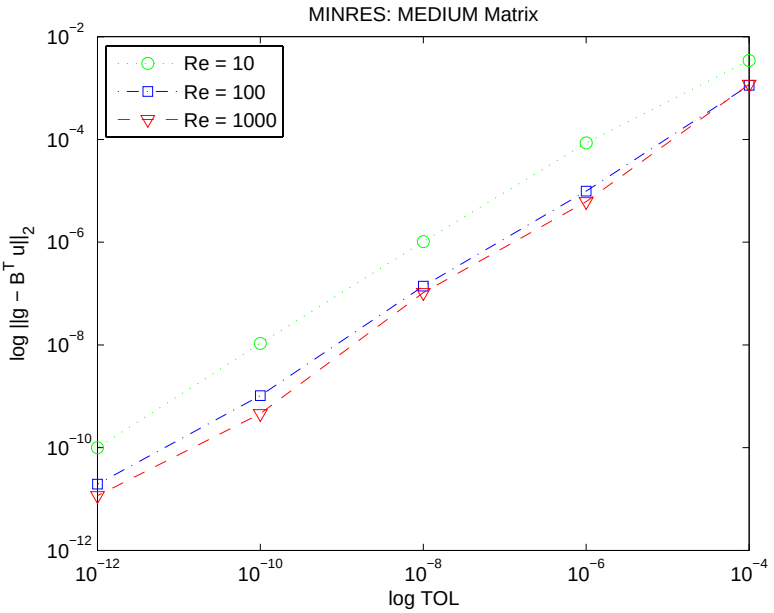

Σχήμα 4.105: Μέθοδος MINRES: TOL vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



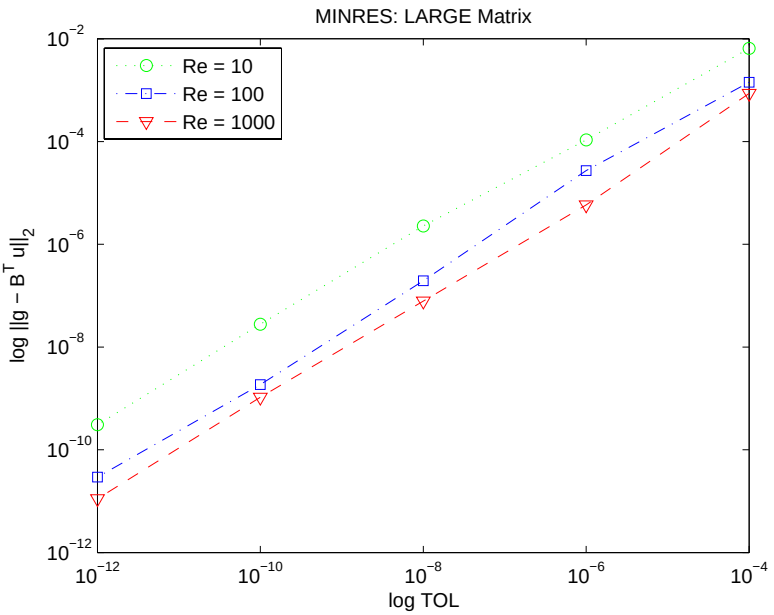

Σχήμα 4.106: Μέθοδος MINRES: TOL vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



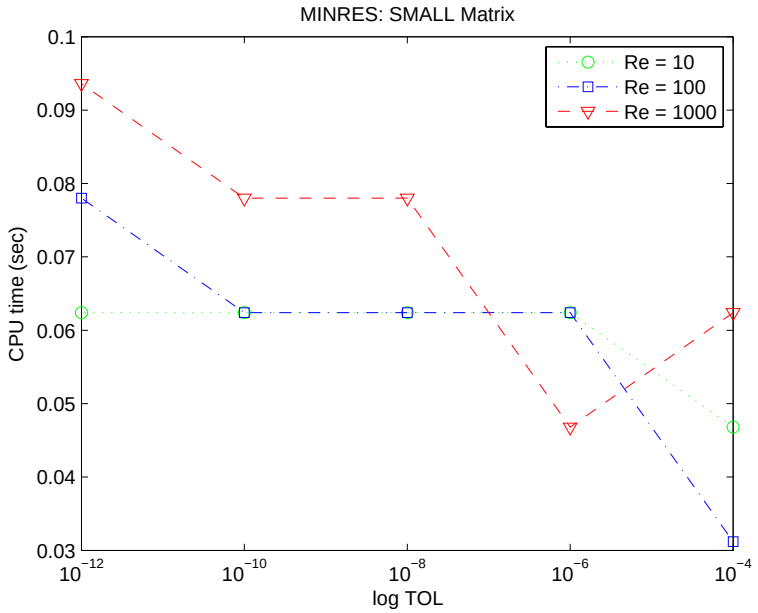

Σχήμα 4.107: Μέθοδος MINRES: TOL vs $\|g - B^T u\|_2$ (SMALL Matrix).



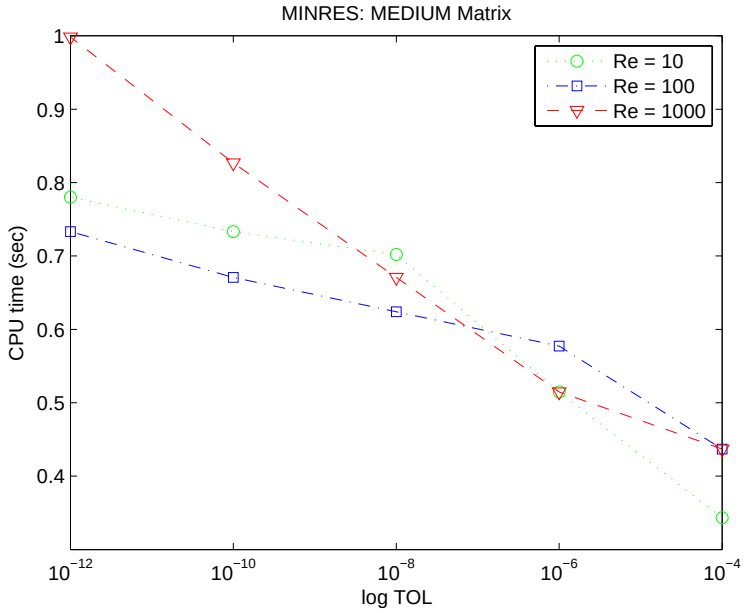

Σχήμα 4.108: Μέθοδος MINRES: TOL vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



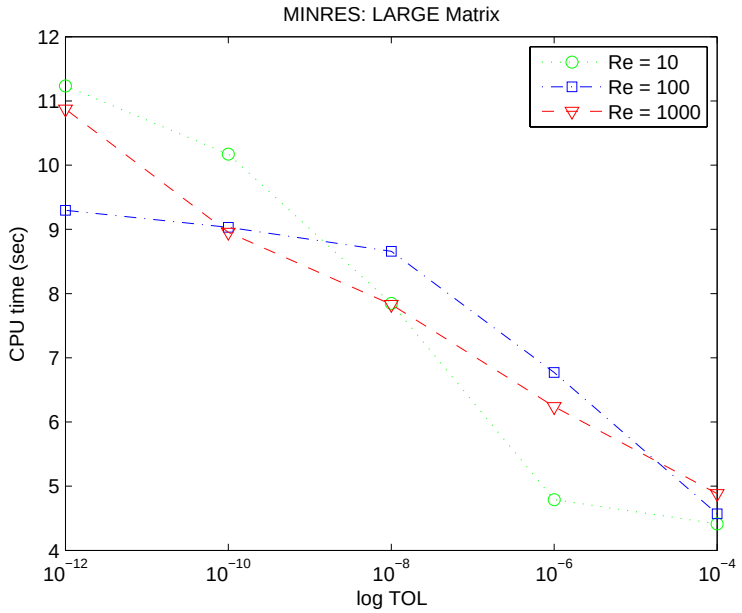

Σχήμα 4.109: Μέθοδος MINRES: TOL vs $\|g - B^T u\|_2$ (LARGE Matrix).



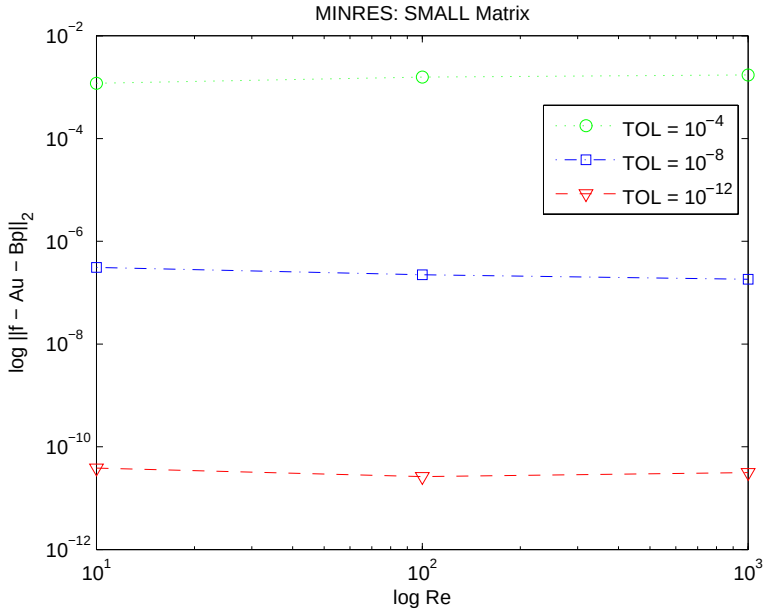

Σχήμα 4.110: Μέθοδος MINRES: TOL vs CPU time, (SMALL Matrix).



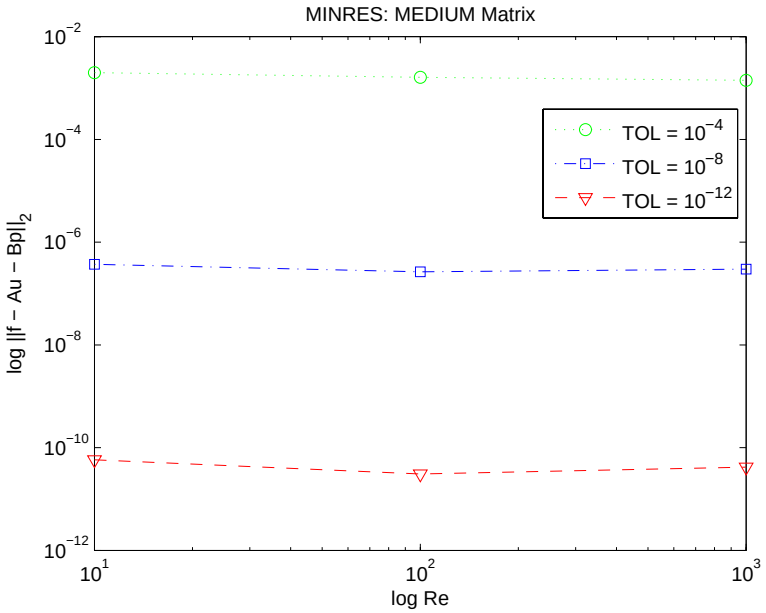

Σχήμα 4.111: Μέθοδος MINRES: TOL vs CPU time, (MEDIUM Matrix).



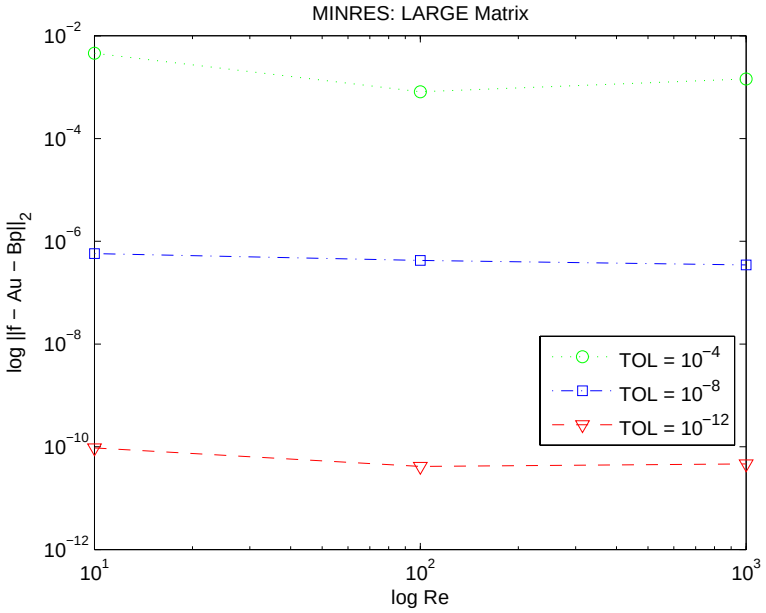

Σχήμα 4.112: Μέθοδος MINRES: TOL vs CPU time, (LARGE Matrix).



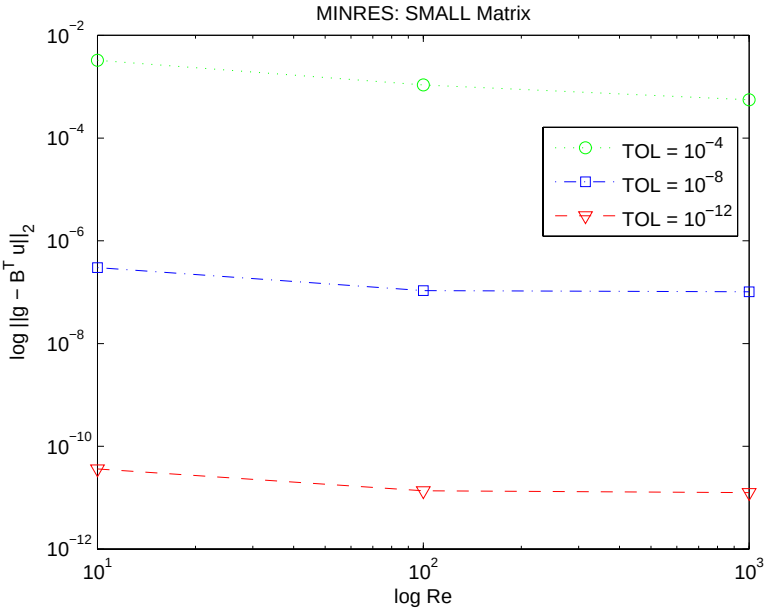

Σχήμα 4.113: Μέθοδος MINRES: Re vs $\|f - Au - Bp\|_2$ (SMALL Matrix).



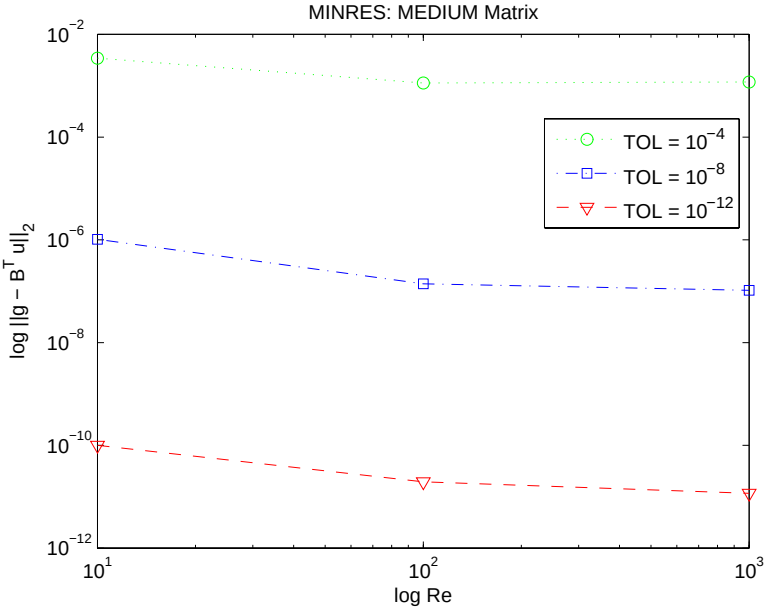

Σχήμα 4.114: Μέθοδος MINRES: Re vs $\|f - Au - Bp\|_2$ (MEDIUM Matrix).



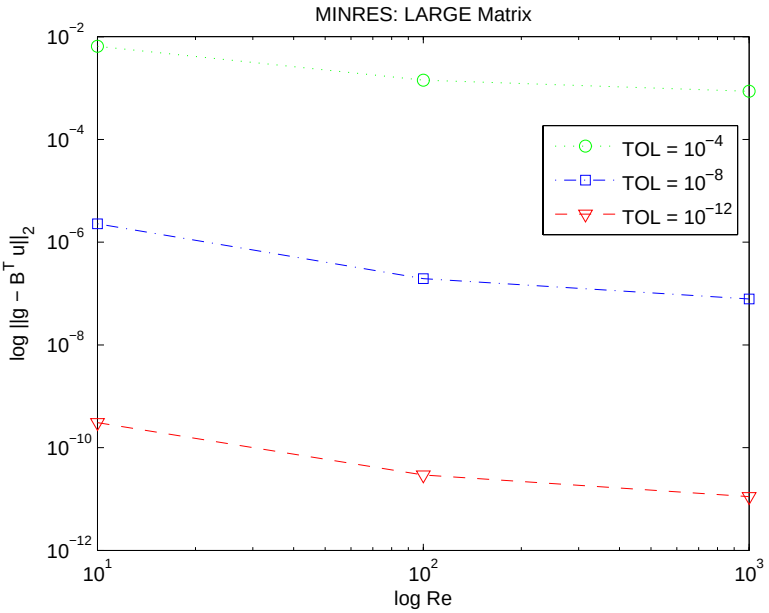

Σχήμα 4.115: Μέθοδος MINRES: Re vs $\|f - Au - Bp\|_2$ (LARGE Matrix).



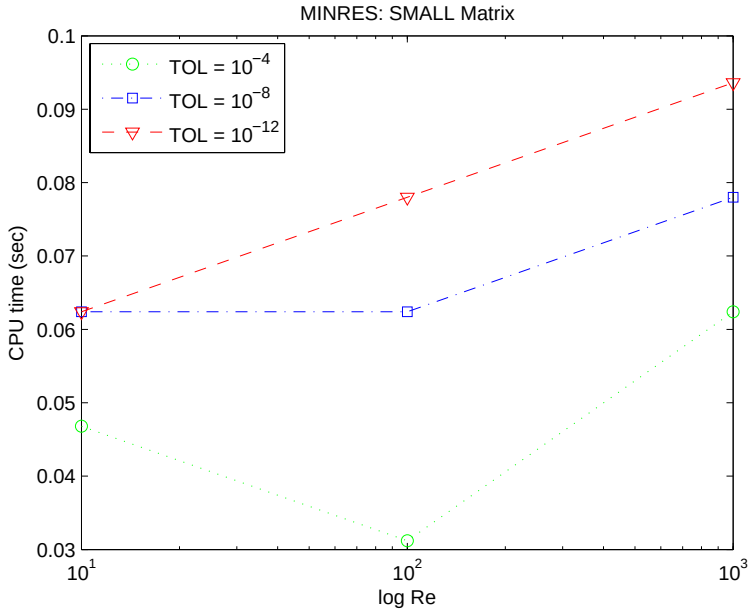

Σχήμα 4.116: Μέθοδος MINRES: Re vs $\|g - B^T u\|_2$ (SMALL Matrix).



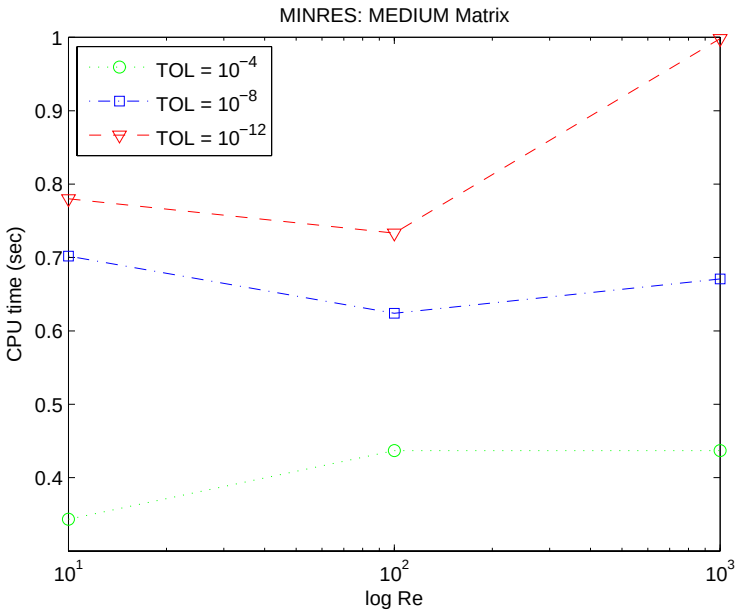

Σχήμα 4.117: Μέθοδος MINRES: Re vs $\|g - B^T u\|_2$ (MEDIUM Matrix).



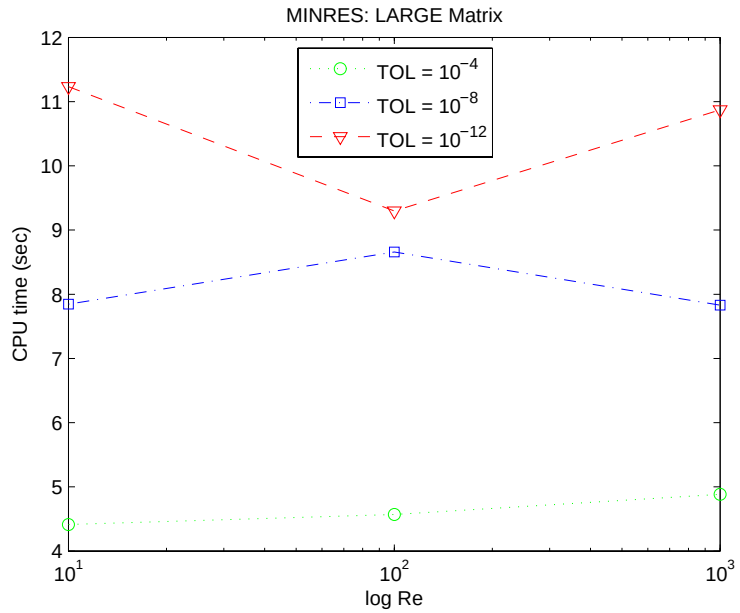

Σχήμα 4.118: Μέθοδος MINRES: Re vs $\|g - B^T u\|_2$ (LARGE Matrix).



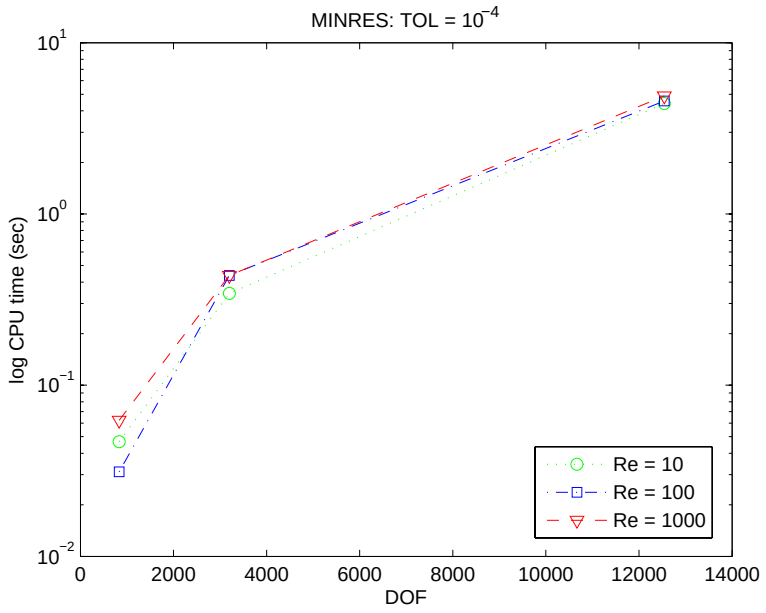

Σχήμα 4.119: Μέθοδος MINRES: Re vs CPU time, (SMALL Matrix).



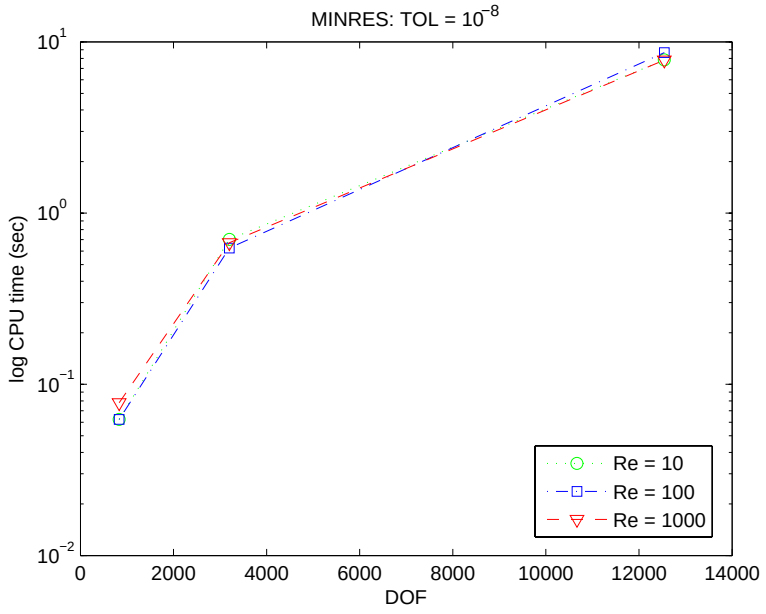

Σχήμα 4.120: Μέθοδος MINRES: Re vs CPU time, (MEDIUM Matrix).



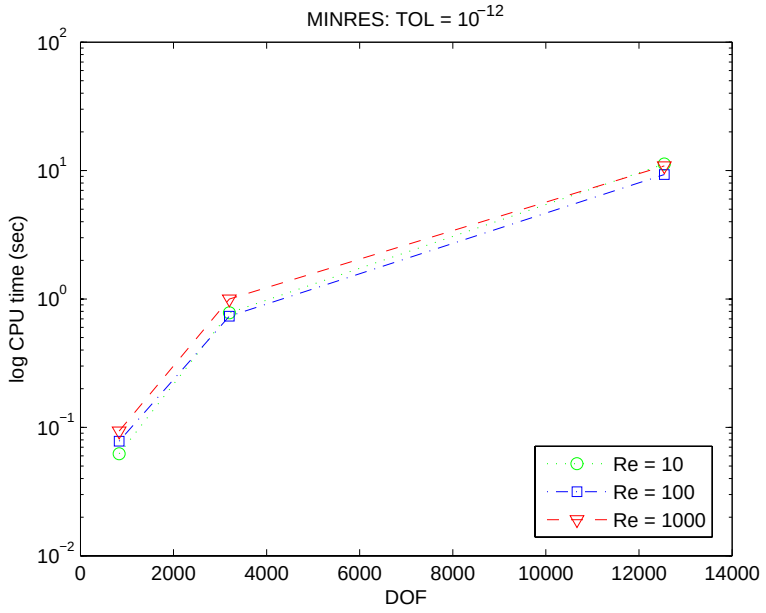

Σχήμα 4.121: Μέθοδος MINRES: Re vs CPU time, (LARGE Matrix).




Σχήμα 4.122: Μέθοδος MINRES: DOF vs CPU time, για TOL $= 10^{-4}$.




Σχήμα 4.123: Μέθοδος MINRES: DOF vs CPU time, για TOL $= 10^{-8}$.




Σχήμα 4.124: Μέθοδος MINRES: DOF vs CPU time, για TOL $= 10^{-12}$.

### 4.3.8 Συγκεντρωτικά Αποτελέσματα

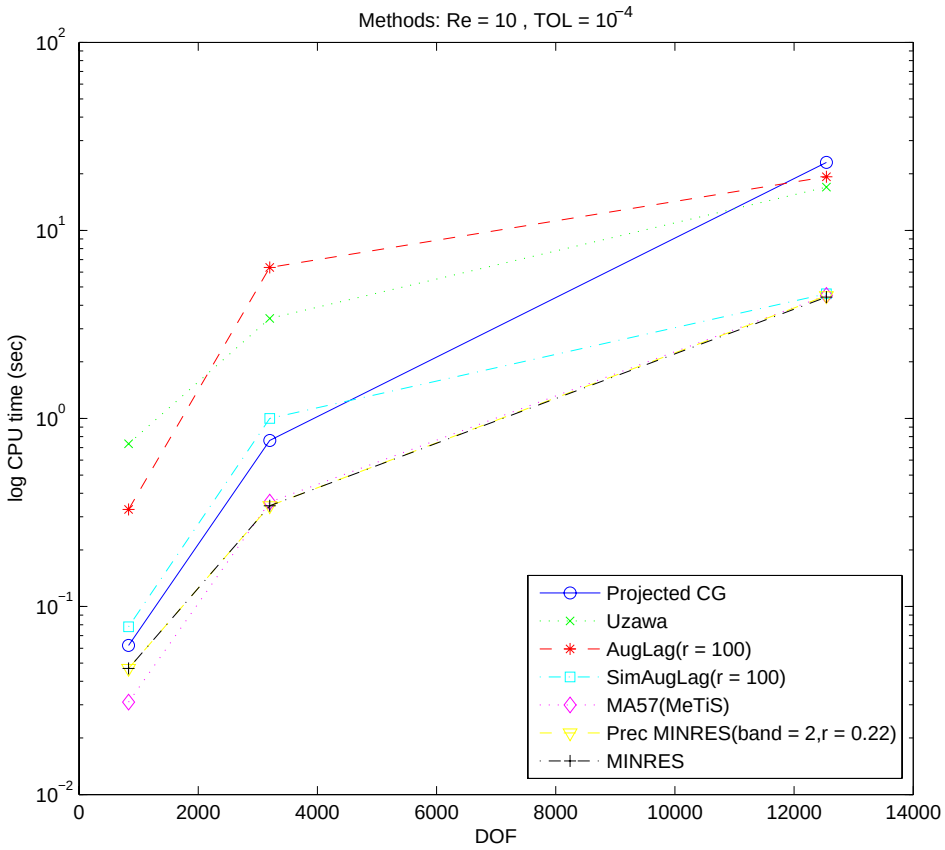
Όπως παρατηρούμε στα γραφήματα (4.125) – (4.133) και στους αντίστοιχους πίνακες (4.4) – (4.19), γρηγορότερη μέθοδος σε κάθε περίπτωση αποδεικνύεται η άμεση μέθοδος MA57, ενώ πολύ κοντά σ΄ αυτή φιγουράρουν οι MINRES και Preconditioned MINRES ($band = 2, r = 0.22$). Ακολουθεί, με σημαντική όμως διαφορά, η Projected CG.

Στη συνέχεια έρχεται η μέθοδος Simplified Augmented Lagrangian($r = 100$), η οποία, ίσως με κάποια καλύτερη επιλογή της παραμέτρου $r$ να μπορεί να γίνει ανταγωνιστικότερη.

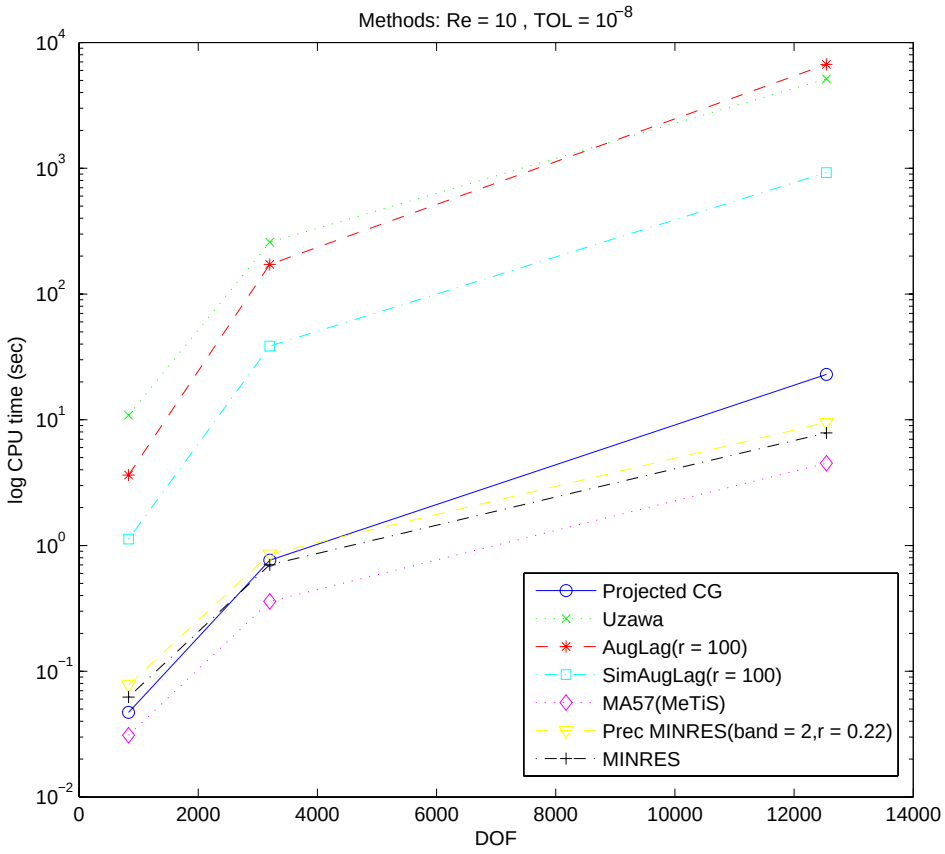
Τελευταίες στην κατάταξη έρχονται οι μέθοδοι Augmented Lagrangian($r = 100$) και Uzawa, οι οποίες αποδεικνύονται εξαιρετικά χρονοβόρες, σε σημείο απαγορευτικό, ειδικά για το μεγάλο πίνακα της δοκιμής.

Για την τιμή της ανοχής $TOL = 10^{-4}$, η μέθοδος Projected CG φαίνεται να είναι η πιο αργή. Ο λόγος είναι ότι η Projected CG σε όλες τις περιπτώσεις της δοκιμής, εκτελεί μία μόνο επανάληψη, στην οποία καταφέρνει να ρίξει το υπόλοιπο σε τάξη $10^{-13}$ – $10^{-16}$. Επομένως, για κάθε κατηγορία διάστασης, η Projected CG χρειάζεται ακριβώς τον ίδιο υπολογιστικό χρόνο, για όλες τις τιμές του αριθμού Reynolds της ροής και για όλες τις τιμές της ανοχής $TOL$, που δοκιμάστηκαν.

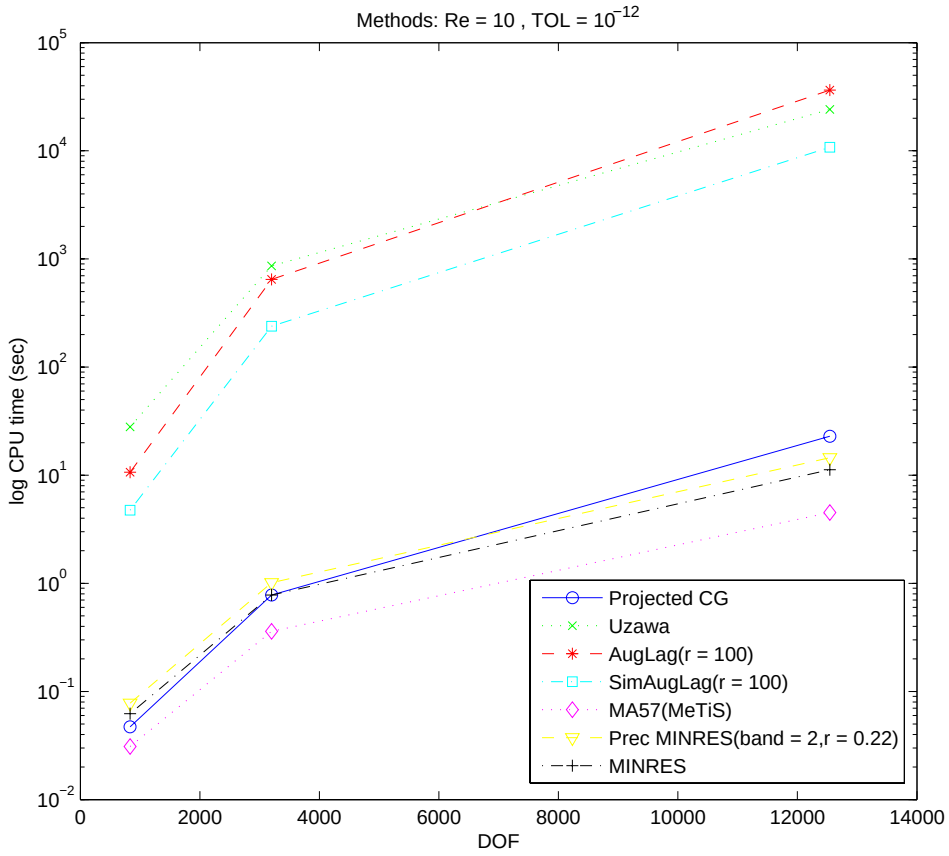
Για τον παραπάνω λόγο, για μικρότερες τιμές της ανοχής $TOL$, η Projected CG φιγουράρει ανάμεσα στις πιο γρήγορες μεθόδους, για κάθε κατηγορία διάστασης.



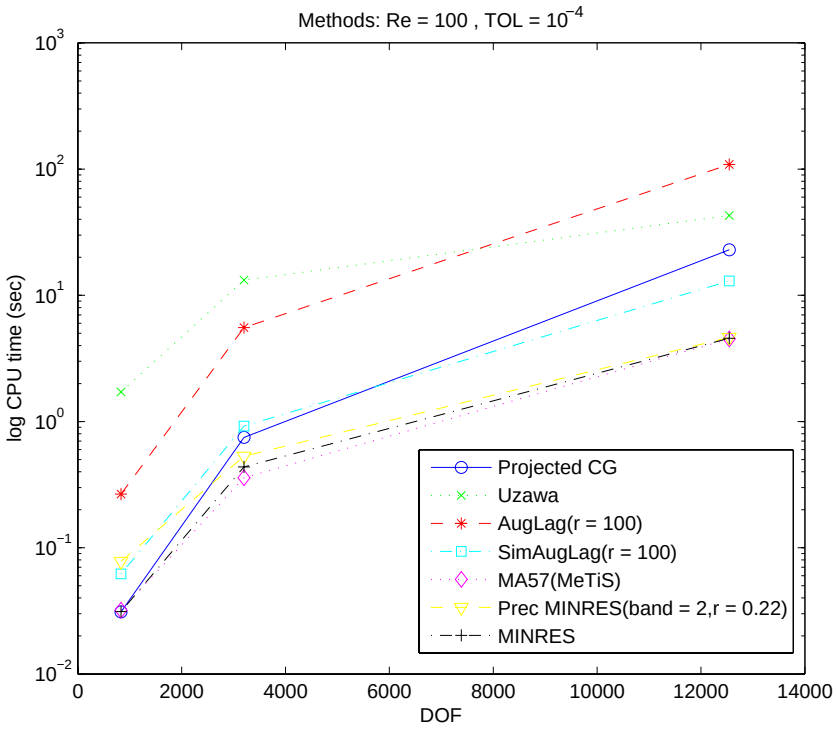

Σχήμα 4.125: Συγκεντρωτικό: DOF vs CPU time, για (Re = 10,TOL = $10^{-4}$).



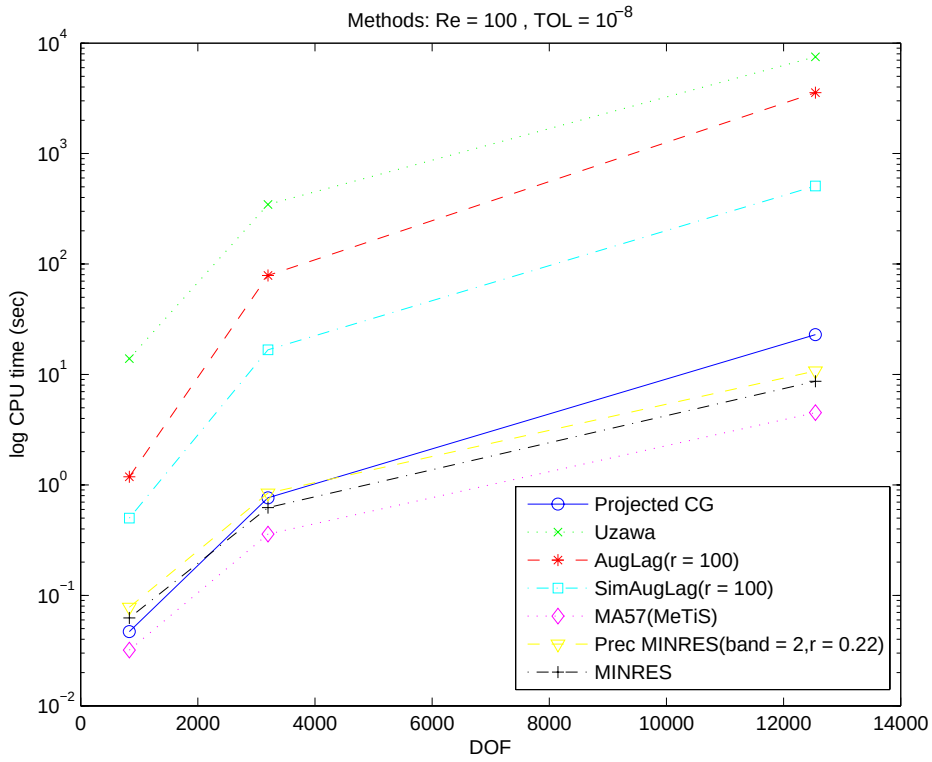

Σχήμα 4.126: Συγκεντρωτικό: DOF vs CPU time, για (Re = 10,TOL = $10^{-8}$).



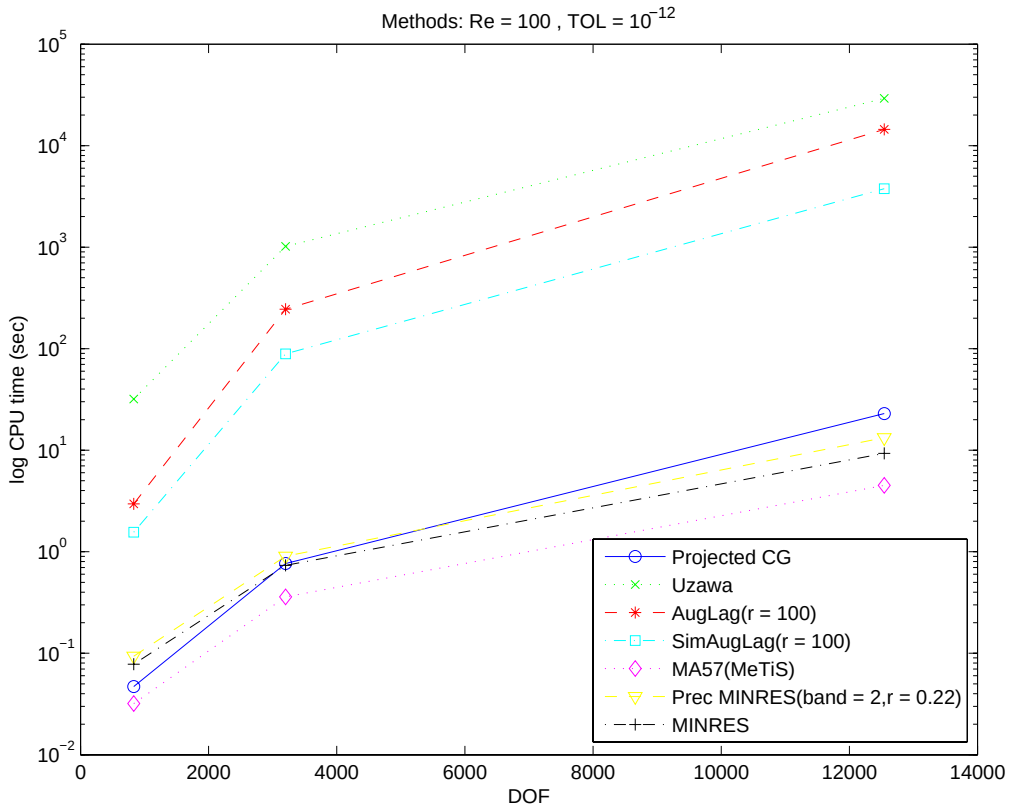

Σχήμα 4.127: Συγκεντρωτικό: DOF vs CPU time, για (Re = 10,TOL = $10^{-12}$).



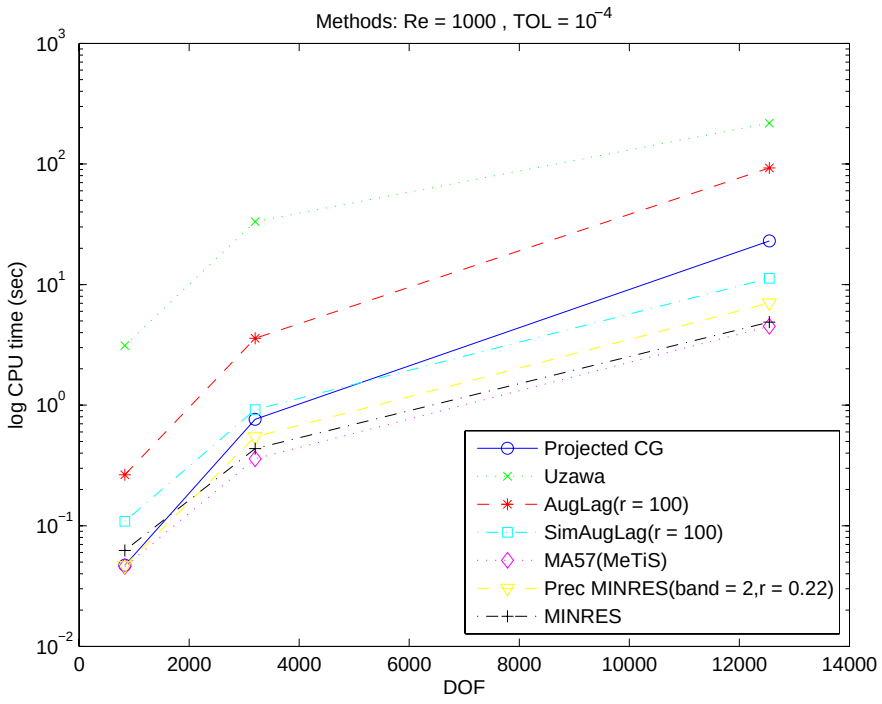

Σχήμα 4.128: Συγκεντρωτικό: DOF vs CPU time, για (Re = 100,TOL = $10^{-4}$).



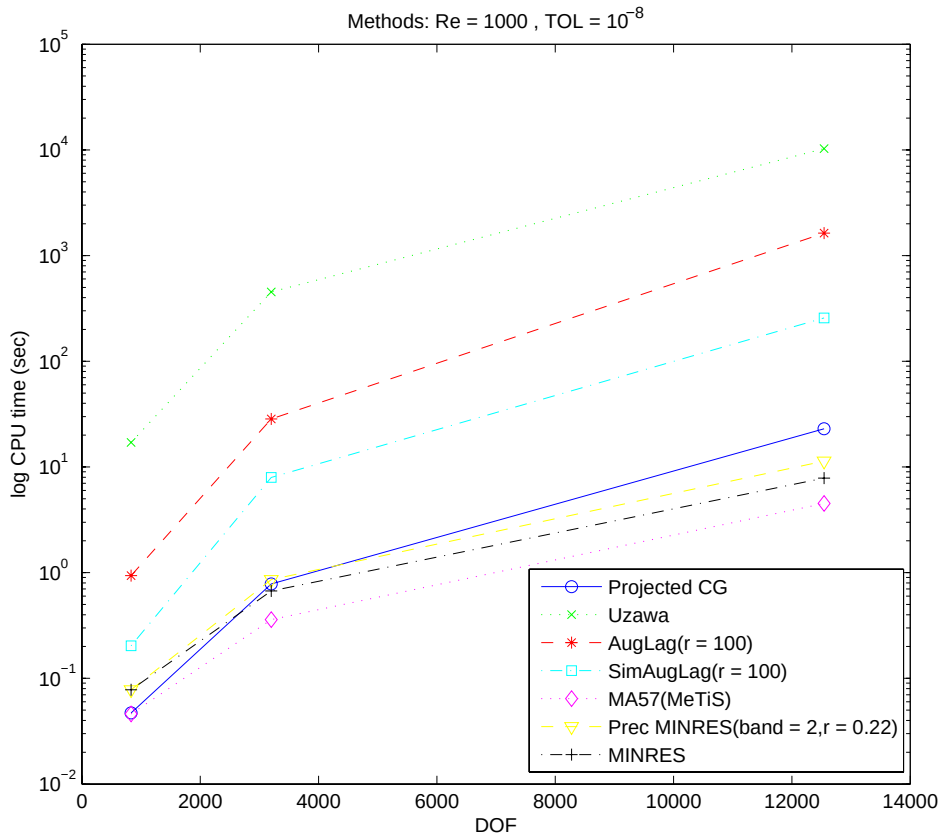

Σχήμα 4.129: Συγκεντρωτικό: DOF vs CPU time, για (Re = 100,TOL = $10^{-8}$).



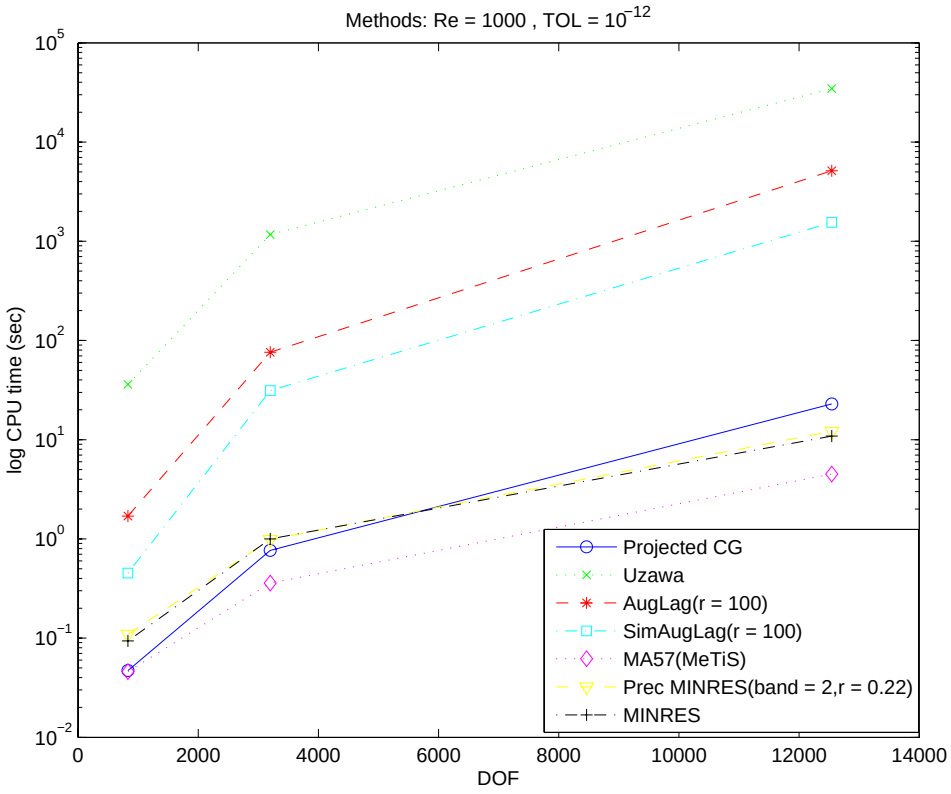

Σχήμα 4.130: Συγκεντρωτικό: DOF vs CPU time, για (Re = 100,TOL = $10^{-12}$).




Σχήμα 4.131: Συγκεντρωτικό: DOF vs CPU time, για (Re = 1000,TOL = $10^{-4}$).




Σχήμα 4.132: Συγκεντρωτικό: DOF vs CPU time, για (Re = 1000,TOL = $10^{-8}$).




Σχήμα 4.133: Συγκεντρωτικό: DOF vs CPU time, για (Re = 1000,TOL = $10^{-12}$).

# Κεφάλαιο 5

# Συμπεράσματα

## 5.1 Γενικές Παρατηρήσεις – Συμπεράσματα

Σε αυτό το κεφάλαιο συνοψίζουμε τα αποτελέσματα που πήραμε και δίνουμε κάποιες προτάσεις για περαιτέρω πειραματισμό, έρευνα και μελέτη. Βασικά μπορούμε να σκιαγραφήσουμε την έρευνα που έγινε ως εξής: Αρχικά αναφερθήκαμε στις εξισώσεις Stokes, ένα θεμελιώδες μοντέλο ιξώδους ροής για τη ρευστοδυναμική. Είδαμε ότι κατά την αριθμητική προσέγγιση των λύσεων τους, απαιτείται η επίλυση πολύ μεγάλων συμμετρικών γραμμικών συστημάτων. Η επίλυση αυτών των συστημάτων καταλαμβάνει τον περισσότερο υπολογιστικό χρόνο, σε ολόκληρη την παραπάνω διαδικασία. Ανάλογα συμπεράσματα, μπορεί να διαπιστώσει κάποιος ότι ισχύουν και για μεγάλα γραμμικά συστήματα, αντίστοιχης μορφής, τα οποία προκύπτουν σε προβλήματα βελτιστοποίησης (βλέπε [4]).

Διαπιστώσαμε λοιπόν, ότι υπάρχει πολύ μεγάλη ανάγκη για γρήγορους και αποδοτικούς αλγορίθμους για την επίλυση μεγάλων και αραιών (συμμετρικών) γραμμικών συστημάτων. Για το λόγο αυτό υλοποιήσαμε, δοκιμάσαμε και συγκρίναμε μεταξύ τους μια σειρά από τις πιο μοντέρνες μεθόδους, για επίλυση μεγάλων και αραιών γραμμικών συστημάτων, που είναι διαθέσιμες αυτή τη στιγμή.

Οι μέθοδοι που μελετήθηκαν μπορούν να κατηγοριοποιηθούν σε τρεις ομάδες:

1. **Μέθοδοι βελτιστοποίησης** (Uzawa, Augmented Lagrangian, Simplified Augmented Lagrangian)
2. **Μέθοδοι υποχώρων Krylov** (Projected CG, Preconditioned MINRES, MINRES)
3. **Άμεσες μέθοδοι** (MA57)

Από τα εκτενή αριθμητικά πειράματα που εκτελέστηκαν μπορούμε να εξάγουμε τα ακόλουθα συμπεράσματα, σχετικά με την αποδοτικότητα των παραπάνω αλγορίθμων:

- Αναφορικά με τους αλγορίθμους που ανήκουν στην κατηγορία των μεθόδων βελτιστοποίησης, παρατηρήσαμε ότι για δεδομένο αριθμό Reynolds και δεδομένη ανοχή TOL, ουσιαστικά σε όλες τις κατηγορίες μεγέθους του γραμμικού συστήματος, η Simplified Augmented Lagrangian απαιτεί το λιγότερο υπολογιστικό χρόνο, στη συνέχεια έρχεται η Augmented Lagrangian η οποία απαιτεί σημαντικά περισσότερο υπολογιστικό χρόνο και τελευταία η Uzawa, ο υπολογιστικός χρόνος της οποίας είναι πολύ μεγάλος σε όλες τις περιπτώσεις και ειδικότερα σε κάποιες απαγορευτικά μεγάλος. Δεδομένου του γεγονότος ότι οι μέθοδοι

Uzawa και Augmented Lagrangian, απαιτούν δύο επιλύσεις γραμμικών συστημάτων με πίνακα $A$ και $A_r$, αντίστοιχα, ανά επανάληψη, ενώ η Simplified Augmented Lagrangian απαιτεί μόνο μια επίλυση γραμμικού συστήματος με πίνακα $A_r$, ανά επανάληψη, συμπεραίνουμε ότι **στην κατηγορία των μεθόδων βελτιστοποίησης, αποδοτικότερη μέθοδος αποδεικνύεται η Simplified Augmented Lagrangian.**

Στο σημείο αυτό θα πρέπει να σημειώσουμε ότι από το Wilkinson Interlock Theorem (βλέπε [28]) προκύπτει ότι καθώς η παράμετρος $r$ των μεθόδων Augmented Lagrangian και Simplified Augmented Lagrangian αυξάνει, οι ιδιοτιμές του πίνακα $A_r = A + rBB^T$ διαχωρίζονται σε δυο ομάδες: Η πρώτη ομάδα φράσεται από πάνω από $\lambda_N(A)$ και η άλλη φράσεται από κάτω από $r\lambda_{N-q+1}(BB^T)$. Ο διαχωρισμός των ιδιοτιμών του πίνακα $A_r$, σ΄ αυτές τις δύο ομάδες, όσο η παράμετρος $r$ αυξάνει, προκαλεί σημαντικές δυσκολίες στις επαναληπτικές μεθόδους.

Σύμφωνα με τα παραπάνω, για τις μεθόδους Augmented Lagrangian και Simplified Augmented Lagrangian θα πρέπει να έχουμε υπόψη μας ότι, αν χρησιμοποιήσουμε κάποια επαναληπτική μέθοδο για την επίλυση του εσωτερικού γραμμικού συστήματος με πίνακα $A_r$, ενδέχεται, (ειδικά για μεγάλες τιμές του $r$) να χρειαστεί μεγάλος αριθμός επαναλήψεων για σύγκλιση ή ακόμα και να μην επιτευχθεί σύγκλιση. Αυτός είναι ένας βασικός λόγος για τον οποίο οι μέθοδοι Augmented Lagrangian και Simplified Augmented Lagrangian ενδείκνυται να συνδυάζονται με άμεσες μεθόδους επίλυσης του γραμμικού συστήματος με πίνακα $A_r$, σε κάθε επανάληψη.

- Αναφορικά με τους αλγορίθμους που ανήκουν στην κατηγορία των μεθόδων υποχώρων Krylov,τα αριθμητικά πειράματα έδειξαν ότι για δεδομένο αριθμό Reynolds και δεδομένη ανοχή TOL, ουσιαστικά σε όλες τις κατηγορίες μεγέθους του γραμμικού συστήματος, οι μέθοδοι MINRES και Preconditioned MINRES απαιτούν περίπου τον ίδιο υπολογιστικό χρόνο σε κάθε περίπτωση, ενώ η Projected CG απαιτεί περίπου το διπλάσιο υπολογιστικό χρόνο, σε σχέση με τις MINRES και Preconditioned MINRES.

  Επίσης ο πιθανότερος λόγος για τον οποίο η Preconditioned MINRES δε φάνηκε να είναι αποδοτικότερη σε σχέση με την κλασσική MINRES, είναι ο χρόνος που καταναλώνεται σε κάθε βήμα για την επίλυση του γραμμικού συστήματος με τον προρρυθμιστή: $Mx = y$.

- Αναφορικά με την άμεση μέθοδο που υλοποιείται στις ρουτίνες MA57 παρατηρήσαμε ότι, σε κάθε περίπτωση, απαιτεί το λιγότερο υπολογιστικό χρόνο, σε σχέση με όλες τις άλλες (επαναληπτικές) μεθόδους που υλοποιήσαμε και δοκιμάσαμε. Το μοναδικό μεγάλο μειονέκτημα είναι οι απαιτήσεις σε μνήμη για την απαλοιφή. Παρόλα αυτά, όπως διαπιστώσαμε, οι παραπάνω απαιτήσεις σε μνήμη μειώνονται σημαντικά, αν γίνει χρήση του λογισμικού MeTiS, το οποίο υπολογίζει ένα reordering που ελαχιστοποιεί κατά πολύ το γέμισμα κατά την απαλοιφή.

Συγκεντρωτικά θα λέγαμε ότι αν έχουμε άφθονη διαθέσιμη μνήμη και ζητάμε απόλυτη αξιοπιστία στη λύση του γραμμικού συστήματος, στο μικρότερο δυνατό υπολογιστικό χρόνο, η άμεση μέθοδος που υλοποιείται στις ρουτίνες MA57 αποτελεί την καλύτερη λύση.

Αν το γραμμικό μας σύστημα είναι πολύ μεγάλο (π.χ. προέρχεται από διακριτοποίηση των εξισώσεων Stokes σε 3 χωρικές διαστάσεις), θέλουμε κάποια αξιοπιστία στη λύση, αλλά δεν έχουμε διαθέσιμα τα τεράστια ποσά μνήμης που θα απαιτούν οι ρουτίνες της MA57, μια καλή επιλογή είναι οι μέθοδοι MINRES - Preconditioned MINRES και Projected CG. Επίσης θα μπορούσαμε

να χρησιμοποιήσουμε και τη μέθοδο Simplified Augmented Lagrangian με κατάλληλη επιλογή της παραμέτρου $r$.

Αν θέλουμε εντελώς σχηματικά να κατατάξουμε τις μεθόδους που δοκιμάστηκαν θα προτείναμε την ακόλουθη κατάταξη:

1. MA57
2. MINRES - Preconditioned MINRES
3. Projected CG
4. Simplified Augmented Lagrangian
5. Augmented Lagrangian
6. Uzawa

## 5.2 Προτάσεις για περαιτέρω έρευνα και μελέτη.

Στη συνέχεια παραθέτουμε κάποια ειδικότερα θέματα που πρέπει να έχουμε υπόψη μας, τα οποία αποτελούν και προτάσεις για περαιτέρω έρευνα και μελέτη.

Τα αριθμητικά πειράματα έδειξαν ότι η μέθοδος Projected CG φαίνεται να «ρίχνει» το υπόλοιπο εξαιρετικά γρήγορα. Φυσικά σε κάθε επανάληψη της Projected CG εκτός από το κόστος του κλασσικού αλγορίθμου της CG, έχουμε και το κόστος του υπολογισμού μιας προβολής. Για τον υπολογισμό μιας προβολής, με τον τρόπο που έχουμε υλοποιήσει τον αλγόριθμο, απαιτείται η επίλυση δύο τριγωνικών γραμμικών συστημάτων.

Για τον παραπάνω λόγο, εικάζουμε ότι είναι πιθανόν σε ακόμα μεγαλύτερους πίνακες, με χειρότερο φασματικό δείκτη κατάστασης, σε σχέση με τους πίνακες που δοκιμάσαμε, η μέθοδος Projected CG να είναι αποδοτικότερη σε σχέση με τις μεθόδους MINRES και Preconditioned MINRES, αφού φαίνεται να έχει την ικανότητα να «ρίχνει» το υπόλοιπο σε μικρό αριθμό επαναλήψεων. Αυτό αποτελεί ένα θέμα που χρειάζεται περαιτέρω μελέτη.

Επίσης από τα αριθμητικά πειράματα, έγινε φανερό ότι για την κατασκευή ενός προρρυθμιστή πρέπει να έχει κανείς κατά νου ότι, για να είναι αυτός αποδοτικός θα πρέπει κανείς να σταθμίσει τη φασματική βελτίωση που δίνει ο προρρυθμιστής σε σχέση με το υπολογιστικό κόστος για την επίλυση ενός γραμμικού συστήματος με πίνακα τον προρρυθμιστή: $Mx = y$, σε κάθε επανάληψη. Αν το κόστος της επίλυσης ενός γραμμικού συστήματος με τον προρρυθμιστή είναι μεγάλο, είναι πολύ πιθανόν αυτό να επιβραδύνει το συνολικό υπολογισμό, καθιστώντας τον προρρυθμιστή αναποτελεσματικό (μη αποδοτικό).

Το υψηλό κόστος της επίλυσης του γραμμικού συστήματος με τον προρυθμιστή, σε κάθε επανάληψη, είναι πιθανότατα ο βασικός λόγος, στον οποίο οφείλεται η μη αποδοτικότητα (από πλευράς υπολογιστικού χρόνου) του προρυθμιστή που χρησιμοποιήθηκε στη MINRES, έναντι της κλασσικής MINRES.

Για τον παραπάνω λόγο, ένα ακόμα σημείο, το οποίο χρειάζεται περαιτέρω μελέτη, είναι ο πειραματισμός με τις παραμέτρους `r` και `band` του προρρυθμιστή της MINRES. Η δική μας πρόχειρη μελέτη έδειξε ότι τιμές $r < 1$ δίνουν αποδοτικούς προρυθμιστές, ενώ πολύ μεγάλες τιμές του $r$, δίνουν, γενικά, μη αποδοτικούς προρυθμιστές.

# Παράρτημα Α΄

# Οι κώδικες που αναπτύχθηκαν

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για όλες τις μεθόδους που δοκιμάστηκαν.

## Α΄.1 Κώδικες για τη μέθοδο Projected CG

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο Projected CG.

```
/* Author      : Manos Psycharis
 * Purpose     : Master Thesis
 * Date        : 03/03/2010
 *
 * Compile     gcc -Wall -ansi -pedantic -o file pro_cg.c memory.c
 *             -llapack -lblas
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>        /* because we use clock()    */
#include <sys/times.h>   /* because we use times()    */
#include <unistd.h>      /* because we use sysconf()  */
#include <string.h>      /* becuase we use strcmp()   */
#include "memory.h"

/*-------------------------25 Prototypes--------------------------------*/
void blank_line(void);
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL);
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm);
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p);
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,const int q,
                 double *g,double *u);
```

```
void CG(double *Bchol,int *I_A,int *J_A,double *A,int *I_B,int *J_B,
        double *B,const int nzA,const int nzB,const int N,const int q,
        double *u0,double *p0,double *d0,double *f,const double stop_tol,
        const int max_iter,double *rel_res,int *iter);
void Init_CG(double *Bchol,int *I_A,int *J_A,double *A,int *I_B,int *J_B,
             double *B,const int nzA,const int nzB,const int N,
             const int q,double *u0,double *p0,double *d0,double *f,
             double *g);
void projection(double *Bchol,int *I_B,int *J_B,double *B,const int nzB,
                const int N,const int q,double *v,double *z,double *Pv);
void mod_daxpy(const int dim,const double scalar,double *x,double *y);
double m_dnrm2(const int dim,double *x);
double m_ddot(const int dim,double *x,double *y);
void m_dcopy(const int dim,double *x,double *y);
void m_dscal(const int dim,const double scalar,double *vector);
void m_daxpy(const int dim,const double scalar,double *x,double *y);
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose);
void triangular_solve(double *AP,double *rhs,const int q,char transpose);
void cholesky(double *AP,const int q);
void cholesky_solve(double *AP,const int q,double *rhs);
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g);
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals);
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals);
void mat_mult(int *I_B,int *J_B,double *B,const int nzB,double *Bmult);
double m_dasum(const int dim,double *x);
int m_idamax(const int dim,double *x);
double my_maxnorm(const int dim,double *x);
/*-------------------3 Lapack Prototypes-------------------------*/
void dpptrf_(char *uplo,int *N,double AP[],int *info);
void dpptrs_(char *uplo,int *N,int *nrhs,double AP[],double b[],int *ldb,
             int *info);
void dtptrs_(char *uplo,char *trans,char *diag,int *N,int *nrhs,
             double AP[],double b[],int *ldb,int *info);
/*-------------------7 BLAS Prototypes-------------------------*/
void daxpy_(int *N,double *da,double dx[],int *incx,double dy[],
            int *incy);
void dscal_(int *N,double *da,double dx[],int *incx);
void dcopy_(int *N,double dx[],int *incx,double dy[],int *incy);
double ddot_(int *N,double dx[],int *incx,double dy[],int *incy);
double dnrm2_(int *N,double x[],int *incx);
double dasum_(int *N,double dx[],int *incx);
int idamax_(int *N,double dx[],int *incx);

/*************************************************************
```

```
 * Let the game begin...
 ************************************************************/
int main()
{
        int i,k,num_mat,tol_vals;
        char filename_A[60];
        char filename_B[60];
        char filenameK[60];
        char input_file[] = "input_data.dat";
        double TOL;
        double t1_clock,t2_clock;
        double t1_times,t2_times,cpu_time;
        struct tms tb1,tb2;          /* times() needs them  */
        double ticspersec;
        FILE *fp;
        fp = FOPEN(input_file,"r");
        num_mat = 9;    /* Set the number of different matrices */
        tol_vals = 5;  /* Set the number of different values of TOL */
        /* casting because sysconf() returns int */
        ticspersec = (double) sysconf(_SC_CLK_TCK);
        printf("\t\t\t\t\t\t\t\t\tMethod: Projected CG\n");
        blank_line();
        printf("TOL\tRe\tDOF\t|| f-Au-Bp||\t|| g-B^Tu||\t\t||r||\t\t
#iter\t||x-x*||_1\t\t||x-x*||_2\t\t||x-x*||_sup\t\t||r||
\t\tCPU time(sec)\n");
        blank_line();
        for(i=1;i<=num_mat;i++){
                fscanf(fp,"%s",filename_A);
                fscanf(fp,"%s",filename_B);
                fscanf(fp,"%s",filenameK);
                for(k=1;k<=tol_vals;k++){
                        fscanf(fp,"%lf",&TOL);
                        /* casting because times() returns (long) int */
                        t1_times = (double) times(&tb1);
                        t1_clock = (double) clock();
                        method(filenameK,filename_A,filename_B,TOL);
                        t2_clock = (double) clock();
                        t2_times = (double) times(&tb2);
                        cpu_time = (double)
                        ((tb2.tms_utime + tb2.tms_stime) -
                        (tb1.tms_utime + tb1.tms_stime));
                        printf("\t%f\n",cpu_time / ticspersec);
                }
                blank_line();
        }
        FCLOSE(fp);
        printf("\nProjected CG runs ended successfully!\n");
```

```
        return 0;
}


/* horizontal line */
void blank_line(void)
{
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------\n");
}


/* parameterization of the Projected CG method  */
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL)
{
        int q,N,nzB,nzA,Re;
        int nBpack,max_iter,iter,global_nz;
        int *I_K,*J_K;
        double *K;
        int *I_A,*J_A,*I_B,*J_B;
        double *A,*B,*Bmult;
        double *u0,*p0,*f,*g,*d0;
        double rel_res,r_a,r_b;
        double norm1,norm2,maxnorm;
        read_whole_mat(filenameK,&nzA,&nzB,&N,&q,&I_K,&J_K,&K);
        readmat(filename_A,&N,&nzA,&I_A,&J_A,&A);
        readmat(filename_B,&q,&nzB,&I_B,&J_B,&B);
        nBpack = (q * (q + 1))/2;
        Bmult = (double *)CALLOC(nBpack,sizeof(double));
        u0 = (double *)MALLOC(N * sizeof(double));
        f = (double *)MALLOC(N * sizeof(double));
        g = (double *)MALLOC(q * sizeof(double));
        p0 = (double *)MALLOC(q * sizeof(double));
        d0 = (double *)MALLOC(N * sizeof(double));
        global_nz = nzA + 2 * nzB;
        construct_rhs(I_K,K,N,q,global_nz,f,g);
        FREE(I_K);
        FREE(J_K);
        FREE(K);
        /* Compute upper triangle of B**T *B and store it,in the vector
         * Bmult in packed form (columnwise)*/
        mat_mult(I_B,J_B,B,nzB,Bmult);
        /* Compute the upper triangular factor U from the Cholesky
         * factorization of B**T *B = U**T * U and store it,in the
```

```
         * vector Bmult in packed form (columnwise) */
        cholesky(Bmult,q);
        /* Initialization of u0,p0,d0    */
        Init_CG(Bmult,I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,q,u0,p0,d0,f,g);
        max_iter = 2 * N;
        /* CG method applied to PAP u = Pf  */
        CG(Bmult,I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,q,u0,p0,d0,f,TOL,max_iter,
           &rel_res,&iter);
/*      for(k=1;k<=N;k++)
                printf("u[%d] = %16.9e\n",k-1,u0[k-1]);
        for(k=1;k<=q;k++)
                printf("p[%d] = %16.9e\n",k-1,p0[k-1]);   */
        /* compute error norms */
        error_norms(u0,p0,N,q,&norm1,&norm2,&maxnorm);
        /* compute residual norms */
        r_a = mom_norm(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,f,u0,p0);
        r_b = comp_norm(I_B,J_B,B,nzB,q,g,u0);
        if((strcmp(filename_A,"MtrxA_Re10_N578_nzA3826.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re10_N2178_nzA16818.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re10_N8450_nzA70450.dat") == 0))
                Re = 10;
        else if((strcmp(filename_A,"MtrxA_Re100_N578_nzA3826.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re100_N2178_nzA16818.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re100_N8450_nzA70450.dat") == 0))
                Re = 100;
        else
                Re = 1000;
        printf("%.1e\t%d\t%d\t%16.9e %16.9e  %16.9e\t%d\t%16.9e\t%16.9e\t
%16.9e\t%16.9e",TOL,Re,N+q,r_a,r_b,sqrt(r_a * r_a + r_b * r_b),iter - 1,
        norm1,norm2,maxnorm,rel_res);
        FREE(I_A);
        FREE(J_A);
        FREE(A);
        FREE(I_B);
        FREE(J_B);
        FREE(B);
        FREE(Bmult);
        FREE(u0);
        FREE(f);
        FREE(g);
        FREE(p0);
        FREE(d0);
}


/* Computation of the error norm : ||x - x*||, where:
 * x  = (u,p)        the computed solution
```

```
 * x* = (1,1,...,1) the exact solution
 * vectors u and p remain unchanged on exit
 * norm1   : returns ||x - x*||_1
 * norm2   : returns ||x - x*||_2
 * maxnorm : returns ||x - x*||_sup    */
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm)
{
        int i,dim;
        double *err;
        dim = N + q;
        err = (double *)MALLOC(dim * sizeof(double));
        for(i=1;i<=N;i++)
                err[i-1] = u[i-1] - 1.0;
        for(i=N+1;i<=dim;i++)
                err[i-1] = p[i-N-1] - 1.0;
        (*norm1) = m_dasum(dim,err);
        (*norm2) = m_dnrm2(dim,err);
        (*maxnorm) = my_maxnorm(dim,err);
        FREE(err);
}



/* Computation of the norm: ||f - Au - Bp||_2, where u and p are the
 * computed velocity and pressure solutions, respectively.
 * This quantity is a measure of how good the solution satisfies the
 * discretized momentum equation: Au + Bp = f    */
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
                const int nzA,const int nzB,const int N,double *f,
                double *u,double *p)
{
        double scalar,norm;
        double *Au,*Bp;
        Au = (double *)MALLOC(N * sizeof(double));
        Bp = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_A,J_A,A,nzA,u,Au,N,'N'); /* Au = A * u     */
        mat_vec(I_B,J_B,B,nzB,p,Bp,N,'N'); /* Bp = B * p     */
        scalar = -1.0;
        mod_daxpy(N,scalar,f,Au); /* Au = f - Au      */
        scalar = -1.0;
        mod_daxpy(N,scalar,Au,Bp);/* Bp = Au - Bp    = (f - Au) - Bp */
        norm = m_dnrm2(N,Bp);/* norm = ||Bp|| = ||f - Au - Bp||*/
        FREE(Au);
        FREE(Bp);
        return norm;
}
```

```
/* Computation of the norm: ||g - B**T u||_2, where u is the computed
 * velocity solution.
 * This quantity is a measure of how good the solution satisfies the
 * discretized incompressibility condition: B**T u = g */
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,const int q,
                 double *g,double *u)
{
        double scalar,norm;
        double *Btu;
        Btu = (double *)MALLOC(q * sizeof(double));
        mat_vec(I_B,J_B,B,nzB,u,Btu,q,'T'); /* Btu = B**T * u  */
        scalar = -1.0;
        mod_daxpy(q,scalar,g,Btu);  /* Btu = g - Btu  */
        norm = m_dnrm2(q,Btu);/* norm = ||Btu|| = ||g - B**T * u||*/
        FREE(Btu);
        return norm;
}


/* Implementation of the Congugate Gradients (CG) method (algorithm)
 * applied to the symmetric positive definite linear system: PAPu = Pf
 * The pressure p is updated appropriately as suggested in the relevant
 * paper */
void CG(double *Bchol,int *I_A,int *J_A,double *A,int *I_B,int *J_B,
        double *B,const int nzA,const int nzB,const int N,const int q,
        double *u0,double *p0,double *d0,double *f,const double stop_tol,
        const int max_iter,double *rel_res,int *iter)
{
        double rold_rold,rnew_rnew,cg_alpha;
        double cg_beta,dp,scalar,norm_r_new,normb;
        double *r,*Ad;
        double *z,*Pv;
        r = (double *)MALLOC(N * sizeof(double));
        Ad = (double *)MALLOC(N * sizeof(double));
        z = (double *)MALLOC(q * sizeof(double));
        Pv = (double *)MALLOC(N * sizeof(double));
        m_dcopy(N,d0,r);  /*  r = d0  */
/*      printf("||r_0|| = %16.9e\n",m_dnrm2(N,r));    */
        /* Computes Pv = P * f, written in Pv */
        projection(Bchol,I_B,J_B,B,nzB,N,q,f,z,Pv);
        normb = m_dnrm2(N,Pv); /* Computes rhs norm : normb = ||Pf||_2 */
        (*rel_res) = 2.0;
        (*iter) = 1;
        while(((*rel_res) > (stop_tol * normb)) && ((*iter) <= max_iter)){
                rold_rold = m_ddot(N,r,r); /* rold_rold = (r0,r0) */
                /* Computes Ad = A * d0, written in Ad */
```

```
                mat_vec(I_A,J_A,A,nzA,d0,Ad,N,'N');
                /* Pv = P * Ad  (= P * Ad0   ),z = B+ * Ad = B+ * Ad0,
                 * written in z */
                projection(Bchol,I_B,J_B,B,nzB,N,q,Ad,z,Pv);
                dp = m_ddot(N,d0,Pv); /* Computes dp = (d0,wk) */
                cg_alpha = rold_rold/dp;
                scalar = cg_alpha;
                /* Computes u0 = cg_alpa * d0 + u0,update velocity u */
                m_daxpy(N,scalar,d0,u0);
                scalar = - cg_alpha;
                /* Computes p0 = - cg_alpha * z + p0,update pressure p */
                m_daxpy(q,scalar,z,p0);
                scalar = - cg_alpha;
                /* Computes  r = -cg_alpha * Pv + r*/
                m_daxpy(N,scalar,Pv,r);
                rnew_rnew = m_ddot(N,r,r);/* Computes rnew_rnew = (r,r)*/
                cg_beta = rnew_rnew/rold_rold;
                /* Computes d0 = r + cg_beta * d0 */
                mod_daxpy(N,cg_beta,r,d0);
                /* Computes norm of new residual*/
                norm_r_new = m_dnrm2(N,r);
        /*      printf("||r_%d|| = %16.9e\n",(*iter),norm_r_new); */
                (*rel_res) = norm_r_new;
                (*iter)++;
        }
        FREE(r);
        FREE(Ad);
        FREE(z);
        FREE(Pv);
        if((*iter) > max_iter){
                fprintf(stderr,"Maximum number of iterations
exceeded!\n");
                exit(2);
        }
}

/* Initialization of initial velocity u0,initial search direction d0
 * and initial pressure p0.
 * 1.Initialize u0 with the value: u0 = B * (B**T *B)^-1 * g
 *      a.Solve B**T *B z = g
 *      b.Set u0 = B * z
 * 2.Initialize d0 with the value: d0 = P *  (f - A * u0)
 * 3.Initialize p0 with the value: p0 = B+ * (f - A * u0)
 * where B+ is the Moore-Penrose pseudoinverse of the matrix B.
 * If the matrix B has full rank (this is the case) B+ has the
 * representation: B+ = (B**T *B)^-1 * B**T       */
void Init_CG(double *Bchol,int *I_A,int *J_A,double *A,int *I_B,
```

```
                int *J_B,double *B,const int nzA,const int nzB,
                const int N,const int q,double *u0,double *p0,double *d0,
                double *f,double *g)
{
        double scalar;
        double *Ad,*rhs;
        Ad = (double *)MALLOC(N * sizeof(double));
        rhs = (double *)MALLOC(q * sizeof(double));
        m_dcopy(q,g,rhs); /* rhs = g, g remains unchanged */
        /* Solve B^TB z = rhs, sol z written in rhs */
        cholesky_solve(Bchol,q,rhs);
        /*initialization of u0: u0 = B * rhs */
        mat_vec(I_B,J_B,B,nzB,rhs,u0,N,'N');
        FREE(rhs);
        /* Computes Ad =Au0, written in Ad */
        mat_vec(I_A,J_A,A,nzA,u0,Ad,N,'N');
        scalar = -1.0;
        m_daxpy(N,scalar,f,Ad); /* Ad = -f + Ad      */
        scalar = -1.0;
        m_dscal(N,scalar,Ad); /* Ad = - Ad (= f - Ad    ) = f - Au0 */
        /* initialization of d0 and p0: d0 = P * Ad (= P *(f - Au0))
         * and p0 = B+ * (f - Au0) */
        projection(Bchol,I_B,J_B,B,nzB,N,q,Ad,p0,d0);
        FREE(Ad);
}



/* Given v in IR^N, computes Pv in IR^N. If it is called with v = Ad_k,
 * on exit z = B^+v = B^+Ad_k  */
void projection(double *Bchol,int *I_B,int *J_B,double *B,const int nzB,
                const int N,const int q,double *v,double *z,double *Pv)
{
        double scalar;
        mat_vec(I_B,J_B,B,nzB,v,z,q,'T'); /* Compute z = B^T * v */
        /*Solve R^T w = z ,solution written in z*/
        triangular_solve(Bchol,z,q,'T');
        /* Solve R   q = w ,solution written in z */
        triangular_solve(Bchol,z,q,'N');
        /* Compute d = B * q, d written in Pv, z remains unchanged */
        mat_vec(I_B,J_B,B,nzB,z,Pv,N,'N');
        scalar = -1.0;
        m_daxpy(N,scalar,v,Pv); /*   Pv = -v + Pv   */
        m_dscal(N,scalar,Pv);   /*   Pv = -Pv       */
}
```

```
/* y = x + scalar * y ,x remains unchanged on exit */
void mod_daxpy(const int dim,const double scalar,double *x,double *y)
{
        double alpha;
        m_dscal(dim,scalar,y); /* y = scalar * y     */
        alpha = 1.0;
        m_daxpy(dim,alpha,x,y);/* Computes y = alpha * x + y,y = x + y */
}

/* Computes l_2 norm of x : ||x||_2 */
double m_dnrm2(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dnrm2_(&N,x,&incx);
        return result;
}

/* Computes the dot product (x,y)_2  */
double m_ddot(const int dim,double *x,double *y)
{
        int N,incx,incy;
        double result;
        incx = 1;
        incy = 1;
        N = dim;
        result = ddot_(&N,x,&incx,y,&incy);
        return result;
}



/* Copies vector x to y, y = x,x remains unchanged on exit */
void m_dcopy(const int dim,double *x,double *y)
{
        int incx,incy,N;
        N = dim;
        incx = 1;
        incy = 1;
        dcopy_(&N,x,&incx,y,&incy);
}



/* On exit vector = scalar * vector  */
```

```
void m_dscal(const int dim,const double scalar,double *vector)
{
        int N,incx;
        double da = scalar;
        incx = 1;
        N = dim;
        dscal_(&N,&da,vector,&incx);
}



/* y = scalar * x + y,x remains unchanged on exit */
void m_daxpy(const int dim,const double scalar,double *x,double *y)
{
        int incx,incy,N;
        double da = scalar;
        incx = 1;
        incy = 1;
        N = dim;
        daxpy_(&N,&da,x,&incx,y,&incy); /*  y = da * x + y  */
}


/* Computes matrix-vector product: y = mat x or y = mat^T x,mat is
 * represented in coordinate sparse format.
 * dimy is the dimension of the output vector y,x remains unchanged
 * on exit */
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose)
{
        int k;
        for(k=1;k<=dimy;k++)
                y[k-1] = 0.0;
        if(transpose == 'N'){     /*  Compute y = mat * x        */
                for(k=1;k<=nz;k++)
                        y[I_mat[k-1]-1] += mat[k-1] * x[J_mat[k-1]-1];
        }
        else if(transpose == 'T'){    /* Compute y = mat^T * x      */
                for(k=1;k<=nz;k++)
                        y[J_mat[k-1]-1] += mat[k-1] * x[I_mat[k-1]-1];
        }
        else {
                fprintf(stderr,"Wrong option for matrix-vector
multiplication!\n");
                exit(2);
        }
}
```

```
/* Solution of a triangular linear system. System matrix is stored in
 * packed format.
 * On entry  : rhs holds the right hand side vector.
 * On exit   : rhs holds the solution vector. */
void triangular_solve(double *AP, double *rhs, const int q, char transpose)
{
        int dim, nrhs, ldb, info;
        char uplo = 'U';
        char trans = transpose;
        char diag = 'N';
        dim = q;
        nrhs = 1;
        ldb = dim;
        dtptrs_(&uplo, &trans, &diag, &dim, &nrhs, AP, rhs, &ldb, &info);
        if(info < 0){
                fprintf(stderr, "The %d argument in dtptrs_() had an
illegal value!\n", -info);
                exit(2);
        }
        else if(info > 0){
                fprintf(stderr, "The %d diagonal element of matrix is
zero!\n", info);
                exit(2);
        }
}


/* Choleksy factorization of a symmetric positive definite matrix A.
 * On entry : AP holds the upper triangle of the matrix stored in
 * packed format (columnwise)
 * On exit  : AP holds the triangular factor U from the Cholesky
 * factorization of A = U**T * U, stored in packed format (columnwise)
*/
void cholesky(double *AP, const int q)
{
        int dim, info;
        char uplo = 'U';
        dim = q;
        dpptrf_(&uplo, &dim, AP, &info);
        if(info < 0){
                fprintf(stderr, "The %d argument in dpptrf_() had an
illegal value!\n", -info);
                exit(2);
        }
        else if(info > 0){
                fprintf(stderr, "The leading minor of order %d of
```

```
matrix is not positive definite!\n",info);
                exit(2);
        }
}

/* Solves a linear system Ax=b with a symmetric positive definite
 * matrix A, stored in packed format, using the Cholesky factorization
 * A = U**T *U, as computed previously by dpptrf_()
 * On entry : AP holds the triangular factor U from the Cholesky
 * factorization of A = U**T *U, stored in packed format, as computed by
 * dpptrf_().AP remains unchanged on exit.
 * On entry : rhs holds the right hand side vector, on exit the solution
 * vector */
void cholesky_solve(double *AP,const int q,double *rhs)
{
        int dim,nrhs,ldb,info;
        char uplo = 'U';
        dim = q;
        nrhs = 1;
        ldb = q;
        dpptrs_(&uplo,&dim,&nrhs,AP,rhs,&ldb,&info);
        if(info < 0){
                fprintf(stderr,"The %d argument in dpptrs_() had an
illegal value!\n",-info);
                exit(2);
        }
}


/* The right hand side vector of the linear system is constructed such
 * that its solution is the vector (1,1,...,1) */
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g)
{
        int i,k,dim;
        double xsum;
        dim = N + q;
        for(i=1;i<=N;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                f[i-1] = xsum;
        }
        for(i=N+1;i<=dim;i++){
                xsum = 0.0;
```

```
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                g[i-N-1] = xsum;
        }
}



/* reading of the nonzero elements of the whole matrix, stored in
 * coordinate format, in global enumeration */
void read_whole_mat(char *filename, int *nzA, int *nzB, int *N, int *q,
                    int **I, int **J, double **vals)
{
        int i,j,k,p;
        int non_zero_A, non_zero_B, dim_A, rank_B;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&non_zero_A);
        fscanf(fp,"%d",&non_zero_B);
        fscanf(fp,"%d",&dim_A);
        fscanf(fp,"%d",&rank_B);
        (*nzA) = non_zero_A;
        (*nzB) = non_zero_B;
        (*N) = dim_A;
        (*q) = rank_B;
        p = non_zero_A + 2 * non_zero_B;
        tmp1 = (int *)MALLOC(p * sizeof(int));
        tmp2 = (int *)MALLOC(p * sizeof(int));
        tmp3 = (double *)MALLOC(p * sizeof(double));
        for(k=1;k<=p;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}
```

```
/* reading of the nonzero elements of matrix A or B, stored in coordinate
 * format, in local enumeration */
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals)
{
        int i,j,k;
        int order,nzero;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&order);
        fscanf(fp,"%d",&nzero);
        (*q) = order;
        (*nz) = nzero;
        tmp1 = (int *)MALLOC((*nz) * sizeof(int));
        tmp2 = (int *)MALLOC((*nz) * sizeof(int));
        tmp3 = (double *)MALLOC((*nz) * sizeof(double));
        for(k=1;k<=nzero;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* Sparse multiplication B**T *B and columnwise storage of the upper
 * triangle of B**T *B */
void mat_mult(int *I_B,int *J_B,double *B,const int nzB,double *Bmult)
{
        int k,el;
        int ki,kj,li,lj,index;
        for(k=1;k<=nzB;k++){
                ki = J_B[k-1];
                kj = I_B[k-1];
                for(el=1;el<=nzB;el++){
                        li = I_B[el-1];
                        lj = J_B[el-1];
                        if(kj == li){
                                if(ki <= lj){
```

```
                                        index= ki + ( lj - 1 ) * lj/2 - 1;
                                        Bmult[index] += B[k-1] * B[el-1];
                                }
                        }
                }
        }
}


/* Computes l_1 norm of x : ||x||_1 */
double m_dasum(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dasum_(&N,x,&incx);
        return result;
}

/*  returns the index of element of vector x,having max. absolute value.
 *  -1 shifting for C compatibility */
int m_idamax(const int dim,double *x)
{
        int N,incx;
        int pos;
        N = dim;
        incx = 1;
        pos = idamax_(&N,x,&incx);
        pos--;
        return pos;
}

/* Computes maximum norm of x : ||x||_sup */
double my_maxnorm(const int dim,double *x)
{
        int pos;
        double result;
        pos = m_idamax(dim,x);
        result = fabs(x[pos]);
        return result;
}
```

## Α΄.2 Κώδικες για τη μέθοδο Uzawa

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο Uzawa.

```
/* Author      : Manos Psycharis
 * Purpose     : Master Thesis
 * Date        : 03/03/2010
 *
 * Compile     gcc -Wall -ansi -pedantic -o file uzawa.c memory.c -lblas
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>        /* because we use clock()    */
#include <sys/times.h>   /* because we use times()    */
#include <unistd.h>      /* because we use sysconf()  */
#include <string.h>      /* becuase we use strcmp()   */
#include "memory.h"

/*-------------------------20 Prototypes-------------------------------*/
void blank_line(void);
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL);
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm);
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p);
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,
                 const int q,double *g,double *u);
void uzawa(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
           const int nzA,const int nzB,const int N,const int q,
           double *f,double *g,const double TOL,double *u,double *p,
           int *n_iter);
void CG(int *I_A,int *J_A,double *A,const int nzA,double *x,double *rhs,
        const double stop_TOL,const int max_iter,const int N,
        double *rel_res,int *iter);
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose);
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g);
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals);
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals);
void mod_daxpy(const int dim,const double scalar,double *x,double *y);
double m_dnrm2(const int dim,double *x);
double m_ddot(const int dim,double *x,double *y);
void m_dcopy(const int dim,double *x,double *y);
void m_dscal(const int dim,const double scalar,double *vector);
void m_daxpy(const int dim,const double scalar,double *x,double *y);
```

```
double m_dasum(const int dim,double *x);
int m_idamax(const int dim,double *x);
double my_maxnorm(const int dim,double *x);

/*--------------------7 BLAS Prototypes--------------------------------*/
void daxpy_(int *N,double *da,double dx[],int *incx,double dy[],
            int *incy);
void dscal_(int *N,double *da,double dx[],int *incx);
void dcopy_(int *N,double dx[],int *incx,double dy[],int *incy);
double ddot_(int *N,double dx[],int *incx,double dy[],int *incy);
double dnrm2_(int *N,double x[],int *incx);
double dasum_(int *N,double dx[],int *incx);
int idamax_(int *N,double dx[],int *incx);

/*****************************************************************
 * Let the game begin...
 *****************************************************************/
int main()
{
        int i,k,num_mat,tol_vals;
        char filename_A[60];
        char filename_B[60];
        char filenameK[60];
        char input_file[] = "input_data.dat";
        double TOL;
        double t1_clock,t2_clock;
        double t1_times,t2_times,cpu_time;
        struct tms tb1,tb2;          /* times() needs them  */
        double ticspersec;
        time_t mytime;
        FILE *fp;
        fp = FOPEN(input_file,"r");
        num_mat = 9;  /* Set the number of different matrices */
        tol_vals = 5; /* Set the number of different values of TOL */
        /* casting because sysconf() returns int */
        ticspersec = (double) sysconf(_SC_CLK_TCK);
        printf("\t\t\t\t\t\t\t\t\tMethod: Uzawa\n");
        blank_line();
        printf("TOL\tRe\tDOF\t   ||f-Au-Bp||\t    ||g-B^Tu||\t\t||r||
\t\t#iter\t   ||x-x*||_1\t\t   ||x-x*||_2\t\t   ||x-x*||_sup
\t\tCPU_time(sec)\n");
        blank_line();
        for(i=1;i<=num_mat;i++){
                fscanf(fp,"%s",filename_A);
                fscanf(fp,"%s",filename_B);
                fscanf(fp,"%s",filenameK);
                for(k=1;k<=tol_vals;k++){
```

```
                        mytime = time(NULL);
                        fprintf(stderr,ctime(&mytime));
                fprintf(stderr,"\nnum_mat = %d\tTOL_val = %d\n\n",i,k);
                        fscanf(fp,"%lf",&TOL);
                        /* casting because times() returns (long) int */
                        t1_times = (double) times(&tb1);
                        t1_clock = (double) clock();
                        method(filenameK,filename_A,filename_B,TOL);
                        t2_clock = (double) clock();
                        t2_times = (double) times(&tb2);
                cpu_time = (double) ((tb2.tms_utime + tb2.tms_stime) -
                                     (tb1.tms_utime + tb1.tms_stime));
                        printf("\t%f\n",cpu_time / ticspersec);
                }
                blank_line();
        }
        FCLOSE(fp);
        printf("\nUzawa runs ended successfully!\n");
        return 0;
}


/* horizontal line */
void blank_line(void)
{
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------\n");
}



/* parameterization of the uzawa method  */
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL)
{
        int q,N,nzB,nzA,Re;
        int global_nz,n_iter,i;
        double r_a,r_b;
        double norm1,norm2,maxnorm;
        int *I_K,*J_K;
        double *K;
        int *I_A,*J_A,*I_B,*J_B;
        double *A,*B;
        double *f,*g;
        double *u,*p;
```

```
        read_whole_mat(filenameK,&nzA,&nzB,&N,&q,&I_K,&J_K,&K);
        readmat(filename_A,&N,&nzA,&I_A,&J_A,&A);
        readmat(filename_B,&q,&nzB,&I_B,&J_B,&B);
        u = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(q * sizeof(double));
        f = (double *)MALLOC(N * sizeof(double));
        g = (double *)MALLOC(q * sizeof(double));
        global_nz = nzA + 2 * nzB;
        construct_rhs(I_K,K,N,q,global_nz,f,g);
        FREE(I_K);
        FREE(J_K);
        FREE(K);
        /* Initially u and p hold u_0 and p_0 respectively */
        for(i=1;i<=N;i++)
                u[i-1] = 0.0; /* initialize u */
        for(i=1;i<=q;i++)
                p[i-1] = 0.0; /* initialize p */
        /* Output the computed solution u and p */
        uzawa(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,q,f,g,TOL,u,p,&n_iter);
/*      for(i=1;i<=N;i++)
                printf("u[%d] = %16.9e\n",i-1,u[i-1]);
        for(i=1;i<=q;i++)
                printf("p[%d] = %16.9e\n",i-1,p[i-1]);    */
        /* compute error norms    */
        error_norms(u,p,N,q,&norm1,&norm2,&maxnorm);
        /* compute residual norms */
        r_a = mom_norm(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,f,u,p);
        r_b = comp_norm(I_B,J_B,B,nzB,q,g,u);
        if((strcmp(filename_A,"MtrxA_Re10_N578_nzA3826.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N2178_nzA16818.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N8450_nzA70450.dat") == 0))
                Re = 10;
        else if((strcmp(filename_A,"MtrxA_Re100_N578_nzA3826.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re100_N2178_nzA16818.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re100_N8450_nzA70450.dat") == 0))
                Re = 100;
        else
                Re = 1000;
        printf("%.1e\t%d\t%d\t%16.9e %16.9e  %16.9e\t%d\t%16.9e\t
%16.9e\t%16.9e",TOL,Re,N+q,r_a,r_b,sqrt(r_a * r_a + r_b * r_b),
n_iter - 1,norm1,norm2,maxnorm);
        FREE(I_A);
        FREE(J_A);
        FREE(A);
        FREE(I_B);
        FREE(J_B);
        FREE(B);
```

```
        FREE(f);
        FREE(g);
        FREE(u);
        FREE(p);
}


/* Computation of the error norm : ||x - x*||, where:
 * x  = (u,p)        the computed solution
 * x* = (1,1,...,1) the exact solution
 * vectors u and p remain unchanged on exit
 * norm1   : returns ||x - x*||_1
 * norm2   : returns ||x - x*||_2
 * maxnorm : returns ||x - x*||_sup     */
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm)
{
        int i,dim;
        double *err;
        dim = N + q;
        err = (double *)MALLOC(dim * sizeof(double));
        for(i=1;i<=N;i++)
                err[i-1] = u[i-1] - 1.0;
        for(i=N+1;i<=dim;i++)
                err[i-1] = p[i-N-1] - 1.0;
        (*norm1) = m_dasum(dim,err);
        (*norm2) = m_dnrm2(dim,err);
        (*maxnorm) = my_maxnorm(dim,err);
        FREE(err);
}

/* Computation of the norm: ||f - Au - Bp||_2, where u and p are the
 * computed velocity and pressure solutions, respectively.
 * This quantity is a measure of how good the solution satisfies the
 * discretized momentum equation: Au + Bp = f     */
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p)
{
        double scalar,norm;
        double *Au,*Bp;
        Au = (double *)MALLOC(N * sizeof(double));
        Bp = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_A,J_A,A,nzA,u,Au,N,'N'); /* Au = A * u     */
        mat_vec(I_B,J_B,B,nzB,p,Bp,N,'N'); /* Bp = B * p     */
        scalar = -1.0;
        mod_daxpy(N,scalar,f,Au); /* Au = f - Au      */
```

```
        scalar = -1.0;
        mod_daxpy(N,scalar,Au,Bp); /* Bp = Au - Bp =(f - Au) - Bp */
        norm = m_dnrm2(N,Bp); /* norm = ||Bp||=||f - Au - Bp|| */
        FREE(Au);
        FREE(Bp);
        return norm;
}


/* Computation of the norm: ||g - B**T u||_2, where u is the computed
 * velocity solution.
 * This quantity is a measure of how good the solution satisfies the
 * discretized incompressibility condition: B**T u = g */
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,const int q,
                 double *g,double *u)
{
        double scalar,norm;
        double *Btu;
        Btu = (double *)MALLOC(q * sizeof(double));
        mat_vec(I_B,J_B,B,nzB,u,Btu,q,'T'); /* Btu = B**T * u  */
        scalar = -1.0;
        mod_daxpy(q,scalar,g,Btu);  /* Btu = g - Btu  */
        norm = m_dnrm2(q,Btu); /* norm =||Btu||=||g - B**T * u|| */
        FREE(Btu);
        return norm;
}



/* On entry : u and p hold u_0 and p_0 respectively, allocation and
 * initialization in main()
 * On exit  : u and p hold the computed velocity and pressure,
 * respectively. */
void uzawa(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
           const int nzA,const int nzB,const int N,const int q,
           double *f,double *g,const double TOL,double *u,double *p,
           int *n_iter)
{
        int max_iter,iter;
        double scalar,norm_R_p,stop_TOL,rel_res,R_pdotR_p,R_pdotz,t;
        double *Bvec,*Btrans_vec,*R_p,*rhs,*w,*z;
        Bvec = (double *)MALLOC(N * sizeof(double));
        Btrans_vec = (double *)MALLOC(q * sizeof(double));
        R_p = (double *)MALLOC(q * sizeof(double));
        rhs = (double *)MALLOC(N * sizeof(double));
        z = (double *)MALLOC(q * sizeof(double));
        /*zero initial guess for Aw = rhs, rhs = BR_p, for the first time*/
```

```
w = (double *)CALLOC(N, sizeof(double));
m_dcopy(q,g,R_p); /*  R_p = g    */
/* Btrans_vec = B^T u_0  */
mat_vec(I_B,J_B,B,nzB,u,Btrans_vec,q,'T');
/* R_p = Btrans_vec - R_p, R_p = B^T u_0 - g  */
scalar = -1.0;
mod_daxpy(q,scalar,Btrans_vec,R_p);
norm_R_p = m_dnrm2(q,R_p); /*  norm_R_p = ||R_p||_2  */
(*n_iter) = 1;
while(norm_R_p > TOL){
/*        printf("Uzawa iteration: %d\n\n",(*n_iter));  */
          mat_vec(I_B,J_B,B,nzB,p,Bvec,N,'N'); /* Bvec = B p */
          m_dcopy(N,Bvec,rhs); /* rhs = Bvec  */
          scalar = -1.0;
          mod_daxpy(N,scalar,f,rhs);/* rhs =f - rhs,rhs = f - B p */
          /* Initial guess for CG,u from previous iteration,
           * first time u_0 */
          stop_TOL = TOL;
          max_iter = 2 * N;
          /* Solve Au = rhs, rhs = f - Bp, sol written to u */
         CG(I_A,J_A,A,nzA,u,rhs,stop_TOL,max_iter,N,&rel_res,&iter);
/*        printf("CG1 Iterations = %d\n",iter-1);    */
          m_dcopy(q,g,R_p); /*  R_p = g    */
          /* Btrans_vec = B^T u  */
          mat_vec(I_B,J_B,B,nzB,u,Btrans_vec,q,'T');
          /* R_p = Btrans_vec - R_p, R_p = B^T u - g  */
          scalar = -1.0;
          mod_daxpy(q,scalar,Btrans_vec,R_p);
          mat_vec(I_B,J_B,B,nzB,R_p,rhs,N,'N'); /* rhs = B R_p */
          /* Initial guess for CG,w from previous iteration,
           * first time w = 0  */
          stop_TOL = TOL;
          max_iter = 2 * N;
          /* Solve Aw = rhs, rhs = B R_p */
         CG(I_A,J_A,A,nzA,w,rhs,stop_TOL,max_iter,N,&rel_res,&iter);
/*        printf("CG2 Iterations = %d\n\n",iter-1);  */
          R_pdotR_p = m_ddot(q,R_p,R_p); /* R_pdotR_p = (R_p,R_p) */
          mat_vec(I_B,J_B,B,nzB,w,z,q,'T'); /* z = B^T w     */
          R_pdotz = m_ddot(q,R_p,z); /* R_pdotz = (R_p,z)    */
          t = R_pdotR_p/R_pdotz;
          scalar = t;
          m_daxpy(q,scalar,R_p,p); /*  p = t R_p + p    */
          norm_R_p = m_dnrm2(q,R_p); /* norm_R_p = ||R_p||_2 */
          (*n_iter)++;
}
mat_vec(I_B,J_B,B,nzB,p,Bvec,N,'N'); /* Bvec = B p */
m_dcopy(N,Bvec,rhs); /* rhs = Bvec  */
```

```
        scalar = -1.0;
        mod_daxpy(N,scalar,f,rhs); /* rhs = f - rhs, rhs = f - B p */
        /* Initial guess for CG,u from the last loop  */
        stop_TOL = TOL;
        max_iter = 2 * N;
        /* Solve Au = rhs, rhs = f - Bp */
        CG(I_A,J_A,A,nzA,u,rhs,stop_TOL,max_iter,N,&rel_res,&iter);
/*      printf("Last_CG Iterations = %d\n",iter-1);  */
        FREE(Bvec);
        FREE(Btrans_vec);
        FREE(R_p);
        FREE(rhs);
        FREE(w);
        FREE(z);
}

/* On entry x contains the initial guess,on exit the solution     */
void CG(int *I_A,int *J_A,double *A,const int nzA,double *x,double *rhs,
        const double stop_TOL,const int max_iter,const int N,
        double *rel_res,int *iter)
{
        double norm_rhs,scalar,Ap_p,rold_rold;
        double cg_alpha,rnew_rnew,cg_beta,norm_r_new;
        double *r,*p,*Ap;
        r = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(N * sizeof(double));
        Ap = (double *)MALLOC(N * sizeof(double));
        norm_rhs = m_dnrm2(N,rhs); /* norm_rhs = ||rhs||_2  */
        mat_vec(I_A,J_A,A,nzA,x,r,N,'N'); /* r = A x       */
        scalar = -1.0;
        mod_daxpy(N,scalar,rhs,r); /* r =rhs + scalar r,r = rhs - r */
        m_dcopy(N,r,p);    /* p = r  */
        (*rel_res) = 2.0;
        (*iter) = 1;
      while(((*rel_res) > (stop_TOL * norm_rhs)) && ((*iter) <= max_iter)){
                mat_vec(I_A,J_A,A,nzA,p,Ap,N,'N'); /*  Ap = A p */
                Ap_p = m_ddot(N,Ap,p);             /*  Ap_p = (Ap,p) */
                rold_rold = m_ddot(N,r,r);         /* rold_rold = (r,r) */
                cg_alpha = rold_rold/Ap_p;
                scalar = cg_alpha;
                /*  x = cg_alpha p + x,update solution vector */
                m_daxpy(N,scalar,p,x);
                scalar = -cg_alpha;
                /*  r = - cg_alpha Ap + r,update residual vector  */
                m_daxpy(N,scalar,Ap,r);
                rnew_rnew = m_ddot(N,r,r);/* rnew_rnew = (r,r) */
                cg_beta = rnew_rnew/rold_rold;
```

```
                scalar = cg_beta;
                /* p = r(new) + cg_beta p, update search direction */
                mod_daxpy(N, scalar, r, p);
                norm_r_new = m_dnrm2(N, r); /* norm_r_new = ||r(new)|| */
                (*rel_res) = norm_r_new;
                (*iter)++;
        }
        FREE(r);
        FREE(p);
        FREE(Ap);
        if((*iter) > max_iter){
                fprintf(stderr,"Maximum number of iterations exceeded!\n");
                exit(2);
        }
}


/* Computes matrix-vector product: y = mat x or y = mat^T x, mat is
 * represented in coordinate sparse format.
 * dimy is the dimension of the output vector y,
 * x remains unchanged on exit */
void mat_vec(int *I_mat, int *J_mat, double *mat, const int nz, double *x,
             double *y, const int dimy, char transpose)
{
        int k;
        for(k=1;k<=dimy;k++)
                y[k-1] = 0.0;
        if(transpose == 'N'){     /*  Compute y = mat * x        */
                for(k=1;k<=nz;k++)
                        y[I_mat[k-1]-1] += mat[k-1] * x[J_mat[k-1]-1];
        }
        else if(transpose == 'T'){    /* Compute y = mat^T * x      */
                for(k=1;k<=nz;k++)
                        y[J_mat[k-1]-1] += mat[k-1] * x[I_mat[k-1]-1];
        }
        else {
        fprintf(stderr,"Wrong option for matrix-vector multiplication!\n");
        exit(2);
        }
}


/* The right hand side vector of the linear system is constructed
 * such that its solution is the vector (1,1,...,1) */
void construct_rhs(int *I_K, double *K, const int N, const int q,
                   const int global_nz, double *f, double *g)
{
```

```
        int i,k,dim;
        double xsum;
        dim = N + q;
        for(i=1;i<=N;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                f[i-1] = xsum;
        }
        for(i=N+1;i<=dim;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                g[i-N-1] = xsum;
        }
}


/* reading of the nonzero elements of the whole matrix,
 * stored in coordinate format, in global enumeration */
void read_whole_mat(char *filename, int *nzA, int *nzB, int *N, int *q,
                    int **I, int **J, double **vals)
{
        int i,j,k,p;
        int non_zero_A, non_zero_B, dim_A, rank_B;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&non_zero_A);
        fscanf(fp,"%d",&non_zero_B);
        fscanf(fp,"%d",&dim_A);
        fscanf(fp,"%d",&rank_B);
        (*nzA) = non_zero_A;
        (*nzB) = non_zero_B;
        (*N) = dim_A;
        (*q) = rank_B;
        p = non_zero_A + 2 * non_zero_B;
        tmp1 = (int *)MALLOC(p * sizeof(int));
        tmp2 = (int *)MALLOC(p * sizeof(int));
        tmp3 = (double *)MALLOC(p * sizeof(double));
        for(k=1;k<=p;k++){
```

```
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* reading of the nonzero elements of matrix A or B,
 * stored in coordinate format, in local enumeration */
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals)
{
        int i,j,k;
        int order,nzero;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&order);
        fscanf(fp,"%d",&nzero);
        (*q) = order;
        (*nz) = nzero;
        tmp1 = (int *)MALLOC((*nz) * sizeof(int));
        tmp2 = (int *)MALLOC((*nz) * sizeof(int));
        tmp3 = (double *)MALLOC((*nz) * sizeof(double));
        for(k=1;k<=nzero;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* y = x + scalar * y ,x remains unchanged on exit */
void mod_daxpy(const int dim,const double scalar,double *x,double *y)
```

```
{
        double alpha;
        m_dscal(dim,scalar,y); /* y = scalar * y     */
        alpha = 1.0;
        m_daxpy(dim,alpha,x,y);/* Computes y = alpha * x + y, y = x + y */
}

/* Computes l_2 norm of x : ||x||_2 */
double m_dnrm2(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dnrm2_(&N,x,&incx);
        return result;
}

/* Computes the dot product (x,y)_2  */
double m_ddot(const int dim,double *x,double *y)
{
        int N,incx,incy;
        double result;
        incx = 1;
        incy = 1;
        N = dim;
        result = ddot_(&N,x,&incx,y,&incy);
        return result;
}

/* Copies vector x to y, y = x,x remains unchanged on exit */
void m_dcopy(const int dim,double *x,double *y)
{
        int incx,incy,N;
        N = dim;
        incx = 1;
        incy = 1;
        dcopy_(&N,x,&incx,y,&incy);
}

/* On exit vector = scalar * vector */
void m_dscal(const int dim,const double scalar,double *vector)
{
        int N,incx;
        double da = scalar;
        incx = 1;
        N = dim;
```

```
        dscal_(&N,&da,vector,&incx);
}

/* y = scalar * x + y, x remains unchanged on exit */
void m_daxpy(const int dim, const double scalar, double *x, double *y)
{
        int incx, incy, N;
        double da = scalar;
        incx = 1;
        incy = 1;
        N = dim;
        daxpy_(&N,&da,x,&incx,y,&incy); /*  y = da * x + y  */
}

/* Computes l_1 norm of x : ||x||_1 */
double m_dasum(const int dim, double *x)
{
        int N, incx;
        double result;
        N = dim;
        incx = 1;
        result = dasum_(&N,x,&incx);
        return result;
}

/*  returns the index of element of vector x, having max. absolute value.
 *  -1 shifting for C compatibility */
int m_idamax(const int dim, double *x)
{
        int N, incx;
        int pos;
        N = dim;
        incx = 1;
        pos = idamax_(&N,x,&incx);
        pos--;
        return pos;
}

/* Computes maximum norm of x : ||x||_sup */
double my_maxnorm(const int dim, double *x)
{
        int pos;
        double result;
        pos = m_idamax(dim,x);
        result = fabs(x[pos]);
        return result;
}
```

## Α΄.3 Κώδικες για τη μέθοδο Augmented Lagrangian

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο Augmented Lagrangian.

```
/* Author      : Manos Psycharis
 * Purpose     : Master Thesis
 * Date        : 03/03/2010
 *
 * Compile
 *       gcc -Wall -ansi -pedantic -o file aug_lag.c memory.c -lblas -lg2c
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>        /* because we use clock()    */
#include <sys/times.h>   /* because we use times()    */
#include <unistd.h>      /* because we use sysconf()  */
#include <string.h>      /* becuase we use strcmp()   */
#include "memory.h"

/*-------------------------22 Prototypes-------------------------------*/
void blank_line(void);
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL,const double rho);
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm);
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p);
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,
                 const int q,double *g,double *u);
void construct_fr(int *I_B,int *J_B,double *B,const int nzB,
                  const int N,const double rho,double *f,
                  double *g,double *f_r);
void aug_lag(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
             const int nzA,const int nzB,const int N,const int q,
             double *f_r,double *g,const double rho,const double TOL,
             double *u,double *p,int *n_iter);
void CG(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
        const int nzA,const int nzB,double *x,double *rhs,
        const double rho,const double stop_TOL,const int max_iter,
        const int N,const int q,double *rel_res,int *iter);
void CG_mat_vec(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const double rho,
                const int N,const int q,double *x,double *Ar_x);
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
```

```
                double *y,const int dimy,char transpose);
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g);
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals);
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals);
void mod_daxpy(const int dim,const double scalar,double *x,double *y);
double m_dnrm2(const int dim,double *x);
double m_ddot(const int dim,double *x,double *y);
void m_dcopy(const int dim,double *x,double *y);
void m_dscal(const int dim,const double scalar,double *vector);
void m_daxpy(const int dim,const double scalar,double *x,double *y);
double m_dasum(const int dim,double *x);
int m_idamax(const int dim,double *x);
double my_maxnorm(const int dim,double *x);
/*--------------------7 BLAS Prototypes-------------------------------*/
void daxpy_(int *N,double *da,double dx[],int *incx,double dy[],
            int *incy);
void dscal_(int *N,double *da,double dx[],int *incx);
void dcopy_(int *N,double dx[],int *incx,double dy[],int *incy);
double ddot_(int *N,double dx[],int *incx,double dy[],int *incy);
double dnrm2_(int *N,double x[],int *incx);
double dasum_(int *N,double dx[],int *incx);
int idamax_(int *N,double dx[],int *incx);

/*****************************************************************
 * Let the game begin...
 *****************************************************************/
int main()
{
        int i,k,num_mat,tol_vals;
        char filename_A[60];
        char filename_B[60];
        char filenameK[60];
        char input_file[] = "input_data.dat";
        double TOL,rho;
        double t1_clock,t2_clock;
        double t1_times,t2_times,cpu_time;
        struct tms tb1,tb2;          /* times() needs them  */
        double ticspersec;
        time_t mytime;
        FILE *fp;
        fp = FOPEN(input_file,"r");
        num_mat = 9; /* Set the number of different matrices */
        tol_vals = 5; /* Set the number of different values of TOL */
        /* Set the parameter rho (positive real number) of the
         * Augmented Lagrangian Method */
```

```
        rho = 100.0;
        /* casting because sysconf() returns int */
        ticspersec = (double) sysconf(_SC_CLK_TCK);
        printf("\t\t\t\t\t\t\t\t\t\tMethod:
Augmented_Lagrangian_(r_=_%f_)\n",rho);
        blank_line();
        printf("TOL\tRe\tDOF\t___||f-Au-Bp||\t_____||g-B^Tu||\t\t||r||
\t\t#iter\t___||x-x*||_1\t\t___||x-x*||_2\t\t___||x-x*||_sup\t\t
CPU_time(sec)\n");
        blank_line();
        for(i=1;i<=num_mat;i++){
                fscanf(fp,"%s",filename_A);
                fscanf(fp,"%s",filename_B);
                fscanf(fp,"%s",filenameK);
                for(k=1;k<=tol_vals;k++){
                        mytime = time(NULL);
                        fprintf(stderr,ctime(&mytime));
                fprintf(stderr,"\nnum_mat_=_%d\tTOL_val_=_%d\n\n",i,k);
                        fscanf(fp,"%lf",&TOL);
                        /* casting because times() returns (long) int */
                        t1_times = (double) times(&tb1);
                        t1_clock = (double) clock();
                        method(filenameK,filename_A,filename_B,TOL,rho);
                        t2_clock = (double) clock();
                        t2_times = (double) times(&tb2);
                cpu_time = (double) ((tb2.tms_utime + tb2.tms_stime) -
                                        (tb1.tms_utime + tb1.tms_stime));
                        printf("\t%f\n",cpu_time / ticspersec);
                }
                blank_line();
        }
        FCLOSE(fp);
        printf("\nAugmented_Lagrangian_runs_ended_successfully!\n");
        return 0;
}

/* horizontal line */
void blank_line(void)
{
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------\n");
}


/* parameterization of the Augmented Lagrangian Method  */
```

```
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL,const double rho)
{
        int q,N,nzB,nzA,Re;
        int global_nz,n_iter,i;
        double r_a,r_b;
        double norm1,norm2,maxnorm;
        int *I_K,*J_K;
        double *K;
        int *I_A,*J_A,*I_B,*J_B;
        double *A,*B;
        double *f,*g,*f_r;
        double *u,*p;
        read_whole_mat(filenameK,&nzA,&nzB,&N,&q,&I_K,&J_K,&K);
        readmat(filename_A,&N,&nzA,&I_A,&J_A,&A);
        readmat(filename_B,&q,&nzB,&I_B,&J_B,&B);
        u = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(q * sizeof(double));
        f = (double *)MALLOC(N * sizeof(double));
        f_r = (double *)MALLOC(N * sizeof(double));
        g = (double *)MALLOC(q * sizeof(double));
        global_nz = nzA + 2 * nzB;
        construct_rhs(I_K,K,N,q,global_nz,f,g);
        FREE(I_K);
        FREE(J_K);
        FREE(K);
        /* construction of f_r = f + r B g */
        construct_fr(I_B,J_B,B,nzB,N,rho,f,g,f_r);
        /* Initially u and p hold u_0 and p_0 respectively */
        for(i=1;i<=N;i++)
                u[i-1] = 0.0; /* initialize u */
        for(i=1;i<=q;i++)
                p[i-1] = 0.0; /* initialize p */
        /* Output the computed solution u and p */
        aug_lag(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,q,f_r,g,rho,TOL,u,p,
                &n_iter);
/*      for(i=1;i<=N;i++)
                printf("u[%d] = %16.9e\n",i-1,u[i-1]);
        for(i=1;i<=q;i++)
                printf("p[%d] = %16.9e\n",i-1,p[i-1]);    */
        /* compute error norms */
        error_norms(u,p,N,q,&norm1,&norm2,&maxnorm);
        /* compute residual norms */
        r_a = mom_norm(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,f,u,p);
        r_b = comp_norm(I_B,J_B,B,nzB,q,g,u);
        if((strcmp(filename_A,"MtrxA_Re10_N578_nzA3826.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N2178_nzA16818.dat") == 0) ||
```

```
            (strcmp(filename_A,"MtrxA_Re10_N8450_nzA70450.dat") == 0))
                Re = 10;
        else if((strcmp(filename_A,"MtrxA_Re100_N578_nzA3826.dat") == 0)
            || (strcmp(filename_A,"MtrxA_Re100_N2178_nzA16818.dat") == 0)
            || (strcmp(filename_A,"MtrxA_Re100_N8450_nzA70450.dat") == 0))
                Re = 100;
        else
                Re = 1000;
        printf("%.1e\t%d\t%d\t%16.9e %16.9e  %16.9e\t%d\t%16.9e\t
%16.9e\t%16.9e",TOL,Re,N+q,r_a,r_b,sqrt(r_a * r_a + r_b * r_b),
n_iter - 1,norm1,norm2,maxnorm);
        FREE(I_A);
        FREE(J_A);
        FREE(A);
        FREE(I_B);
        FREE(J_B);
        FREE(B);
        FREE(f);
        FREE(f_r);
        FREE(g);
        FREE(u);
        FREE(p);
}

/* Computation of the error norm : ||x - x*||, where:
 * x  = (u,p)        the computed solution
 * x* = (1,1,...,1) the exact solution
 * vectors u and p remain unchanged on exit
 * norm1   : returns ||x - x*||_1
 * norm2   : returns ||x - x*||_2
 * maxnorm : returns ||x - x*||_sup    */
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm)
{
        int i,dim;
        double *err;
        dim = N + q;
        err = (double *)MALLOC(dim * sizeof(double));
        for(i=1;i<=N;i++)
                err[i-1] = u[i-1] - 1.0;
        for(i=N+1;i<=dim;i++)
                err[i-1] = p[i-N-1] - 1.0;
        (*norm1) = m_dasum(dim,err);
        (*norm2) = m_dnrm2(dim,err);
        (*maxnorm) = my_maxnorm(dim,err);
        FREE(err);
}
```

```
/* Computation of the norm: ||f - Au - Bp||_2, where u and p are the
 * computed velocity and pressure solutions, respectively.
 * This quantity is a measure of how good the solution satisfies the
 * discretized momentum equation: Au + Bp = f     */
double mom_norm(int *I_A, int *J_A, double *A, int *I_B, int *J_B,
                double *B, const int nzA, const int nzB, const int N,
                double *f, double *u, double *p)
{
        double scalar, norm;
        double *Au, *Bp;
        Au = (double *)MALLOC(N * sizeof(double));
        Bp = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_A, J_A, A, nzA, u, Au, N, 'N'); /* Au = A * u     */
        mat_vec(I_B, J_B, B, nzB, p, Bp, N, 'N'); /* Bp = B * p     */
        scalar = -1.0;
        mod_daxpy(N, scalar, f, Au); /* Au = f - Au      */
        scalar = -1.0;
        mod_daxpy(N, scalar, Au, Bp); /* Bp = Au - Bp = (f - Au) - Bp */
        norm = m_dnrm2(N, Bp); /* norm = ||Bp|| = ||f - Au - Bp|| */
        FREE(Au);
        FREE(Bp);
        return norm;
}


/* Computation of the norm: ||g - B**T u||_2, where u is the computed
 * velocity solution.
 * This quantity is a measure of how good the solution satisfies the
 * discretized incompressibility condition: B**T u = g */
double comp_norm(int *I_B, int *J_B, double *B, const int nzB,
                 const int q, double *g, double *u)
{
        double scalar, norm;
        double *Btu;
        Btu = (double *)MALLOC(q * sizeof(double));
        mat_vec(I_B, J_B, B, nzB, u, Btu, q, 'T'); /* Btu = B**T * u  */
        scalar = -1.0;
        mod_daxpy(q, scalar, g, Btu);  /* Btu = g - Btu  */
        norm = m_dnrm2(q, Btu); /* norm = ||Btu|| = ||g - B**T * u|| */
        FREE(Btu);
        return norm;
}
```

```
/* Construction of the vector f_r = f + r B * g.
 * Allocation of f_r in main() */
void construct_fr(int *I_B,int *J_B,double *B,const int nzB,
                  const int N,const double rho,double *f,double *g,
                  double *f_r)
{
        double scalar;
        mat_vec(I_B,J_B,B,nzB,g,f_r,N,'N'); /* f_r = B * g  */
        scalar = rho;
        mod_daxpy(N,scalar,f,f_r); /* f_r = f + r f_r = f + r B * g */
}


/* On entry : u and p hold u_0 and p_0 respectively,
 * allocation and initialization in main()
 * On exit  : u and p hold the computed velocity and pressure,
 * respectively. */
void aug_lag(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
             const int nzA,const int nzB,const int N,const int q,
             double *f_r,double *g,const double rho,const double TOL,
             double *u,double *p,int *n_iter)
{
        int max_iter,iter;
        double scalar,norm_R_p,stop_TOL,rel_res,R_pdotR_p,R_pdotz,t;
        double *Bvec,*Btrans_vec,*R_p,*rhs,*w,*z;
        Bvec = (double *)MALLOC(N * sizeof(double));
        Btrans_vec = (double *)MALLOC(q * sizeof(double));
        R_p = (double *)MALLOC(q * sizeof(double));
        rhs = (double *)MALLOC(N * sizeof(double));
        z = (double *)MALLOC(q * sizeof(double));
        /*zero initial guess for Aw = rhs,rhs = BR_p,for the first time */
        w = (double *)CALLOC(N,sizeof(double));
        m_dcopy(q,g,R_p); /*  R_p = g   */
        /* Btrans_vec = B^T u_0 */
        mat_vec(I_B,J_B,B,nzB,u,Btrans_vec,q,'T');
        /* R_p = Btrans_vec - R_p, R_p = B^T u_0 - g  */
        scalar = -1.0;
        mod_daxpy(q,scalar,Btrans_vec,R_p);
        norm_R_p = m_dnrm2(q,R_p); /*  norm_R_p = ||R_p||_2  */
        (*n_iter) = 1;
        while(norm_R_p > TOL){
        /*      printf("AL iteration: %d\n\n",(*n_iter));   */
                mat_vec(I_B,J_B,B,nzB,p,Bvec,N,'N'); /* Bvec = B p */
                m_dcopy(N,Bvec,rhs); /* rhs = Bvec  */
                /* rhs = f_r - rhs, rhs = f_r - B p */
                scalar = -1.0;
                mod_daxpy(N,scalar,f_r,rhs);
```

```
                /* Initial guess for CG,u from previous iteration,
                 * first time u_0 */
                stop_TOL = TOL;
                max_iter = 2 * N;
                /* Solve A_r u = rhs, rhs = f_r - B p */
                CG(I_A,J_A,A,I_B,J_B,B,nzA,nzB,u,rhs,rho,stop_TOL,
                    max_iter,N,q,&rel_res,&iter);
        /*      printf("CG1 Iterations = %d\n",iter-1);     */
                m_dcopy(q,g,R_p); /*  R_p = g    */
                /* Btrans_vec = B^T u  */
                mat_vec(I_B,J_B,B,nzB,u,Btrans_vec,q,'T');
                /* R_p = Btrans_vec - R_p, R_p = B^T u - g  */
                scalar = -1.0;
                mod_daxpy(q,scalar,Btrans_vec,R_p);
                mat_vec(I_B,J_B,B,nzB,R_p,rhs,N,'N'); /* rhs = B R_p */
                /* Initial guess for CG,w from previous iteration,
                 * first time w = 0  */
                stop_TOL = TOL;
                max_iter = 2 * N;
                /* Solve A_r w = rhs, rhs = B R_p */
                CG(I_A,J_A,A,I_B,J_B,B,nzA,nzB,w,rhs,rho,stop_TOL,
                    max_iter,N,q,&rel_res,&iter);
        /*      printf("CG2 Iterations = %d\n\n",iter-1);    */
                R_pdotR_p = m_ddot(q,R_p,R_p); /* R_pdotR_p = (R_p,R_p) */
                mat_vec(I_B,J_B,B,nzB,w,z,q,'T'); /* z = B^T w     */
                R_pdotz = m_ddot(q,R_p,z); /* R_pdotz = (R_p,z)    */
                t = R_pdotR_p/R_pdotz;
                scalar = t;
                m_daxpy(q,scalar,R_p,p); /*  p = t R_p + p    */
                norm_R_p = m_dnrm2(q,R_p); /* norm_R_p = ||R_p||_2 */
                (*n_iter)++;
        }
        mat_vec(I_B,J_B,B,nzB,p,Bvec,N,'N'); /* Bvec = B p */
        m_dcopy(N,Bvec,rhs); /* rhs = Bvec  */
        scalar = -1.0;
        mod_daxpy(N,scalar,f_r,rhs); /* rhs = f_r - rhs,rhs = f_r - B p */
        /* Initial guess for CG,u from the last loop  */
        stop_TOL = TOL;
        max_iter = 2 * N;
        /* Solve A_r u = rhs, rhs = f_r - B p */
        CG(I_A,J_A,A,I_B,J_B,B,nzA,nzB,u,rhs,rho,stop_TOL,
            max_iter,N,q,&rel_res,&iter);
/*      printf("Last_CG Iterations = %d\n",iter-1);    */
        FREE(Bvec);
        FREE(Btrans_vec);
        FREE(R_p);
        FREE(rhs);
```

```
        FREE(w);
        FREE(z);
}

/* On entry x contains the initial guess,on exit the solution     */
void CG(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
        const int nzA,const int nzB,double *x,double *rhs,
        const double rho,const double stop_TOL,const int max_iter,
        const int N,const int q,double *rel_res,int *iter)
{
        double norm_rhs,scalar,Ap_p,rold_rold;
        double cg_alpha,rnew_rnew,cg_beta,norm_r_new;
        double *r,*p,*Ap;
        r = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(N * sizeof(double));
        Ap = (double *)MALLOC(N * sizeof(double));
        norm_rhs = m_dnrm2(N,rhs); /* norm_rhs = ||rhs||_2   */
        CG_mat_vec(I_A,J_A,A,I_B,J_B,B,nzA,nzB,rho,N,q,x,r);/* r =A_r x */
        scalar = -1.0;
        mod_daxpy(N,scalar,rhs,r); /* r = rhs + scalar r,r = rhs - r */
        m_dcopy(N,r,p); /* p = r   */
        (*rel_res) = 2.0;
        (*iter) = 1;
      while(((*rel_res) > (stop_TOL * norm_rhs)) && ((*iter) <= max_iter)){
                /* Ap = A_r p   */
                CG_mat_vec(I_A,J_A,A,I_B,J_B,B,nzA,nzB,rho,N,q,p,Ap);
                Ap_p = m_ddot(N,Ap,p); /*   Ap_p = (Ap,p) */
                rold_rold = m_ddot(N,r,r); /* rold_rold = (r,r) */
                cg_alpha = rold_rold/Ap_p;
                /*   x = cg_alpha p + x,update solution vector */
                scalar = cg_alpha;
                m_daxpy(N,scalar,p,x);
                /*   r =- cg_alpha Ap + r,update residual vector */
                scalar =-cg_alpha;
                m_daxpy(N,scalar,Ap,r);
                rnew_rnew = m_ddot(N,r,r); /* rnew_rnew = (r,r) */
                cg_beta = rnew_rnew/rold_rold;
                /* p = r(new) + cg_beta p,update search direction */
                scalar = cg_beta;
                mod_daxpy(N,scalar,r,p);
                norm_r_new = m_dnrm2(N,r); /* norm_r_new = ||r(new)|| */
                (*rel_res) = norm_r_new;
                (*iter)++;
        }
        FREE(r);
        FREE(p);
        FREE(Ap);
```

```
        if((*iter) > max_iter){
                fprintf(stderr,"Maximum number of iterations exceeded!\n");
                exit(2);
        }
}

/* Given x computes Ar_x = A x + r B B^T x ,
 * x remains unchanged on exit */
void CG_mat_vec(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
                const int nzA,const int nzB,const double rho,const int N,
                const int q,double *x,double *Ar_x)
{
        double scalar;
        double *y,*w;
        y = (double *)MALLOC(q * sizeof(double));
        w = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_B,J_B,B,nzB,x,y,q,'T'); /* y = B^T x  */
        mat_vec(I_B,J_B,B,nzB,y,Ar_x,N,'N'); /* Ar_x = B y  */
        mat_vec(I_A,J_A,A,nzA,x,w,N,'N'); /* w = A x  */
        scalar = rho;
        mod_daxpy(N,scalar,w,Ar_x); /* Ar_x = w + r Ar_x  */
        FREE(y);
        FREE(w);
}


/* Computes matrix-vector product: y = mat x or y = mat^T x,
 * mat is represented in coordinate sparse format.
 * dimy is the dimension of the output vector y,
 * x remains unchanged on exit */
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose)
{
        int k;
        for(k=1;k<=dimy;k++)
                y[k-1] = 0.0;
        if(transpose == 'N'){     /*  Compute y = mat * x        */
                for(k=1;k<=nz;k++)
                        y[I_mat[k-1]-1] += mat[k-1] * x[J_mat[k-1]-1];
        }
        else if(transpose == 'T'){    /* Compute y = mat^T * x      */
                for(k=1;k<=nz;k++)
                        y[J_mat[k-1]-1] += mat[k-1] * x[I_mat[k-1]-1];
        }
        else {
        fprintf(stderr,"Wrong option for matrix-vector multiplication!\n");
        exit(2);
```

```
        }
}


/* The right hand side vector of the linear system is constructed
 * such that its solution is the vector (1,1,...,1) */
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g)
{
        int i,k,dim;
        double xsum;
        dim = N + q;
        for(i=1;i<=N;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                f[i-1] = xsum;
        }
        for(i=N+1;i<=dim;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                g[i-N-1] = xsum;
        }
}


/* reading of the nonzero elements of the whole matrix,
 * stored in coordinate format,in global enumeration */
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals)
{
        int i,j,k,p;
        int non_zero_A,non_zero_B,dim_A,rank_B;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&non_zero_A);
        fscanf(fp,"%d",&non_zero_B);
        fscanf(fp,"%d",&dim_A);
        fscanf(fp,"%d",&rank_B);
```

```
        (*nzA) = non_zero_A;
        (*nzB) = non_zero_B;
        (*N) = dim_A;
        (*q) = rank_B;
        p = non_zero_A + 2 * non_zero_B;
        tmp1 = (int *)MALLOC(p * sizeof(int));
        tmp2 = (int *)MALLOC(p * sizeof(int));
        tmp3 = (double *)MALLOC(p * sizeof(double));
        for(k=1;k<=p;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* reading of the nonzero elements of matrix A or B,
 * stored in coordinate format, in local enumeration */
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals)
{
        int i,j,k;
        int order,nzero;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&order);
        fscanf(fp,"%d",&nzero);
        (*q) = order;
        (*nz) = nzero;
        tmp1 = (int *)MALLOC((*nz) * sizeof(int));
        tmp2 = (int *)MALLOC((*nz) * sizeof(int));
        tmp3 = (double *)MALLOC((*nz) * sizeof(double));
        for(k=1;k<=nzero;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
```

```
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* y = x + scalar * y ,x remains unchanged on exit */
void mod_daxpy(const int dim,const double scalar,double *x,double *y)
{
        double alpha;
        m_dscal(dim,scalar,y); /* y = scalar * y     */
        alpha = 1.0;
        m_daxpy(dim,alpha,x,y);/* Computes y = alpha * x + y,y = x + y */
}

/* Computes l_2 norm of x : ||x||_2 */
double m_dnrm2(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dnrm2_(&N,x,&incx);
        return result;
}

/* Computes the dot product (x,y)_2  */
double m_ddot(const int dim,double *x,double *y)
{
        int N,incx,incy;
        double result;
        incx = 1;
        incy = 1;
        N = dim;
        result = ddot_(&N,x,&incx,y,&incy);
        return result;
}

/* Copies vector x to y, y = x,x remains unchanged on exit */
void m_dcopy(const int dim,double *x,double *y)
{
        int incx,incy,N;
        N = dim;
        incx = 1;
        incy = 1;
        dcopy_(&N,x,&incx,y,&incy);
```

```
}

/* On exit vector = scalar * vector */
void m_dscal(const int dim,const double scalar,double *vector)
{
        int N,incx;
        double da = scalar;
        incx = 1;
        N = dim;
        dscal_(&N,&da,vector,&incx);
}

/* y = scalar * x + y,x remains unchanged on exit */
void m_daxpy(const int dim,const double scalar,double *x,double *y)
{
        int incx,incy,N;
        double da = scalar;
        incx = 1;
        incy = 1;
        N = dim;
        daxpy_(&N,&da,x,&incx,y,&incy); /*  y = da * x + y  */
}

/* Computes l_1 norm of x : ||x||_1 */
double m_dasum(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dasum_(&N,x,&incx);
        return result;
}

/*  returns the index of element of vector x,having max. absolute value.
 *  -1 shifting for C compatibility */
int m_idamax(const int dim,double *x)
{
        int N,incx;
        int pos;
        N = dim;
        incx = 1;
        pos = idamax_(&N,x,&incx);
        pos--;
        return pos;
}
```

```
/* Computes maximum norm of x : ||x||_sup */
double my_maxnorm(const int dim,double *x)
{
        int pos;
        double result;
        pos = m_idamax(dim,x);
        result = fabs(x[pos]);
        return result;
}
```

## Α΄.4 Κώδικες για τη μέθοδο Simplified Augmented Lagrangian

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο Simplified Augmented Lagrangian.

```
/* Author      : Manos Psycharis
 * Purpose     : Master Thesis
 * Date        : 03/03/2010
 *
 * Compile
 *       gcc -Wall -ansi -pedantic -o file sim_aug_lag.c memory.c -lblas
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>       /* because we use clock()   */
#include <sys/times.h>  /* because we use times()   */
#include <unistd.h>     /* because we use sysconf() */
#include <string.h>     /* becuase we use strcmp()  */
#include "memory.h"

/*-------------------------22 Prototypes-------------------------------*/
void blank_line(void);
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL,const double rho);
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm);
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p);
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,const int q,
                 double *g,double *u);
void construct_fr(int *I_B,int *J_B,double *B,const int nzB,const int N,
                  const double rho,double *f,double *g,double *f_r);
```

```
void sim_aug_lag(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                 double *B,const int nzA,const int nzB,const int N,
                 const int q,double *f_r,double *g,const double rho,
                 const double TOL,double *u,double *p,int *n_iter);
void CG(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
        const int nzA,const int nzB,double *x,double *rhs,
        const double rho,const double stop_TOL,const int max_iter,
        const int N,const int q,double *rel_res,int *iter);
void CG_mat_vec(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const double rho,
                const int N,const int q,double *x,double *Ar_x);
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose);
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g);
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals);
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals);
void mod_daxpy(const int dim,const double scalar,double *x,double *y);
double m_dnrm2(const int dim,double *x);
double m_ddot(const int dim,double *x,double *y);
void m_dcopy(const int dim,double *x,double *y);
void m_dscal(const int dim,const double scalar,double *vector);
void m_daxpy(const int dim,const double scalar,double *x,double *y);
double m_dasum(const int dim,double *x);
int m_idamax(const int dim,double *x);
double my_maxnorm(const int dim,double *x);
/*-------------------7 BLAS Prototypes-------------------------------*/
void daxpy_(int *N,double *da,double dx[],int *incx,double dy[],
            int *incy);
void dscal_(int *N,double *da,double dx[],int *incx);
void dcopy_(int *N,double dx[],int *incx,double dy[],int *incy);
double ddot_(int *N,double dx[],int *incx,double dy[],int *incy);
double dnrm2_(int *N,double x[],int *incx);
double dasum_(int *N,double dx[],int *incx);
int idamax_(int *N,double dx[],int *incx);

/*************************************************************
 * Let the game begin...
 *************************************************************/
int main()
{
        int i,k,num_mat,tol_vals;
        char filename_A[60];
        char filename_B[60];
        char filenameK[60];
        char input_file[] = "input_data.dat";
```

```
        double TOL, rho;
        double t1_clock, t2_clock;
        double t1_times, t2_times, cpu_time;
        struct tms tb1, tb2;          /* times() needs them  */
        double ticspersec;
        time_t mytime;
        FILE *fp;
        fp = FOPEN(input_file, "r");
        num_mat = 9; /* Set the number of different matrices */
        tol_vals = 5; /* Set the number of different values of TOL */
        /* Set the parameter rho (positive real number) of the
         * Simplified Augmented Lagrangian Method */
        rho = 100.0;
        /* casting because sysconf() returns int */
        ticspersec = (double) sysconf(_SC_CLK_TCK);
        printf("\t\t\t\t\t\t\t\t\t\tMethod:
Simplified Augmented Lagrangian (r = %f)\n", rho);
        blank_line();
        printf("TOL\tRe\tDOF\t    ||f-Au-Bp||\t    ||g-B^Tu||\t\t
||r||\t\t#iter\t    ||x-x*||_1\t\t    ||x-x*||_2\t\t    ||x-x*||_sup\t\t
CPU time(sec)\n");
        blank_line();
        for(i=1;i<=num_mat;i++){
                fscanf(fp,"%s",filename_A);
                fscanf(fp,"%s",filename_B);
                fscanf(fp,"%s",filenameK);
                for(k=1;k<=tol_vals;k++){
                        mytime = time(NULL);
                        fprintf(stderr,ctime(&mytime));
                        fprintf(stderr,"\nnum_mat = %d\t
TOL_val = %d\n\n",i,k);
                        fscanf(fp,"%lf",&TOL);
                        /* casting because times() returns (long) int */
                        t1_times = (double) times(&tb1);
                        t1_clock = (double) clock();
                        method(filenameK,filename_A,filename_B,TOL,rho);
                        t2_clock = (double) clock();
                        t2_times = (double) times(&tb2);
                cpu_time = (double) ((tb2.tms_utime + tb2.tms_stime) -
                                     (tb1.tms_utime + tb1.tms_stime));
                        printf("\t%f\n",cpu_time / ticspersec);
                }
                blank_line();
        }
        FCLOSE(fp);
        printf("\nSimplified Augmented Lagrangian runs ended
successfully!\n");
```

```
        return 0;
}

/* horizontal line */
void blank_line(void)
{
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------");
        printf("-------------------------\n");
}


/* parameterization of the Simplified Augmented Lagrangian Method */
void method(char *filenameK,char *filename_A,char *filename_B,
            const double TOL,const double rho)
{
        int q,N,nzB,nzA,Re;
        int global_nz,n_iter,i;
        double r_a,r_b;
        double norm1,norm2,maxnorm;
        int *I_K,*J_K;
        double *K;
        int *I_A,*J_A,*I_B,*J_B;
        double *A,*B;
        double *f,*g,*f_r;
        double *u,*p;
        read_whole_mat(filenameK,&nzA,&nzB,&N,&q,&I_K,&J_K,&K);
        readmat(filename_A,&N,&nzA,&I_A,&J_A,&A);
        readmat(filename_B,&q,&nzB,&I_B,&J_B,&B);
        u = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(q * sizeof(double));
        f = (double *)MALLOC(N * sizeof(double));
        f_r = (double *)MALLOC(N * sizeof(double));
        g = (double *)MALLOC(q * sizeof(double));
        global_nz = nzA + 2 * nzB;
        construct_rhs(I_K,K,N,q,global_nz,f,g);
        FREE(I_K);
        FREE(J_K);
        FREE(K);
        /* construction of f_r = f + r B g */
        construct_fr(I_B,J_B,B,nzB,N,rho,f,g,f_r);
        /* Initially u and p hold u_0 and p_0 respectively */
        for(i=1;i<=N;i++)
                u[i-1] = 0.0; /* initialize u */
        for(i=1;i<=q;i++)
                p[i-1] = 0.0; /* initialize p */
```

```
        /* Output the computed solution u and p */
        sim_aug_lag(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,q,f_r,g,rho,TOL,
                    u,p,&n_iter);
/*      for(i=1;i<=N;i++)
                printf("u[%d] = %16.9e\n",i-1,u[i-1]);
        for(i=1;i<=q;i++)
                printf("p[%d] = %16.9e\n",i-1,p[i-1]);   */
        /* compute error norms */
        error_norms(u,p,N,q,&norm1,&norm2,&maxnorm);
        /* compute residual norms */
        r_a = mom_norm(I_A,J_A,A,I_B,J_B,B,nzA,nzB,N,f,u,p);
        r_b = comp_norm(I_B,J_B,B,nzB,q,g,u);
        if((strcmp(filename_A,"MtrxA_Re10_N578_nzA3826.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N2178_nzA16818.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N8450_nzA70450.dat") == 0))
                Re = 10;
        else if((strcmp(filename_A,"MtrxA_Re100_N578_nzA3826.dat") == 0)
           || (strcmp(filename_A,"MtrxA_Re100_N2178_nzA16818.dat") == 0)
           || (strcmp(filename_A,"MtrxA_Re100_N8450_nzA70450.dat") == 0))
                Re = 100;
        else
                Re = 1000;
        printf("%.1e\t%d\t%d\t\t%16.9e %16.9e  %16.9e\t%d\t\t%16.9e\t
%16.9e\t%16.9e",TOL,Re,N+q,r_a,r_b,sqrt(r_a * r_a + r_b * r_b),
n_iter - 1,norm1,norm2,maxnorm);
        FREE(I_A);
        FREE(J_A);
        FREE(A);
        FREE(I_B);
        FREE(J_B);
        FREE(B);
        FREE(f);
        FREE(f_r);
        FREE(g);
        FREE(u);
        FREE(p);
}

/* Computation of the error norm : ||x - x*||, where:
 * x  = (u,p)        the computed solution
 * x* = (1,1,...,1)  the exact solution
 * vectors u and p remain unchanged on exit
 * norm1   : returns ||x - x*||_1
 * norm2   : returns ||x - x*||_2
 * maxnorm : returns ||x - x*||_sup    */
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm)
```

```
{
        int i,dim;
        double *err;
        dim = N + q;
        err = (double *)MALLOC(dim * sizeof(double));
        for(i=1;i<=N;i++)
                err[i-1] = u[i-1] - 1.0;
        for(i=N+1;i<=dim;i++)
                err[i-1] = p[i-N-1] - 1.0;
        (*norm1) = m_dasum(dim,err);
        (*norm2) = m_dnrm2(dim,err);
        (*maxnorm) = my_maxnorm(dim,err);
        FREE(err);
}



/* Computation of the norm: ||f - Au - Bp||_2, where u and p are the
 * computed velocity and pressure solutions, respectively.
 * This quantity is a measure of how good the solution satisfies the
 * discretized momentum equation: Au + Bp = f     */
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p)
{
        double scalar,norm;
        double *Au,*Bp;
        Au = (double *)MALLOC(N * sizeof(double));
        Bp = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_A,J_A,A,nzA,u,Au,N,'N'); /* Au = A * u      */
        mat_vec(I_B,J_B,B,nzB,p,Bp,N,'N'); /* Bp = B * p      */
        scalar = -1.0;
        mod_daxpy(N,scalar,f,Au); /* Au = f - Au       */
        scalar = -1.0;
        mod_daxpy(N,scalar,Au,Bp); /* Bp = Au - Bp = (f - Au) - Bp */
        norm = m_dnrm2(N,Bp); /* norm = ||Bp|| = ||f - Au - Bp|| */
        FREE(Au);
        FREE(Bp);
        return norm;
}



/* Computation of the norm: ||g - B**T u||_2, where u is the computed
 * velocity solution.
 * This quantity is a measure of how good the solution satisfies the
 * discretized incompressibility condition: B**T u = g */
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,const int q,
```

```
                    double *g,double *u)
{
        double scalar,norm;
        double *Btu;
        Btu = (double *)MALLOC(q * sizeof(double));
        mat_vec(I_B,J_B,B,nzB,u,Btu,q,'T'); /* Btu = B**T * u  */
        scalar = -1.0;
        mod_daxpy(q,scalar,g,Btu);  /* Btu = g - Btu  */
        norm = m_dnrm2(q,Btu); /* norm = ||Btu|| = ||g - B**T * u|| */
        FREE(Btu);
        return norm;
}


/* Construction of the vector f_r = f + r B * g.
 * Allocation of f_r in main() */
void construct_fr(int *I_B,int *J_B,double *B,const int nzB,
                  const int N,const double rho,double *f,
                  double *g,double *f_r)
{
        double scalar;
        mat_vec(I_B,J_B,B,nzB,g,f_r,N,'N'); /* f_r = B g  */
        scalar = rho;
        mod_daxpy(N,scalar,f,f_r); /* f_r = f + r f_r  */
}


/* On entry : u and p hold u_0 and p_0 respectively,
 * allocation and initialization in main()
 * On exit  : u and p hold the computed velocity and pressure,
 * respectively. */
void sim_aug_lag(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                 double *B,const int nzA,const int nzB,const int N,
                 const int q,double *f_r,double *g,const double rho,
                 const double TOL,double *u,double *p,int *n_iter)
{
        int max_iter,iter;
        double scalar,norm_R_p,stop_TOL,rel_res;
        double *Bvec,*Btrans_vec,*R_p,*rhs;
        Bvec = (double *)MALLOC(N * sizeof(double));
        Btrans_vec = (double *)MALLOC(q * sizeof(double));
        R_p = (double *)MALLOC(q * sizeof(double));
        rhs = (double *)MALLOC(N * sizeof(double));
        m_dcopy(q,g,R_p); /*  R_p = g   */
        /* Btrans_vec = B^T u_0 */
        mat_vec(I_B,J_B,B,nzB,u,Btrans_vec,q,'T');
        /* R_p = Btrans_vec - R_p, R_p = B^T u_0 - g  */
```

```
        scalar = -1.0;
        mod_daxpy(q,scalar,Btrans_vec,R_p);
        norm_R_p = m_dnrm2(q,R_p); /*  norm_R_p = ||R_p||_2  */
        (*n_iter) = 1;
        while(norm_R_p > TOL){
        /*      printf("Simplified AL iteration: %d\n\n",(*n_iter)); */
                mat_vec(I_B,J_B,B,nzB,p,Bvec,N,'N'); /* Bvec = B p */
                m_dcopy(N,Bvec,rhs); /* rhs = Bvec  */
                /* rhs = f_r - rhs,  rhs = f_r - B p */
                scalar = -1.0;
                mod_daxpy(N,scalar,f_r,rhs);
                /* Initial guess for CG,u from previous iteration,
                 * first time u_0 */
                stop_TOL = TOL;
                max_iter = 2 * N;
                /* Solve A_r u = rhs,  rhs = f_r - B p */
                CG(I_A,J_A,A,I_B,J_B,B,nzA,nzB,u,rhs,rho,stop_TOL,
                    max_iter,N,q,&rel_res,&iter);
        /*      printf("CG1 Iterations = %d\n",iter-1);    */
                m_dcopy(q,g,R_p); /*  R_p = g    */
                /* Btrans_vec = B^T u  */
                mat_vec(I_B,J_B,B,nzB,u,Btrans_vec,q,'T');
                /* R_p = Btrans_vec - R_p,  R_p = B^T u - g  */
                scalar = -1.0;
                mod_daxpy(q,scalar,Btrans_vec,R_p);
                scalar = rho;
                m_daxpy(q,scalar,R_p,p); /*  p = r R_p + p    */
                norm_R_p = m_dnrm2(q,R_p); /* norm_R_p = ||R_p||_2 */
                (*n_iter)++;
        }
        mat_vec(I_B,J_B,B,nzB,p,Bvec,N,'N'); /* Bvec = B p */
        m_dcopy(N,Bvec,rhs); /* rhs = Bvec  */
        /* rhs = f_r - rhs,  rhs = f_r - B p */
        scalar = -1.0;
        mod_daxpy(N,scalar,f_r,rhs);
        /* Initial guess for CG,u from the last loop  */
        stop_TOL = TOL;
        max_iter = 2 * N;
        /* Solve A_r u = rhs,  rhs = f_r - B p */
        CG(I_A,J_A,A,I_B,J_B,B,nzA,nzB,u,rhs,rho,stop_TOL,max_iter,
            N,q,&rel_res,&iter);
/*      printf("Last_CG Iterations = %d\n",iter-1);  */
        FREE(Bvec);
        FREE(Btrans_vec);
        FREE(R_p);
        FREE(rhs);
}
```

```
/* On entry x contains the initial guess,on exit the solution     */
void CG(int *I_A,int *J_A,double *A,int *I_B,int *J_B,double *B,
        const int nzA,const int nzB,double *x,double *rhs,
        const double rho,const double stop_TOL,const int max_iter,
        const int N,const int q,double *rel_res,int *iter)
{
        double norm_rhs,scalar,Ap_p,rold_rold;
        double cg_alpha,rnew_rnew,cg_beta,norm_r_new;
        double *r,*p,*Ap;
        r = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(N * sizeof(double));
        Ap = (double *)MALLOC(N * sizeof(double));
        norm_rhs = m_dnrm2(N,rhs); /* norm_rhs = ||rhs||_2  */
        /* r = A_r x */
        CG_mat_vec(I_A,J_A,A,I_B,J_B,B,nzA,nzB,rho,N,q,x,r);
        /* r = rhs + scalar r, r = rhs - r  */
        scalar = -1.0;
        mod_daxpy(N,scalar,rhs,r);
        m_dcopy(N,r,p); /* p = r  */
        (*rel_res) = 2.0;
        (*iter) = 1;
      while(((*rel_res) > (stop_TOL * norm_rhs)) && ((*iter) <= max_iter)){
                /* Ap = A_r p  */
                CG_mat_vec(I_A,J_A,A,I_B,J_B,B,nzA,nzB,rho,N,q,p,Ap);
                Ap_p = m_ddot(N,Ap,p); /*  Ap_p = (Ap,p) */
                rold_rold = m_ddot(N,r,r); /* rold_rold = (r,r) */
                cg_alpha = rold_rold/Ap_p;
                /*  x = cg_alpha p + x,update solution vector */
                scalar = cg_alpha;
                m_daxpy(N,scalar,p,x);
                /*  r =- cg_alpha Ap + r,update residual vector */
                scalar =-cg_alpha;
                m_daxpy(N,scalar,Ap,r);
                rnew_rnew = m_ddot(N,r,r); /* rnew_rnew = (r,r) */
                cg_beta = rnew_rnew/rold_rold;
                /* p = r(new) + cg_beta p,update search direction */
                scalar = cg_beta;
                mod_daxpy(N,scalar,r,p);
                norm_r_new = m_dnrm2(N,r); /* norm_r_new = ||r(new)|| */
                (*rel_res) = norm_r_new;
                (*iter)++;
        }
        FREE(r);
        FREE(p);
        FREE(Ap);
        if((*iter) > max_iter){
```

```
                fprintf(stderr,"Maximum number of iterations exceeded!\n");
                exit(2);
        }
}

/* Given x computes Ar_x = A x + r B B^T x ,
 * x remains unchanged on exit */
void CG_mat_vec(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const double rho,
                const int N,const int q,double *x,double *Ar_x)
{
        double scalar;
        double *y,*w;
        y = (double *)MALLOC(q * sizeof(double));
        w = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_B,J_B,B,nzB,x,y,q,'T'); /* y = B^T x  */
        mat_vec(I_B,J_B,B,nzB,y,Ar_x,N,'N'); /* Ar_x = B y  */
        mat_vec(I_A,J_A,A,nzA,x,w,N,'N'); /* w = A x  */
        scalar = rho;
        mod_daxpy(N,scalar,w,Ar_x); /* Ar_x = w + r Ar_x  */
        FREE(y);
        FREE(w);
}


/* Computes matrix-vector product: y = mat x or y = mat^T x,
 * mat is represented in coordinate sparse format.
 * dimy is the dimension of the output vector y,
 * x remains unchanged on exit. */
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose)
{
        int k;
        for(k=1;k<=dimy;k++)
                y[k-1] = 0.0;
        if(transpose == 'N'){     /*  Compute y = mat * x        */
                for(k=1;k<=nz;k++)
                        y[I_mat[k-1]-1] += mat[k-1] * x[J_mat[k-1]-1];
        }
        else if(transpose == 'T'){    /* Compute y = mat^T * x      */
                for(k=1;k<=nz;k++)
                        y[J_mat[k-1]-1] += mat[k-1] * x[I_mat[k-1]-1];
        }
        else {
      fprintf(stderr,"Wrong option for matrix-vector multiplication!\n");
      exit(2);
        }
```

```
}


/* The right hand side vector of the linear system is constructed
 * such that its solution is the vector (1,1,...,1) */
void construct_rhs(int *I_K,double *K,const int N,const int q,
                   const int global_nz,double *f,double *g)
{
        int i,k,dim;
        double xsum;
        dim = N + q;
        for(i=1;i<=N;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                f[i-1] = xsum;
        }
        for(i=N+1;i<=dim;i++){
                xsum = 0.0;
                for(k=1;k<=global_nz;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                g[i-N-1] = xsum;
        }
}

/* reading of the nonzero elements of the whole matrix,
 * stored in coordinate format, in global enumeration */
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals)
{
        int i,j,k,p;
        int non_zero_A,non_zero_B,dim_A,rank_B;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&non_zero_A);
        fscanf(fp,"%d",&non_zero_B);
        fscanf(fp,"%d",&dim_A);
        fscanf(fp,"%d",&rank_B);
        (*nzA) = non_zero_A;
        (*nzB) = non_zero_B;
```

```
        (*N) = dim_A;
        (*q) = rank_B;
        p = non_zero_A + 2 * non_zero_B;
        tmp1 = (int *)MALLOC(p * sizeof(int));
        tmp2 = (int *)MALLOC(p * sizeof(int));
        tmp3 = (double *)MALLOC(p * sizeof(double));
        for(k=1;k<=p;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* reading of the nonzero elements of matrix A or B,
 * stored in coordinate format, in local enumeration */
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals)
{
        int i,j,k;
        int order,nzero;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&order);
        fscanf(fp,"%d",&nzero);
        (*q) = order;
        (*nz) = nzero;
        tmp1 = (int *)MALLOC((*nz) * sizeof(int));
        tmp2 = (int *)MALLOC((*nz) * sizeof(int));
        tmp3 = (double *)MALLOC((*nz) * sizeof(double));
        for(k=1;k<=nzero;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
```

```
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* y = x + scalar * y ,x remains unchanged on exit */
void mod_daxpy(const int dim,const double scalar,double *x,double *y)
{
        double alpha;
        m_dscal(dim,scalar,y); /* y = scalar * y     */
        alpha = 1.0;
        m_daxpy(dim,alpha,x,y);/* Computes y = alpha * x + y,y = x + y */
}

/* Computes l_2 norm of x : ||x||_2 */
double m_dnrm2(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dnrm2_(&N,x,&incx);
        return result;
}

/* Computes the dot product (x,y)_2  */
double m_ddot(const int dim,double *x,double *y)
{
        int N,incx,incy;
        double result;
        incx = 1;
        incy = 1;
        N = dim;
        result = ddot_(&N,x,&incx,y,&incy);
        return result;
}

/* Copies vector x to y, y = x,x remains unchanged on exit */
void m_dcopy(const int dim,double *x,double *y)
{
        int incx,incy,N;
        N = dim;
        incx = 1;
        incy = 1;
        dcopy_(&N,x,&incx,y,&incy);
}
```

```
/* On exit vector = scalar * vector */
void m_dscal(const int dim,const double scalar,double *vector)
{
        int N,incx;
        double da = scalar;
        incx = 1;
        N = dim;
        dscal_(&N,&da,vector,&incx);
}

/* y = scalar * x + y,x remains unchanged on exit */
void m_daxpy(const int dim,const double scalar,double *x,double *y)
{
        int incx,incy,N;
        double da = scalar;
        incx = 1;
        incy = 1;
        N = dim;
        daxpy_(&N,&da,x,&incx,y,&incy); /*  y = da * x + y  */
}

/* Computes l_1 norm of x : ||x||_1 */
double m_dasum(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dasum_(&N,x,&incx);
        return result;
}

/*  returns the index of element of vector x,having max. absolute value.
 *  -1 shifting for C compatibility */
int m_idamax(const int dim,double *x)
{
        int N,incx;
        int pos;
        N = dim;
        incx = 1;
        pos = idamax_(&N,x,&incx);
        pos--;
        return pos;
}

/* Computes maximum norm of x : ||x||_sup */
double my_maxnorm(const int dim,double *x)
```

```
{
        int pos;
        double result;
        pos = m_idamax(dim,x);
        result = fabs(x[pos]);
        return result;
}
```

## Α΄.5 Κώδικες για τη μέθοδο MA57

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο MA57.

```
/* Author      : Manos Psycharis
 * Purpose     : Master Thesis
 * Date        : 03/03/2010
 *
 * Compile with MeTiS (use library libmetis.a) :
 *
 * gcc -c -Wall -ansi -pedantic direct_sol_cma57.c memory.c
 * g95 -c ma57.f ma57dp.f
 * g95 -o sol direct_sol_cma57.o memory.o ma57.o ma57dp.o libmetis.a
 *     -lblas
 *
 * Compile WITHOUT MeTiS (use the replacement routine metis.f) :
 *
 * gcc -c -Wall -ansi -pedantic direct_sol_cma57.c memory.c
 * g95 -c ma57.f ma57dp.f metis.f
 * g95 -o sol direct_sol_cma57.o memory.o ma57.o ma57dp.o metis.o -lblas
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>        /* because we use clock()    */
#include <sys/times.h>   /* because we use times()    */
#include <unistd.h>      /* because we use sysconf()  */
#include <string.h>      /* becuase we use strcmp()   */
#include "memory.h"
#ifndef max
    #define max( a, b ) ( ((a) > (b)) ? (a) : (b) )
#endif

/*-------------------------18 Prototypes-------------------------*/
void blank_line(void);
void method(char *filenameK,char *filename_A,char *filename_B);
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm);
```

```
double mom_norm(int *I_A,int *J_A,double *A,int *I_B,int *J_B,
                double *B,const int nzA,const int nzB,const int N,
                double *f,double *u,double *p);
double comp_norm(int *I_B,int *J_B,double *B,const int nzB,const int q,
                 double *g,double *u);
void hold_rhs_sol(double *RHS,double *first,double *second,const int N,
                  const int q);
void construct_HSL_mat(int *I_K,int *J_K,double *K,const int global_nz,
                       int *IRN,int *JCN,double *A);
void construct_rhs(int *I_K,double *K,const int NDIM,const int global_nz,
                   double *RHS);
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals);
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals);
double m_dnrm2(const int dim,double *x);
void m_daxpy(const int dim,const double scalar,double *x,double *y);
void m_dscal(const int dim,const double scalar,double *vector);
void mod_daxpy(const int dim,const double scalar,double *x,double *y);
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose);
double m_dasum(const int dim,double *x);
int m_idamax(const int dim,double *x);
double my_maxnorm(const int dim,double *x);
/*-------------------------MA57 Prototypes-------------------*/
void ma57id_(double CNTL[],int ICNTL[]);
void ma57ad_(int *N,int *NE,int IRN[],int JCN[],int *LKEEP,
             int KEEP[],int IWORK[],int ICNTL[],int INFO[],
             double RINFO[]);
void ma57bd_(int *N,int *NE,double A[],double FACT[],int *LFACT,
             int IFACT[],int *LIFACT,int *LKEEP,int KEEP[],int IWORK[],
             int ICNTL[],double CNTL[],int INFO[],double RINFO[]);
void ma57cd_(int *JOB,int *N,double FACT[],int *LFACT,int IFACT[],
             int *LIFACT,int *NRHS,double RHS[],int *LRHS,double WORK[],
             int *LWORK,int IWORK[],int ICNTL[],int INFO[]);
/*-----------------5 BLAS Prototypes-------------------------*/
void daxpy_(int *N,double *da,double dx[],int *incx,double dy[],
            int *incy);
void dscal_(int *N,double *da,double dx[],int *incx);
double dnrm2_(int *N,double x[],int *incx);
double dasum_(int *N,double dx[],int *incx);
int idamax_(int *N,double dx[],int *incx);

/**************************************************************
 * Let the game begin...
 **************************************************************/
int main()
{
```

```
        int i,num_mat;
        char filename_A[60];
        char filename_B[60];
        char filenameK[60];
        char input_file[] = "ma57_input_data.dat";
        double t1_clock,t2_clock;
        double t1_times,t2_times,cpu_time;
        struct tms tb1,tb2;          /* times() needs them  */
        double ticspersec;
        FILE *fp;
        fp = FOPEN(input_file,"r");
        num_mat = 9; /* Set the number of different matrices */
        /* casting because sysconf() returns int */
        ticspersec = (double) sysconf(_SC_CLK_TCK);
        printf("\t\t\t\t\t\t\t\t\t\tMethod: MA57 (Direct Solution)\n");
        blank_line();
        printf("Re\tDOF\t   ||f-Au-Bp||\t    ||g-B^Tu||\t\t||r||\t\t
||x-x*||_1\t\t   ||x-x*||_2\t\t   ||x-x*||_sup\t\tCPU time(sec)\n");
        blank_line();
        for(i=1;i<=num_mat;i++){
                fscanf(fp,"%s",filename_A);
                fscanf(fp,"%s",filename_B);
                fscanf(fp,"%s",filenameK);
                /* casting because times() returns (long) int */
                t1_times = (double) times(&tb1);
                t1_clock = (double) clock();
                method(filenameK,filename_A,filename_B);
                t2_clock = (double) clock();
                t2_times = (double) times(&tb2);
                cpu_time = (double) ((tb2.tms_utime + tb2.tms_stime) -
                                     (tb1.tms_utime + tb1.tms_stime));
                printf("\t%f\n",cpu_time / ticspersec);
                if((i % 3) == 0 )
                        blank_line();
        }
        FCLOSE(fp);
        printf("\nMA57 Direct Solution runs ended successfully!\n");
        return 0;
}

/* horizontal line */
void blank_line(void)
{
        printf("-------------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------------");
        printf("-------------------------------------------------------------------------------");
        printf("------------------------\n");
```

```
}


/* parameterization of MA57 Direct Solution routines */
void method(char *filenameK,char *filename_A,char *filename_B)
{
        int nzA,nzB,N,q,global_nz,Re;
        int max_NDIM_NE;
        int *I_K,*J_K;
        double *K;
        int *I_A,*J_A,*I_B,*J_B;
        double *A_mat,*B_mat;
        double *f,*g,*u,*p;
        double r_a,r_b;
        double norm1,norm2,maxnorm;
/*---------- Metavlhtes MA57 ----------*/
        /* MA57ID variables */
        double *CNTL;
        int *ICNTL;
        /* MA57AD variables */
        int NDIM,NE,LKEEP;
        int *IRN,*JCN,*KEEP,*IWORK,*INFO;
        double *RINFO;
        /* MA57BD variables */
        int LFACT,LIFACT;
        int *IFACT;
        double *A,*FACT;
        /* MA57CD variables */
        int JOB,NRHS,LRHS,LWORK;
        double *RHS,*WORK;
/*-------------------------------------------------*/
        read_whole_mat(filenameK,&nzA,&nzB,&N,&q,&I_K,&J_K,&K);
        global_nz = nzA + 2 * nzB;
        NDIM = N + q; /* dimension of the whole matrix */
        LRHS = NDIM;
        RHS = (double *)MALLOC(LRHS * sizeof(double));
        /* construction of RHS    */
        construct_rhs(I_K,K,NDIM,global_nz,RHS);
        /* NE is the number of nonzero elements in the upper
         * triangle of global matrix, including the non-zero
         * diagonal elements */
        NE = (nzA + N)/2 + nzB;
        IRN = (int *)MALLOC(NE * sizeof(int));
        JCN = (int *)MALLOC(NE * sizeof(int));
        A = (double *)MALLOC(NE * sizeof(double));
        /* fills arrays IRN,JCN and the global matrix A */
        construct_HSL_mat(I_K,J_K,K,global_nz,IRN,JCN,A);
```

```
        FREE(I_K);
        FREE(J_K); /* No more need for the global matrix */
        FREE(K);
        CNTL = (double *)MALLOC(5 * sizeof(double));
        ICNTL = (int *)MALLOC(20 * sizeof(int));
        /* Set default values for control parameters.  */
        ma57id_(CNTL,ICNTL);
        /*  Request MeTiS ordering  */
        ICNTL[6-1] = 4; /* ICNTL(6)= 4 ,-1 shifting */
        /* Ask for full printing from MA57 package */
/*      ICNTL[5-1] = 4; */ /* ICNTL(5) = 4 ,-1 shifting */
        max_NDIM_NE = max(NDIM,NE);
        /* This is the minimum length for integer array KEEP */
        LKEEP = 5 * NDIM + NE + max_NDIM_NE + 42;
        /* LKEEP = LKEEP + 2 * NDIM; allocate 2*N more space for safety */
        LKEEP += 2 * NDIM;
        KEEP = (int *)MALLOC(LKEEP * sizeof(int));
        IWORK = (int *)MALLOC((5 * NDIM) * sizeof(int));
        INFO = (int *)MALLOC(40 * sizeof(int));
        RINFO = (double *)MALLOC(20 * sizeof(double));
        /* Analyse sparsity pattern */
        ma57ad_(&NDIM,&NE,IRN,JCN,&LKEEP,KEEP,IWORK,ICNTL,INFO,RINFO);
/*      fprintf(stderr,"INFO(36) = %d\n",INFO[36-1]);    */
        if(INFO[1-1] < 0){
                fprintf(stderr,"An error is occured in MA57AD!\n");
                exit(2);
        }
        /* LFACT = INFO(9) as output from ma57ad_()
         * is the minimun value required for LFACT,-1 shifting */
        LFACT = INFO[9-1];
        /* allocate more space for safety
         * (sometimes LFACT requires a value
         * more than twice INFO(9) )   */
        LFACT = 2 * LFACT + 2 * NDIM;
        FACT = (double *)MALLOC(LFACT * sizeof(double));
        /* LIFACT = INFO(10) as output from ma57ad_()
         * is the minimun value required for LIFACT,-1 shifting */
        LIFACT = INFO[10-1];
        /* allocate more space for safety,
         * because numerical pivoting may
         * increase storage requirements */
        LIFACT = 2 * LIFACT + 2 * NDIM;
        IFACT = (int *)MALLOC(LIFACT * sizeof(int));
        /* Factorize matrix */
        ma57bd_(&NDIM,&NE,A,FACT,&LFACT,IFACT,&LIFACT,&LKEEP,KEEP,
                IWORK,ICNTL,CNTL,INFO,RINFO);
        if(INFO[1-1] < 0){
```

```
                fprintf(stderr,"An error is occured in MA57BD!\n");
                exit(2);
        }
        f = (double *)MALLOC(N * sizeof(double));
        g = (double *)MALLOC(q * sizeof(double));
        hold_rhs_sol(RHS,f,g,N,q); /* constructs f and g */
        /* Solve the equations */
        JOB = 1;
        NRHS = 1;
        LWORK = NDIM * NRHS;
        WORK = (double *)MALLOC(LWORK * sizeof(double));
        /* solution written in RHS */
        ma57cd_(&JOB,&NDIM,FACT,&LFACT,IFACT,&LIFACT,&NRHS,RHS,&LRHS,
                WORK,&LWORK,IWORK,ICNTL,INFO);
        if(INFO[1-1] < 0){
                fprintf(stderr,"An error is occured in MA57CD!\n");
                exit(2);
        }
/*      for(i=1;i<=NDIM;i++)
                printf("Sol[%d]=%16.9e\n",i-1,RHS[i-1]);   */
        u = (double *)MALLOC(N * sizeof(double));
        p = (double *)MALLOC(q * sizeof(double));
        hold_rhs_sol(RHS,u,p,N,q); /* constructs u and p  */
        readmat(filename_A,&N,&nzA,&I_A,&J_A,&A_mat);
        readmat(filename_B,&q,&nzB,&I_B,&J_B,&B_mat);
        /* compute error norms */
        error_norms(u,p,N,q,&norm1,&norm2,&maxnorm);
        /* compute residual norms */
        r_a = mom_norm(I_A,J_A,A_mat,I_B,J_B,B_mat,nzA,nzB,N,f,u,p);
        r_b = comp_norm(I_B,J_B,B_mat,nzB,q,g,u);
        if((strcmp(filename_A,"MtrxA_Re10_N578_nzA3826.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N2178_nzA16818.dat") == 0) ||
           (strcmp(filename_A,"MtrxA_Re10_N8450_nzA70450.dat") == 0))
                Re = 10;
        else if((strcmp(filename_A,"MtrxA_Re100_N578_nzA3826.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re100_N2178_nzA16818.dat") == 0)
        || (strcmp(filename_A,"MtrxA_Re100_N8450_nzA70450.dat") == 0))
                Re = 100;
        else
                Re = 1000;
        printf("%d\t%d\t%16.9e %16.9e  %16.9e\t%16.9e\t%16.9e\t%16.9e",
Re,N+q,r_a,r_b,sqrt(r_a * r_a + r_b * r_b),norm1,norm2,maxnorm);
/*---------Memory deallocation ---------*/
/*---------MA57 relevant vectors-------*/
        FREE(A);
        FREE(RHS);
        FREE(IRN);
```

```
        FREE(JCN);
        FREE(CNTL);
        FREE(ICNTL);
        FREE(KEEP);
        FREE(IWORK);
        FREE(INFO);
        FREE(RINFO);
        FREE(FACT);
        FREE(IFACT);
        FREE(WORK);
/*---------- Other vectors ---------*/
        FREE(I_A);
        FREE(J_A);
        FREE(A_mat);
        FREE(I_B);
        FREE(J_B);
        FREE(B_mat);
        FREE(f);
        FREE(g);
        FREE(u);
        FREE(p);
}


/* Computation of the error norm : ||x - x*||, where:
 * x  = (u,p)        the computed solution
 * x* = (1,1,...,1) the exact solution
 * vectors u and p remain unchanged on exit
 * norm1   : returns ||x - x*||_1
 * norm2   : returns ||x - x*||_2
 * maxnorm : returns ||x - x*||_sup    */
void error_norms(double *u,double *p,const int N,const int q,
                 double *norm1,double *norm2,double *maxnorm)
{
        int i,dim;
        double *err;
        dim = N + q;
        err = (double *)MALLOC(dim * sizeof(double));
        for(i=1;i<=N;i++)
                err[i-1] = u[i-1] - 1.0;
        for(i=N+1;i<=dim;i++)
                err[i-1] = p[i-N-1] - 1.0;
        (*norm1) = m_dasum(dim,err);
        (*norm2) = m_dnrm2(dim,err);
        (*maxnorm) = my_maxnorm(dim,err);
        FREE(err);
}
```

```
/* Computation of the norm: ||f - Au - Bp||_2, where u and p are the
 * computed velocity and pressure solutions, respectively.
 * This quantity is a measure of how good the solution satisfies the
 * discretized momentum equation: Au + Bp = f     */
double mom_norm(int *I_A, int *J_A, double *A, int *I_B, int *J_B,
                double *B, const int nzA, const int nzB, const int N,
                double *f, double *u, double *p)
{
        double scalar, norm;
        double *Au, *Bp;
        Au = (double *)MALLOC(N * sizeof(double));
        Bp = (double *)MALLOC(N * sizeof(double));
        mat_vec(I_A, J_A, A, nzA, u, Au, N, 'N'); /* Au = A * u     */
        mat_vec(I_B, J_B, B, nzB, p, Bp, N, 'N'); /* Bp = B * p     */
        scalar = -1.0;
        mod_daxpy(N, scalar, f, Au); /* Au = f - Au      */
        scalar = -1.0;
        mod_daxpy(N, scalar, Au, Bp); /* Bp = Au - Bp = (f - Au) - Bp */
        norm = m_dnrm2(N, Bp); /* norm = ||Bp|| = ||f - Au - Bp|| */
        FREE(Au);
        FREE(Bp);
        return norm;
}

/* Computation of the norm: ||g - B**T u||_2, where u is the computed
 * velocity solution.
 * This quantity is a measure of how good the solution satisfies the
 * discretized incompressibility condition: B**T u = g */
double comp_norm(int *I_B, int *J_B, double *B, const int nzB, const int q,
                 double *g, double *u)
{
        double scalar, norm;
        double *Btu;
        Btu = (double *)MALLOC(q * sizeof(double));
        mat_vec(I_B, J_B, B, nzB, u, Btu, q, 'T'); /* Btu = B**T * u  */
        scalar = -1.0;
        mod_daxpy(q, scalar, g, Btu);   /* Btu = g - Btu  */
        norm = m_dnrm2(q, Btu); /* norm = ||Btu|| = ||g - B**T * u|| */
        FREE(Btu);
        return norm;
}


/* recover f and g or u and p from RHS */
void hold_rhs_sol(double *RHS, double *first, double *second, const int N,
```

```
                    const int q)
{
        int i,dim;
        dim = N + q;
        for(i=1;i<=N;i++)
                first[i-1] = RHS[i-1];
        for(i=N+1;i<=dim;i++)
                second[i-N-1] = RHS[i-1];
}

/* Stores the nonzero elements of the upper triangle of the symmetric
 * matrix (including the nonzero diagonal elements) in coordinate
 * sparse format */
void construct_HSL_mat(int *I_K,int *J_K,double *K,const int global_nz,
                       int *IRN,int *JCN,double *A)
{
        int k,i,j,m,p;
        p = global_nz;
        m = 1;
        for(k=1;k<=p;k++){
                i = I_K[k-1];
                j = J_K[k-1];
                if(i <= j){
                        IRN[m-1] = i;
                        JCN[m-1] = j;
                        A[m-1] = K[k-1];
                        m++;
                }
        }
}

/* The right hand side vector of the linear system is constructed
 * such that its solution is the vector (1,1,...,1) */
void construct_rhs(int *I_K,double *K,const int NDIM,const int global_nz,
                   double *RHS)
{
        int i,k,p;
        double xsum;
        p = global_nz;
        for(i=1;i<=NDIM;i++){
                xsum = 0.0;
                for(k=1;k<=p;k++){
                        if(I_K[k-1] == i)
                                xsum += K[k-1];
                }
                RHS[i-1] = xsum;
        }
```

```
}


/* reading of the nonzero elements of the whole matrix,
 * stored in coordinate format,in global enumeration */
void read_whole_mat(char *filename,int *nzA,int *nzB,int *N,int *q,
                    int **I,int **J,double **vals)
{
        int i,j,k,p;
        int non_zero_A,non_zero_B,dim_A,rank_B;
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&non_zero_A);
        fscanf(fp,"%d",&non_zero_B);
        fscanf(fp,"%d",&dim_A);
        fscanf(fp,"%d",&rank_B);
        (*nzA) = non_zero_A;
        (*nzB) = non_zero_B;
        (*N) = dim_A;
        (*q) = rank_B;
        p = non_zero_A + 2 * non_zero_B;
        tmp1 = (int *)MALLOC(p * sizeof(int));
        tmp2 = (int *)MALLOC(p * sizeof(int));
        tmp3 = (double *)MALLOC(p * sizeof(double));
        for(k=1;k<=p;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* reading of the nonzero elements of matrix A or B,
 * stored in coordinate format, in local enumeration */
void readmat(char *filename,int *q,int *nz,int **I,int **J,double **vals)
{
        int i,j,k;
        int order,nzero;
```

```
        double x;
        int *tmp1, *tmp2;
        double *tmp3;
        FILE *fp;
        fp = FOPEN(filename,"r");
        fscanf(fp,"%d",&order);
        fscanf(fp,"%d",&nzero);
        (*q) = order;
        (*nz) = nzero;
        tmp1 = (int *)MALLOC((*nz) * sizeof(int));
        tmp2 = (int *)MALLOC((*nz) * sizeof(int));
        tmp3 = (double *)MALLOC((*nz) * sizeof(double));
        for(k=1;k<=nzero;k++){
                fscanf(fp,"%d",&i);
                fscanf(fp,"%d",&j);
                fscanf(fp,"%lf",&x);
                tmp1[k-1] = i;
                tmp2[k-1] = j;
                tmp3[k-1] = x;
        }
        (*I) = tmp1;
        (*J) = tmp2;
        (*vals) = tmp3;
        FCLOSE(fp);
}

/* Computes l_2 norm of x : ||x||_2 */
double m_dnrm2(const int dim,double *x)
{
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dnrm2_(&N,x,&incx);
        return result;
}

/* y = scalar * x + y,x remains unchanged on exit */
void m_daxpy(const int dim,const double scalar,double *x,double *y)
{
        int incx,incy,N;
        double da = scalar;
        incx = 1;
        incy = 1;
        N = dim;
        daxpy_(&N,&da,x,&incx,y,&incy); /*  y = da * x + y  */
}
```

```
/* On exit vector = scalar * vector  */
void m_dscal(const int dim,const double scalar,double *vector)
{
        int N,incx;
        double da = scalar;
        incx = 1;
        N = dim;
        dscal_(&N,&da,vector,&incx);
}

/* y = x + scalar * y ,x remains unchanged on exit */
void mod_daxpy(const int dim,const double scalar,double *x,double *y)
{
        double alpha;
        m_dscal(dim,scalar,y); /* y = scalar * y    */
        alpha = 1.0;
        m_daxpy(dim,alpha,x,y);/* Computes y = alpha * x + y,y = x + y */
}

/* Computes matrix-vector product: y = mat x or y = mat^T x,
 * mat is represented in coordinate sparse format.
 * dimy is the dimension of the output vector y,
 * x remains unchanged on exit */
void mat_vec(int *I_mat,int *J_mat,double *mat,const int nz,double *x,
             double *y,const int dimy,char transpose)
{
        int k;
        for(k=1;k<=dimy;k++)
                y[k-1] = 0.0;
        if(transpose == 'N'){     /*  Compute y = mat * x        */
                for(k=1;k<=nz;k++)
                        y[I_mat[k-1]-1] += mat[k-1] * x[J_mat[k-1]-1];
        }
        else if(transpose == 'T'){    /* Compute y = mat^T * x      */
                for(k=1;k<=nz;k++)
                        y[J_mat[k-1]-1] += mat[k-1] * x[I_mat[k-1]-1];
        }
        else {
      fprintf(stderr,"Wrong option for matrix-vector multiplication!\n");
      exit(2);
        }
}

/* Computes l_1 norm of x : ||x||_1 */
double m_dasum(const int dim,double *x)
{
```

```
        int N,incx;
        double result;
        N = dim;
        incx = 1;
        result = dasum_(&N,x,&incx);
        return result;
}

/*  returns the index of element of vector x,having max. absolute value.
 *  -1 shifting for C compatibility */
int m_idamax(const int dim,double *x)
{
        int N,incx;
        int pos;
        N = dim;
        incx = 1;
        pos = idamax_(&N,x,&incx);
        pos--;
        return pos;
}

/* Computes maximum norm of x : ||x||_sup */
double my_maxnorm(const int dim,double *x)
{
        int pos;
        double result;
        pos = m_idamax(dim,x);
        result = fabs(x[pos]);
        return result;
}
```

## Α΄.6 Κώδικες για τη μέθοδο Preconditioned MINRES

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο Preconditioned MINRES.

```
c234567
c * Author      : Manos Psycharis
c * Purpose     : Master Thesis
c * Date        : 03/03/2010
c *
c * Compile      g95 -o file MINRES_prec.f minres.f minresblas.f ma27.f
c                ma27dp.f syprec.f -lblas
      program main
      implicit none
```

```
      character*30 filename
      character*60 filenameK,filenameA,filenameB
      character*60 stringK,stringA,stringB
      integer i,k,band,nummat,tolvals
      double precision r,TOL
      real timebegin,timeend
      filename = 'input_data.dat'
      open(unit = 50,file = filename,status ='old')
c     Set the number of matrices
      nummat = 9
c     Set the number of different values of TOL
      tolvals = 5
c     Set the preconditioner parameter band = 0,1,2,3
c     (explained in subroutine method)
      band = 2
c     Set the preconditioner parameter r
      r = 0.22d0
      write(6,100) 'Method : Prec MINRES ( r = ',r,',band = ',band,')'
      call blankline()
      write(6,200) 'TOL','DOF','||f-Au-Bp||','||g-B^Tu||','||r||'
     $,'#iter','||x-x*||_1','||x-x*||_2','||x-x*||_sup'
     $,'CPU time(sec)','filenameA'
      call blankline()
      do i=1,nummat
            read(50,*) filenameA
            read(50,*) filenameB
            read(50,*) filenameK
            stringA = trim(adjustl(filenameA))
            stringB = trim(adjustl(filenameB))
            stringK = trim(adjustl(filenameK))
            do k=1,tolvals
                 read(50,*) TOL
                 call cpu_time(timebegin)
c                 call method(filenameK,filenameA,filenameB,band,r,TOL)
                 call method(stringK,stringA,stringB,band,r,TOL)
                 call cpu_time(timeend)
                 write(6,300,advance="no") timeend - timebegin
                 write(6,400) trim(filenameA)
            enddo
            call blankline()
      enddo
      close(50)
      write(6,*)
      write(6,*) 'Prec MINRES runs ended successfully!'
 100  format(80x,a,e16.9,a,i1,a)
 200  format(1x,a,7x,a,9x,a,7x,a,12x,a,11x,a,7x,a,
     $15x,a,15x,a,9x,a,9x,a)
```

```
 300  format(7x,f10.7)
 400  format(12x,a)
      stop
      end

c     horizontal line
      subroutine blankline()
      implicit none
      write(6,150,advance="no") '-------------------------------------------'
      write(6,150,advance="no") '-------------------------------------------'
      write(6,150,advance="no") '-------------------------------------------'
      write(6,150,advance="no") '-------------------------------------------'
      write(6,150,advance="no") '-------------------------------------------'
      write(6,150) '-------------------------------------------'
 150  format(a)
      return
      end

c     parameterization of Preconditioned MINRES method
      subroutine method(filenameK,filenameA,filenameB,band,rpar,rtol)
      implicit none
      character*60 filenameK,filenameA,filenameB
      integer band
      double precision rpar,rtol
      integer LARGENZ
c     LARGENZ : The maximum number of nonzero elements
c               of the big matrix.
c               The number of nonzero elements of the
c               big matrix is: nzA + 2*nzB
c               The biggest matrices in our tests have:
c               MAXNZA = 70450
c               MAXNZB = 31746
c               So we assign it the value:
c               LARGENZ = MAXNZA + 2*MAXNZB
c                       = 70450  + 2*31746
c                       = 133942
      parameter (LARGENZ = 133942)
      double precision K(LARGENZ)
      integer IK(LARGENZ),JK(LARGENZ)
      integer GLOBALNZ,LARGEDIM,i
      double precision ra,rb
      double precision norm1,norm2,maxnorm
c     functions momnorm and compnorm compute 2 norms
      double precision momnorm,compnorm
c-------------------------------------------------------------
c     MA27 variables
c*************************************************************
```

```
c     MA27ID variables
c
      integer ICNTL(30)
      double precision CNTL(5)
c
c     MA27AD variables
c
      integer NDIM,NDIMPAR
      integer NZ,NZPAR
      integer LM,LMPAR
      integer LIW,LIWPAR
c     NZPAR   : The maximum number of non-zero elements of the
c               upper triangle (including diagonal) of the
c               preconditioned matrix.For the specific
c               preconditioner NZPAR = (band + 1) * NDIMPAR is ok
c               ,where band is 0,1,2,...k
c               band = 0 : Takes the diagonal of big matrix and
c                          on the diagonal of the (2,2) zero block
c                          submatrice sets the value rpar
c               band = 1 : Takes the diagonal of big matrix and
c                          the superdiagonal.On the diagonal of
c                          the (2,2) zero block submatrice sets
c                          the value rpar
c               band = 2 : Takes the diagonal and the two super-
c                          diagonals of the big matrix.On the
c                          diagonal of the (2,2) zero block
c                          submatrice sets  the value rpar
c                 etc..
c               If you want to work for band = 0,1,2,...,k
c               initialize NZAPAR as:
c               NZPAR = (k + 1) * NDIMPAR
c               In the following we initialise it with band = 3 :
c                NZPAR = 4 * NDIMPAR = 4 * 12544 = 50176
c               This initialization is ok for values of band = 0,1,2,3
c               In order to work for values of band larger than 3
c               change the value of NZPAR appropriately as shown
c               above.
c
c     LIWPAR  : The length of the array IW.It must be at least:
c               LIWPAR = 2*NZPAR + 3*NDIMPAR + 1
c               It is recommended that this length should be
c               at least 20 % greate than this.
c               We take it approximately 30% greater for safety.
c
c               For NZPAR = 50176,NDIMPAR = 12544  we have:
c                   NZPAR = 2*50176 + 3*12544 + 1 = 137985
c                    0.3 * 137985 = 41395.5
```

```
c                So     137985 + 41396 = 179381
c                We take it NZPAR = 180000
c
c     NDIMPAR : The largest dimension possible.We assign the value
c                NDIMPAR = N + q = 8450 + 4094 = 12544
c
c
      parameter (NZPAR = 50176,LMPAR = 100000000,LIWPAR = 180000,
     $           NDIMPAR = 12544   )
      integer IRN(NZPAR),ICN(NZPAR)
      integer IW(LIWPAR)
c     IKEEP must have length 3*N.We assign it 3*NDIMPAR = 37632
      integer IKEEP(37632)
c     IW1 must have length 2*N.We assign it 2*NDIMPAR = 25088
      integer IW1(25088)
      integer NSTEPS
      integer IFLAG
      integer INFO(20)
      double precision OPS
c     MA27BD variables
c
      double precision M(LMPAR)
      integer MAXFRT

c     syprec variables
c
      integer neg1,neg2

c     MA27CD variables
c
      double precision RHS(NDIMPAR)
c-----------------------------------------------------------------
      logical checkA
      logical precon
      integer nout,itnlim,istop,itn
      double precision shift
      double precision Anorm,Acond,rnorm,ynorm
      double precision   r1(NDIMPAR), r2(NDIMPAR),
     $                   v(NDIMPAR), w (NDIMPAR), w1(NDIMPAR),
     $                   w2(NDIMPAR),x(NDIMPAR), y (NDIMPAR)
      external Aprod,Msolve

c     parameterization of the largest N and q
      integer NPAR,QPAR
      parameter (NPAR = 8450,QPAR = 4094)
      double precision f(NPAR),g(QPAR),u(NPAR),p(QPAR)
```

```
      integer MAXNZA,MAXNZB
      parameter (MAXNZA = 70450,MAXNZB = 31746)
      integer NZA,NZB,N,Q
      double precision A(MAXNZA),B(MAXNZB)
      integer IA(MAXNZA),JA(MAXNZA)
      integer IB(MAXNZB),JB(MAXNZB)

c     common block for A and B
      common /MAT/ A,B
      common /IND/ IA,JA,IB,JB
      common /PAR/ NZA,NZB,N,Q

c     common block for preconditioner
      common /prec/ M
      common /intvec/ IW,IW1,ICNTL,INFO
      common /intvar/ LM,LIW,MAXFRT,NSTEPS

c     Assign values
      LM  = LMPAR
      LIW = LIWPAR

      call READWHOLEMAT(K,IK,JK,NZA,NZB,N,Q,filenameK)
      GLOBALNZ = NZA + 2 * NZB
      LARGEDIM = N + Q
      call CONSTRUCTRHS(IK,K,LARGEDIM,GLOBALNZ,RHS)

c     construction of the vectors f and g
      call holdrhssol(RHS,f,g,N,Q)
c-----------------------------------------------------------------
c     initialize MA27
      call MA27ID(ICNTL,CNTL)
c
c     ask for full printing from MA27 package
c      ICNTL(3) = 2
c
c     set IFLAG to indicate pivot sequence is to be found by MA27AD
      IFLAG = 0

c     set the order of the preconditioner matrix,construct the
c     preconditioner and set the number of nonzero elements of
c     the preconditioner (fill  NDIM,IRN,ICN,M,NZ)
      NDIM = LARGEDIM

      call CONSTRUCTM(IK,JK,K,GLOBALNZ,IRN,ICN,M,band,NZ,N,Q,rpar)
c
c     analyse sparsity pattern
      call MA27AD(NDIM,NZ,IRN,ICN,IW,LIW,IKEEP,IW1,NSTEPS,IFLAG,
```

```
     +               ICNTL,CNTL,INFO,OPS)
c       write(6,*) 'INFO(5)=',INFO(5)
c       write(6,*) 'INFO(6)=',INFO(6)
c
c      factorize matrix
       call MA27BD(NDIM,NZ,IRN,ICN,M,LM,IW,LIW,IKEEP,NSTEPS,MAXFRT,
     *               IW1,ICNTL,CNTL,INFO)
c      modify essential diagonal blocks
       call syprec( NDIM, LM, LIW, M, IW, neg1, neg2)

c      read the matrices A and B
       call READMAT(IA,JA,A,filenameA)
       call READMAT(IB,JB,B,filenameB)

c      set essential parameters for MINRES
       checkA = .true.
       precon = .true.
       shift = 0.0d0
c      Set output unit number.
c      set nout = 6 for screen or
c      set nout = 0 to suppress output
       nout = 0
       itnlim = 2 * LARGEDIM
       call minres( LARGEDIM, RHS, r1, r2, v, w, w1, w2, x, y,
     $              Aprod, Msolve, checkA, precon, shift,
     $              nout , itnlim, rtol,
     $              istop, itn, Anorm, Acond, rnorm, ynorm )
c       do i=1,LARGEDIM
c             write(6,*) 'sol(',i,')=',x(i)
c       enddo
       call holdrhssol(x,u,p,N,Q)
       call errornorms(u,p,norm1,norm2,maxnorm)
       ra = momnorm(IA,JA,A,IB,JB,B,f,u,p)
       rb = compnorm(IB,JB,B,g,u)
       write(6,500,advance="no") rtol,N+Q,ra,rb,
     $dsqrt(ra * ra + rb * rb),itn,norm1,norm2,maxnorm
  500 format(e8.1,1x,i5,6x,e16.9,2x,e16.9,3x,e16.9,5x,i5,5x,e16.9,
     $9x,e16.9,9x,e16.9)
       return
       end


c Computation of the error norm : ||x - x*||,where:
c x  = (u,p)        the computed solution
c x* = (1,1,...,1) the exact solution
c vectors u and p remain unchanged on exit
c norm1   : returns ||x - x*||_1
```

```
c norm2    : returns ||x - x*||_2
c maxnorm : returns ||x - x*||_sup
      subroutine errornorms(u,p,norm1,norm2,maxnorm)
      implicit none
      double precision u(*),p(*)
      double precision norm1,norm2,maxnorm
      integer dimen,i
      integer NZA,NZB,N,Q
      double precision mydasum,mydnrm2,mymaxnorm
      common /PAR/ NZA,NZB,N,Q
      double precision err(N+Q)
      dimen = N + Q
      do i=1,N
         err(i) = u(i) - 1.0d0
      enddo
      do i=N+1,dimen
         err(i) = p(i-N) - 1.0d0
      enddo
      norm1 = mydasum(dimen,err)
      norm2 =  mydnrm2(dimen,err)
      maxnorm = mymaxnorm(dimen,err)
      return
      end

c     Computes l^1 norm of x : ||x||_1
      double precision function mydasum(dimen,x)
      implicit none
      integer dimen
      double precision x(*)
      double precision dasum
      mydasum = dasum(dimen,x,1)
      return
      end

c     finds the index of element of x,having max. absolute value.
      integer function myidamax(dimen,x)
      implicit none
      integer dimen
      double precision x(*)
      integer idamax
      myidamax = idamax(dimen,x,1)
      return
      end

c     Computes maximum norm of x: ||x||_sup
      double precision function mymaxnorm(dimen,x)
      implicit none
```

```
      integer dimen
      double precision x(*)
      integer pos
      integer myidamax
      pos = myidamax(dimen,x)
      mymaxnorm = dabs(x(pos))
      return
      end

c  Computation of the norm: ||f - Au - Bp||_2, where u and p are the
c  computed velocity and pressure solutions, respectively.
c  This quantity is a measure of how good the solution satisfies the
c  discretized momentum equation: Au + Bp = f
      double precision function momnorm(IA,JA,A,IB,JB,B,f,u,p)
      implicit none
      integer IA(*),JA(*),IB(*),JB(*)
      double precision A(*),B(*),f(*),u(*),p(*)
      integer NZA,NZB,N,Q
      common /PAR/ NZA,NZB,N,Q
      double precision Au(N),Bp(N)
      double precision scalar
      double precision mydnrm2
c     Au = A * u
      call MATVEC(IA,JA,A,NZA,u,Au,N)
c     Bp = B * p
      call MATVEC(IB,JB,B,NZB,p,Bp,N)
c     Au = f - Au
      scalar = -1.0d0
      call moddaxpy(N,scalar,f,Au)
c     Bp = Au - Bp = (f - Au) - Bp
      scalar = -1.0d0
      call moddaxpy(N,scalar,Au,Bp)
c     return ||Bp||=||f - Au - Bp||
      momnorm =  mydnrm2(N,Bp)
      return
      end

c  Computation of the norm: ||g - B**T u||_2, where u is the computed
c  velocity solution.
c  This quantity is a measure of how good the solution satisfies the
c  discretized incompressibility condition: B**T u = g
      double precision function compnorm(IB,JB,B,g,u)
      implicit none
      integer IB(*),JB(*)
      double precision B(*),g(*),u(*)
      integer NZA,NZB,N,Q
      common /PAR/ NZA,NZB,N,Q
```

```
      double precision Btu(Q)
      double precision scalar
      double precision mydnrm2
c     Btu = B**T * u
      call TRANSMATVEC(IB,JB,B,NZB,u,Btu,Q)
c     Btu = g - Btu
      scalar = -1.0d0
      call moddaxpy(Q,scalar,g,Btu)
c     return ||Btu||=||g - B**T * u||
      compnorm = mydnrm2(Q,Btu)
      return
      end

c     Computes l^2 norm of x : ||x||_2
      double precision function mydnrm2(dimen,x)
      implicit none
      integer dimen
      double precision x(*)
      double precision dnrm2
      mydnrm2 = dnrm2(dimen,x,1)
      return
      end


c     y = x + scalar * y,x remains unchanged on exit
      subroutine moddaxpy(dimen,scalar,x,y)
      implicit none
      integer dimen
      double precision scalar
      double precision x(*),y(*)
      double precision alpha
c     y = scalar * y
      call mydscal(dimen,scalar,y)
c     y = x + y
      alpha = 1.0d0
      call mydaxpy(dimen,alpha,x,y)
      return
      end

c     On exit vector = scalar * vector
      subroutine mydscal(dimen,scalar,vector)
      implicit none
      integer dimen
      double precision scalar
      double precision vector(*)
      call dscal(dimen,scalar,vector,1)
      return
```

```
      end

c     y = scalar * x + y, x remains unchanged on exit
      subroutine mydaxpy(dimen, scalar, x, y)
      implicit none
      integer dimen
      double precision scalar
      double precision x(*), y(*)
      call daxpy(dimen, scalar, x, 1, y, 1)
      return
      end


c  recover f and g or u and p from RHS or x
      subroutine holdrhssol(RHS, first, second, N, q)
      implicit none
      double precision RHS(*), first(*), second(*)
      integer N, q
      integer ndim, i
      ndim = N + q
      do i=1,N
            first(i) = RHS(i)
      enddo
      do i=N+1,ndim
            second(i-N) = RHS(i)
      enddo
      return
      end

c  Msolve solves M*y = x for some symmetric positive-definite
c  matrix M.
      subroutine Msolve( NDIM, x, y )
      implicit none
      integer NDIM
      double precision   x(NDIM), y(NDIM)
      double precision temprhs(NDIM)
c     declaration of common block variables
      integer LMPAR
      integer LIWPAR
      parameter (LMPAR = 100000000, LIWPAR = 180000 )
      integer LM, LIW, MAXFRT, NSTEPS
      double precision M(LMPAR)
      integer IW(LIWPAR)
c     IW1 must have length 2*N. Must declared exactly as in main program.
      integer IW1(25088)
      integer ICNTL(30)
      integer INFO(20)
```

```
c      common block for preconditioner
       common /prec/ M
       common /intvec/ IW,IW1,ICNTL,INFO
       common /intvar/ LM,LIW,MAXFRT,NSTEPS
c
c      local vector
       double precision W(MAXFRT)
c      temprhs = x
       call dcopy(NDIM,x,1,temprhs,1)



c      solve the equations
       call MA27CD(NDIM,M,LM,IW,LIW,W,MAXFRT,temprhs,IW1,NSTEPS,ICNTL,
     +             INFO)

c      y = temprhs
       call dcopy(NDIM,temprhs,1,y,1)
       return
       end

c reading of the nonzero elements of the whole matrix,
c stored in coordinate format,in global enumeration
       subroutine READWHOLEMAT(K,IK,JK,NZA,NZB,N,q,filename)
       implicit none
       integer IK(*),JK(*)
       double precision K(*)
       integer NZA,NZB,N,q
       character*60 filename
       integer p
       integer m,i,j
       double precision x
       open(unit = 10,file = filename,status ='old')
       read(10,*) NZA,NZB,N,q
       p = NZA + 2 * NZB
       do m=1,p
          read(10,*) i,j,x
          IK(m) = i
          JK(m) = j
          K(m) = x
       enddo
       close(10)
       return
       end

c  The right hand side vector of the linear system is constructed
c  such that its solution is the vector (1,1,...,1)
       subroutine CONSTRUCTRHS(IK,K,LARGEDIM,GLOBALNZ,RHS)
```

```
      implicit none
      integer IK(*),LARGEDIM,GLOBALNZ
      double precision K(*),RHS(*)
      double precision xsum
      integer i,m
      do i=1,LARGEDIM
                  xsum = 0.0d0
                  do m=1,GLOBALNZ
                        if(IK(m).eq.i) then
                              xsum = xsum + K(m)
                        endif
                  enddo
                  RHS(i) = xsum
      enddo
      return
      end

c     Construction of the preconditioner
      subroutine CONSTRUCTM(IK,JK,K,GLOBALNZ,IM,JM,M,band,nz,N,q,r)
      implicit none
      integer IK(*),JK(*),IM(*),JM(*),GLOBALNZ
      double precision K(*),M(*)
      integer band,nz,N,q
      double precision r
      integer el,num,i,j,p
      num = 1
      do el=1,GLOBALNZ
                  i = IK(el)
                  j = JK(el)
                  do p=0,band
                        if(j .eq. (i + p)) then
                              IM(num) = i
                              JM(num) = j
                              M(num) = K(el)
                              num = num + 1
                              goto 300
                        endif
                  enddo
 300        continue
      enddo
      num = num - 1
      do el=1,q
           IM(num + el) = N + el
           JM(num + el) = N + el
            M(num + el) = r
      enddo
      nz = num + q
```

```
      return
      end

c  reading of the nonzero elements of matrix A or B,
c  stored in coordinate format, in local enumeration
      subroutine READMAT(IMAT,JMAT,MAT,filename)
      implicit none
      integer IMAT(*),JMAT(*)
      double precision MAT(*)
      character*60 filename
      integer N,nz,k,i,j
      double precision x
      open(unit = 20,file = filename,status ='old')
      read(20,*) N,nz
      do k=1,nz
          read(20,*) i,j,x
          IMAT(k) = i
          JMAT(k) = j
          MAT(k) = x
      enddo
      close(20)
      return
      end


c  Computes matrix-vector product: y = mat x,
c  mat is represented in coordinate sparse format.
c  dimy is the dimension of the output vector y,
c  x remains unchanged on exit
      subroutine MATVEC(IMAT,JMAT,MAT,NZ,X,Y,DIMY)
      implicit none
      integer IMAT(*),JMAT(*),NZ,DIMY
      double precision MAT(*),X(*),Y(*)
      integer k
      do k=1,DIMY
           Y(k) = 0.0d0
      enddo
      do k=1,NZ
           Y(IMAT(k)) = Y(IMAT(k)) + MAT(k) * X(JMAT(k))
      enddo
      return
      end


c  Computes the transpose matrix-vector product: y = mat^T x,
c  mat is represented in coordinate sparse format.
c  dimy is the dimension of the output vector y,
```

```
c  x remains unchanged on exit
      subroutine TRANSMATVEC(IMAT,JMAT,MAT,NZ,X,Y,DIMY)
      implicit none
      integer IMAT(*),JMAT(*),NZ,DIMY
      double precision MAT(*),X(*),Y(*)
      integer k
      do k=1,DIMY
            Y(k) = 0.0d0
      enddo
      do k=1,NZ
            Y(JMAT(k)) = Y(JMAT(k)) + MAT(k) * X(IMAT(k))
      enddo
      return
      end

c   Aprod computes y = A*x for some matrix A.
      subroutine Aprod ( ndim, x, y )
      implicit none
      integer ndim
      double precision    x(ndim), y(ndim)
      double precision temp(ndim)
      double precision da
      integer i
      integer MAXNZA,MAXNZB
      parameter (MAXNZA = 70450,MAXNZB = 31746)
      integer NZA,NZB,N,Q
      double precision A(MAXNZA),B(MAXNZB)
      integer IA(MAXNZA),JA(MAXNZA)
      integer IB(MAXNZB),JB(MAXNZB)
      common /MAT/ A,B
      common /IND/ IA,JA,IB,JB
      common /PAR/ NZA,NZB,N,Q
      double precision u(N),p(Q),Au(N),Bp(N)
c     temp = x
      call dcopy(ndim,x,1,temp,1)
      do i=1,N
            u(i) = temp(i)
      enddo
      do i=N+1,ndim
            p(i-N) = temp(i)
      enddo
c     Au = A * u
      call matvec(IA,JA,A,NZA,u,Au,N)
c     Bp = B * p
      call matvec(IB,JB,B,NZB,p,Bp,N)
c     Bp = da * Au + Bp
      da = 1.0d0
```

```
      call daxpy(N,da,Au,1,Bp,1)
c     p = B^T * u
      call transmatvec(IB,JB,B,NZB,u,p,Q)
      do i=1,N
            y(i) = Bp(i)
      enddo
      do i=1,Q
            y(N+i) = p(i)
      enddo
      return
      end
```

## Α΄.7 Κώδικες για τη μέθοδο MINRES

Στη συνέχεια παραθέτουμε τους κώδικες που αναπτύχθηκαν για τη μέθοδο MINRES.

```
c234567
c * Author      : Manos Psycharis
c * Purpose     : Master Thesis
c * Date        : 03/03/2010
c *
c * Compile      g95 -o file MINRES_no_prec.f minres.f minresblas.f
c               -lblas
      program main
      implicit none
      character*30 filename
      character*60 filenameK,filenameA,filenameB
      character*60 stringK,stringA,stringB
      integer i,k,nummat,tolvals
      double precision TOL
      real timebegin,timeend
      filename = 'input_data.dat'
      open(unit = 50,file = filename,status ='old')
c     Set the number of matrices
      nummat = 9
c     Set the number of different values of TOL
      tolvals = 5
      write(6,100) 'Method_:_MINRES'
      call blankline()
      write(6,200) 'TOL','DOF','||f-Au-Bp||','||g-B^Tu||','||r||'
     $,'#iter','||x-x*||_1','||x-x*||_2','||x-x*||_sup'
     $,'CPU_time(sec)','filenameA'
      call blankline()
      do i=1,nummat
            read(50,*) filenameA
            read(50,*) filenameB
            read(50,*) filenameK
```

```
            stringA = trim(adjustl(filenameA))
            stringB = trim(adjustl(filenameB))
            stringK = trim(adjustl(filenameK))
            do k=1,tolvals
                read(50,*) TOL
                call cpu_time(timebegin)
c                call method(filenameK,filenameA,filenameB,TOL)
                call method(stringK,stringA,stringB,TOL)
                call cpu_time(timeend)
                write(6,300,advance="no") timeend - timebegin
                write(6,400) trim(filenameA)
            enddo
            call blankline()
        enddo
        close(50)
        write(6,*)
        write(6,*) 'MINRES runs ended successfully!'
 100    format(80x,a)
 200    format(1x,a,7x,a,9x,a,7x,a,12x,a,11x,a,7x,a,
     $15x,a,15x,a,9x,a,9x,a)
 300    format(7x,f10.7)
 400    format(12x,a)
        stop
        end


c       horizontal line
        subroutine blankline()
        implicit none
        write(6,150,advance="no") '-------------------------------------------'
        write(6,150,advance="no") '-------------------------------------------'
        write(6,150,advance="no") '-------------------------------------------'
        write(6,150,advance="no") '-------------------------------------------'
        write(6,150,advance="no") '-------------------------------------------'
        write(6,150) '-------------------------------------------'
 150    format(a)
        return
        end


c       parameterization of MINRES method
        subroutine method(filenameK,filenameA,filenameB,rtol)
        implicit none
        character*60 filenameK,filenameA,filenameB
        double precision rtol
        integer LARGENZ,NDIMPAR
c       LARGENZ : The maximum number of nonzero elements
```

```
c                of the big matrix.
c                The number of nonzero elements of the
c                big matrix is: nzA + 2*nzB
c                The biggest matrices in our tests have:
c                MAXNZA = 70450
c                MAXNZB = 31746
c                So we assign it the value:
c                LARGENZ = MAXNZA + 2*MAXNZB
c                        = 70450  + 2*31746
c                        = 133942
c     NDIMPAR : The largest dimension possible.We assign the value
c                NDIMPAR = N + q = 8450 + 4094 = 12544
      parameter (LARGENZ = 133942,NDIMPAR = 12544)
      double precision K(LARGENZ),RHS(NDIMPAR)
      integer IK(LARGENZ),JK(LARGENZ)
      integer GLOBALNZ,LARGEDIM, i
      double precision ra,rb
      double precision norm1,norm2,maxnorm
c     functions momnorm and compnorm compute 2 norms
      double precision momnorm,compnorm
      logical checkA
      logical precon
      integer nout,itnlim,istop,itn
      double precision shift
      double precision Anorm,Acond,rnorm,ynorm
      double precision   r1(NDIMPAR), r2(NDIMPAR),
     $                   v(NDIMPAR), w (NDIMPAR), w1(NDIMPAR),
     $                   w2(NDIMPAR),x(NDIMPAR), y (NDIMPAR)
      external Aprod

c     parameterization of the largest N and q
      integer NPAR,QPAR
      parameter (NPAR = 8450,QPAR = 4094)
      double precision f(NPAR),g(QPAR),u(NPAR),p(QPAR)


      integer MAXNZA,MAXNZB
      parameter (MAXNZA = 70450,MAXNZB = 31746)
      integer NZA,NZB,N,Q
      double precision A(MAXNZA),B(MAXNZB)
      integer IA(MAXNZA),JA(MAXNZA)
      integer IB(MAXNZB),JB(MAXNZB)
c     common block for A and B
      common /MAT/ A,B
      common /IND/ IA,JA,IB,JB
      common /PAR/ NZA,NZB,N,Q
```

```
      call READWHOLEMAT(K,IK,JK,NZA,NZB,N,Q,filenameK)
      GLOBALNZ = NZA + 2 * NZB
      LARGEDIM = N + Q
c     construct the right hand side
      call CONSTRUCTRHS(IK,K,LARGEDIM,GLOBALNZ,RHS)
c     construction of the vectors f and g
      call holdrhssol(RHS,f,g,N,Q)
c     read the matrices A and B
      call READMAT(IA,JA,A,filenameA)
      call READMAT(IB,JB,B,filenameB)
      checkA = .true.
      precon = .false.
      shift = 0.0d0
c     Set output unit number.
c     set nout = 6 for screen or
c     set nout = 0 to suppress output
      nout = 0
      itnlim = 2 * LARGEDIM
      call minres( LARGEDIM, RHS, r1, r2, v, w, w1, w2, x, y,
     $             Aprod, Aprod, checkA, precon, shift,
     $             nout , itnlim, rtol,
     $             istop, itn, Anorm, Acond, rnorm, ynorm )
c      do i=1,LARGEDIM
c            write(6,*) 'sol(',i,')=',x(i)
c      enddo
      call holdrhssol(x,u,p,N,Q)
      call errornorms(u,p,norm1,norm2,maxnorm)
      ra = momnorm(IA,JA,A,IB,JB,B,f,u,p)
      rb = compnorm(IB,JB,B,g,u)
      write(6,500,advance="no") rtol,N+Q,ra,rb,
     $dsqrt(ra * ra + rb * rb),itn,norm1,norm2,maxnorm
  500 format(e8.1,1x,i5,6x,e16.9,2x,e16.9,3x,e16.9,5x,i5,5x,e16.9,
     $9x,e16.9,9x,e16.9)
      return
      end



c Computation of the error norm : ||x - x*||, where:
c x  = (u,p)        the computed solution
c x* = (1,1,...,1)  the exact solution
c vectors u and p remain unchanged on exit
c norm1   : returns ||x - x*||_1
c norm2   : returns ||x - x*||_2
c maxnorm : returns ||x - x*||_sup
      subroutine errornorms(u,p,norm1,norm2,maxnorm)
      implicit none
```

```
      double precision u(*),p(*)
      double precision norm1,norm2,maxnorm
      integer dimen,i
      integer NZA,NZB,N,Q
      double precision mydasum,mydnrm2,mymaxnorm
      common /PAR/ NZA,NZB,N,Q
      double precision err(N+Q)
      dimen = N + Q
      do i=1,N
         err(i) = u(i) - 1.0d0
      enddo
      do i=N+1,dimen
         err(i) = p(i-N) - 1.0d0
      enddo
      norm1 = mydasum(dimen,err)
      norm2 =  mydnrm2(dimen,err)
      maxnorm = mymaxnorm(dimen,err)
      return
      end

c     Computes l^1 norm of x : ||x||_1
      double precision function mydasum(dimen,x)
      implicit none
      integer dimen
      double precision x(*)
      double precision dasum
      mydasum = dasum(dimen,x,1)
      return
      end

c     finds the index of element of x,having max. absolute value.
      integer function myidamax(dimen,x)
      implicit none
      integer dimen
      double precision x(*)
      integer idamax
      myidamax = idamax(dimen,x,1)
      return
      end

c     Computes maximum norm of x: ||x||_sup
      double precision function mymaxnorm(dimen,x)
      implicit none
      integer dimen
      double precision x(*)
      integer pos
      integer myidamax
```

```
      pos = myidamax(dimen,x)
      mymaxnorm = dabs(x(pos))
      return
      end


c  Computation of the norm: ||f - Au - Bp||_2, where u and p are the
c  computed velocity and pressure solutions, respectively.
c  This quantity is a measure of how good the solution satisfies the
c  discretized momentum equation: Au + Bp = f
      double precision function momnorm(IA,JA,A,IB,JB,B,f,u,p)
      implicit none
      integer IA(*),JA(*),IB(*),JB(*)
      double precision A(*),B(*),f(*),u(*),p(*)
      integer NZA,NZB,N,Q
      common /PAR/ NZA,NZB,N,Q
      double precision Au(N),Bp(N)
      double precision scalar
      double precision mydnrm2
c     Au = A * u
      call MATVEC(IA,JA,A,NZA,u,Au,N)
c     Bp = B * p
      call MATVEC(IB,JB,B,NZB,p,Bp,N)
c     Au = f - Au
      scalar = -1.0d0
      call moddaxpy(N,scalar,f,Au)
c     Bp = Au - Bp = (f - Au) - Bp
      scalar = -1.0d0
      call moddaxpy(N,scalar,Au,Bp)
c     return ||Bp||=||f - Au - Bp||
      momnorm =  mydnrm2(N,Bp)
      return
      end

c  Computation of the norm: ||g - B**T u||_2, where u is the computed
c  velocity solution.
c  This quantity is a measure of how good the solution satisfies the
c  discretized incompressibility condition: B**T u = g
      double precision function compnorm(IB,JB,B,g,u)
      implicit none
      integer IB(*),JB(*)
      double precision B(*),g(*),u(*)
      integer NZA,NZB,N,Q
      common /PAR/ NZA,NZB,N,Q
      double precision Btu(Q)
      double precision scalar
      double precision mydnrm2
```

```
c       Btu = B**T * u
        call TRANSMATVEC(IB,JB,B,NZB,u,Btu,Q)
c       Btu = g - Btu
        scalar = -1.0d0
        call moddaxpy(Q,scalar,g,Btu)
c       return ||Btu||=||g - B**T * u||
        compnorm = mydnrm2(Q,Btu)
        return
        end

c       Computes l^2 norm of x : ||x||_2
        double precision function mydnrm2(dimen,x)
        implicit none
        integer dimen
        double precision x(*)
        double precision dnrm2
        mydnrm2 = dnrm2(dimen,x,1)
        return
        end

c       y = x + scalar * y,x remains unchanged on exit
        subroutine moddaxpy(dimen,scalar,x,y)
        implicit none
        integer dimen
        double precision scalar
        double precision x(*),y(*)
        double precision alpha
c       y = scalar * y
        call mydscal(dimen,scalar,y)
c       y = x + y
        alpha = 1.0d0
        call mydaxpy(dimen,alpha,x,y)
        return
        end

c       On exit vector = scalar * vector
        subroutine mydscal(dimen,scalar,vector)
        implicit none
        integer dimen
        double precision scalar
        double precision vector(*)
        call dscal(dimen,scalar,vector,1)
        return
        end

c       y = scalar * x + y,x remains unchanged on exit
        subroutine mydaxpy(dimen,scalar,x,y)
```

```
      implicit none
      integer dimen
      double precision scalar
      double precision x(*),y(*)
      call daxpy(dimen,scalar,x,1,y,1)
      return
      end


c recover f and g or u and p from RHS or x
      subroutine holdrhssol(RHS,first,second,N,q)
      implicit none
      double precision RHS(*),first(*),second(*)
      integer N,q
      integer ndim,i
      ndim = N + q
      do i=1,N
            first(i) = RHS(i)
      enddo
      do i=N+1,ndim
            second(i-N) = RHS(i)
      enddo
      return
      end

c reading of the nonzero elements of the whole matrix,
c stored in coordinate format,in global enumeration
      subroutine READWHOLEMAT(K,IK,JK,NZA,NZB,N,q,filename)
      implicit none
      integer IK(*),JK(*)
      double precision K(*)
      integer NZA,NZB,N,q
      character*60 filename
      integer p
      integer m,i,j
      double precision x
      open(unit = 10,file = filename,status ='old')
      read(10,*) NZA,NZB,N,q
      p = NZA + 2 * NZB
      do m=1,p
         read(10,*) i,j,x
         IK(m) = i
         JK(m) = j
         K(m) = x
      enddo
      close(10)
      return
```

```
      end

c  The right hand side vector of the linear system is constructed
c  such that its solution is the vector (1,1,...,1)
      subroutine CONSTRUCTRHS(IK,K,LARGEDIM,GLOBALNZ,RHS)
      implicit none
      integer IK(*),LARGEDIM,GLOBALNZ
      double precision K(*),RHS(*)
      double precision xsum
      integer i,m
      do i=1,LARGEDIM
                  xsum = 0.0d0
                  do m=1,GLOBALNZ
                        if(IK(m).eq.i) then
                              xsum = xsum + K(m)
                        endif
                  enddo
                  RHS(i) = xsum
      enddo
      return
      end

c  reading of the nonzero elements of matrix A or B,
c  stored in coordinate format, in local enumeration
      subroutine READMAT(IMAT,JMAT,MAT,filename)
      implicit none
      integer IMAT(*),JMAT(*)
      double precision MAT(*)
      character*60 filename
      integer N,nz,k,i,j
      double precision x
      open(unit = 20,file = filename,status ='old')
      read(20,*) N,nz
      do k=1,nz
          read(20,*) i,j,x
          IMAT(k) = i
          JMAT(k) = j
          MAT(k) = x
      enddo
      close(20)
      return
      end

c  Computes matrix-vector product: y = mat x,
c  mat is represented in coordinate sparse format.
c  dimy is the dimension of the output vector y,
c  x remains unchanged on exit
```

```
      subroutine MATVEC(IMAT,JMAT,MAT,NZ,X,Y,DIMY)
      implicit none
      integer IMAT(*),JMAT(*),NZ,DIMY
      double precision MAT(*),X(*),Y(*)
      integer k
      do k=1,DIMY
            Y(k) = 0.0d0
      enddo
      do k=1,NZ
            Y(IMAT(k)) = Y(IMAT(k)) + MAT(k) * X(JMAT(k))
      enddo
      return
      end

c  Computes the transpose matrix-vector product: y = mat^T x,
c  mat is represented in coordinate sparse format.
c  dimy is the dimension of the output vector y,
c  x remains unchanged on exit
      subroutine TRANSMATVEC(IMAT,JMAT,MAT,NZ,X,Y,DIMY)
      implicit none
      integer IMAT(*),JMAT(*),NZ,DIMY
      double precision MAT(*),X(*),Y(*)
      integer k
      do k=1,DIMY
            Y(k) = 0.0d0
      enddo
      do k=1,NZ
            Y(JMAT(k)) = Y(JMAT(k)) + MAT(k) * X(IMAT(k))
      enddo
      return
      end

c   Aprod computes y = A*x for some matrix A.
      subroutine Aprod ( ndim, x, y )
      implicit none
      integer ndim
      double precision    x(ndim), y(ndim)
      double precision temp(ndim)
      double precision da
      integer i
      integer MAXNZA,MAXNZB
      parameter (MAXNZA = 70450,MAXNZB = 31746)
      integer NZA,NZB,N,Q
      double precision A(MAXNZA),B(MAXNZB)
      integer IA(MAXNZA),JA(MAXNZA)
      integer IB(MAXNZB),JB(MAXNZB)
      common /MAT/ A,B
```

```
      common /IND/ IA,JA,IB,JB
      common /PAR/ NZA,NZB,N,Q
      double precision u(N),p(Q),Au(N),Bp(N)
c     temp = x
      call dcopy(ndim,x,1,temp,1)
      do i=1,N
            u(i) = temp(i)
      enddo
      do i=N+1,ndim
            p(i-N) = temp(i)
      enddo
c     Au = A * u
      call matvec(IA,JA,A,NZA,u,Au,N)
c     Bp = B * p
      call matvec(IB,JB,B,NZB,p,Bp,N)
c     Bp = da * Au + Bp
      da = 1.0d0
      call daxpy(N,da,Au,1,Bp,1)
c     p = B^T * u
      call transmatvec(IB,JB,B,NZB,u,p,Q)
      do i=1,N
            y(i) = Bp(i)
      enddo
      do i=1,Q
            y(N+i) = p(i)
      enddo
      return
      end
```

# Παράρτημα Β′

# Βοηθητικοί κώδικες

Στη συνέχεια παραθέτουμε κάποιους από τους κώδικες που χρησιμοποιήθηκαν βοηθητικά, στους παραπάνω κώδικες που αναπτύχθηκαν στη μελέτη.

## Β′.1 Μια υλοποίηση της Preconditioned MINRES / MINRES

Στη συνέχεια παραθέτουμε μια υλοποίηση της Preconditioned MINRES / MINRES από το Systems Optimization Laboratory του Πανεπιστημίου του Stanford και τον Michael Saunders (βλέπε [33, 32]).

### Β′.1.1 Η υπορουτίνα MINRES.

```
      subroutine MINRES( n, b, r1, r2, v, w, w1, w2, x, y,
     $                   Aprod, Msolve, checkA, precon, shift,
     $                   nout , itnlim, rtol,
     $                   istop, itn, Anorm, Acond, rnorm, ynorm )

      implicit           none
      external           Aprod, Msolve
      integer            n, nout, itnlim, istop, itn
      logical            checkA, precon
      double precision   shift, rtol, Anorm, Acond, rnorm, ynorm,
     $                   b(n), r1(n), r2(n),
     $                   v(n), w(n), w1(n), w2(n), x(n), y(n)
!     ------------------------------------------------------------------
!
!     MINRES  is designed to solve the system of linear equations
!
!                Ax = b
!
!     or the least-squares problem
!
!          min || Ax - b ||_2,
```

```
!
!     where A is an n by n symmetric matrix and b is a given vector.
!     The matrix A may be indefinite and/or singular.
!
!     1. If A is known to be positive definite, the Conjugate Gradient
!        Method might be preferred, since it requires the same number
!        of iterations as MINRES but less work per iteration.
!
!     2. If A is indefinite but Ax = b is known to have a solution
!        (e.g. if A is nonsingular), SYMMLQ might be preferred,
!        since it requires the same number of iterations as MINRES
!        but slightly less work per iteration.
!
!     The matrix A is intended to be large and sparse.  It is accessed
!     by means of a subroutine call of the form
!     SYMMLQ development:
!
!                call Aprod ( n, x, y )
!
!     which must return the product y = Ax for any given vector x.
!
!
!     More generally, MINRES is designed to solve the system
!
!                (A - shift*I) x = b
!     or
!          min || (A - shift*I) x - b ||_2,
!
!     where shift is a specified scalar value.  Again, the matrix
!     (A - shift*I) may be indefinite and/or singular.
!     The work per iteration is very slightly less if  shift = 0.
!
!     Note: If  shift  is an approximate eigenvalue of  A
!     and  b  is an approximate eigenvector,  x  might prove to be
!     a better approximate eigenvector, as in the methods of
!     inverse iteration and/or Rayleigh-quotient iteration.
!     However, we're not yet sure on that -- it may be better
!     to use SYMMLQ.
!
!     A further option is that of preconditioning, which may reduce
!     the number of iterations required.  If M = C C' is a positive
!     definite matrix that is known to approximate  (A - shift*I)
!     in some sense, and if systems of the form  My = x  can be
!     solved efficiently, the parameters precon and Msolve may be
!     used (see below).  When  precon = .true., MINRES will
!     implicitly solve the system of equations
!
```

```
!              P (A - shift*I) P' xbar  =  P b,
!
!     i.e.                   Abar xbar  =  bbar
!     where                        P    =  C**(-1),
!                                Abar   =  P (A - shift*I) P',
!                                bbar   =  P b,
!
!     and return the solution      x    =  P' xbar.
!     The associated residual is rbar   =  bbar - Abar xbar
!                                       =  P (b - (A - shift*I)x)
!                                       =  P r.
!
!     In the discussion below, eps refers to the machine precision.
!     eps is computed by MINRES.  A typical value is eps = 2.22d-16
!     for IEEE double-precision arithmetic.
!
!     Parameters
!     ----------
!
!     n         input      The dimension of the matrix A.
!
!     b(n)      input      The rhs vector b.
!
!     r1(n)     workspace
!     r2(n)     workspace
!     v(n)      workspace
!     w(n)      workspace
!     w1(n)     workspace
!     w2(n)     workspace
!
!     x(n)      output     Returns the computed solution  x.
!
!     y(n)      workspace
!
!     Aprod     external   A subroutine defining the matrix A.
!                          For a given vector x, the statement
!
!                                  call Aprod ( n, x, y )
!
!                          must return the product y = Ax
!                          without altering the vector x.
!
!     Msolve    external   An optional subroutine defining a
!                          preconditioning matrix M, which should
!                          approximate (A - shift*I) in some sense.
!                          M must be positive definite.
!                          For a given vector x, the statement
```

```
!
!                                   call Msolve( n, x, y )
!
!                               must solve the linear system My = x
!                               without altering the vector x.
!
!                               In general, M should be chosen so that Abar has
!                               clustered eigenvalues.  For example,
!                               if A is positive definite, Abar would ideally
!                               be close to a multiple of I.
!                               If A or A - shift*I is indefinite, Abar might
!                               be close to a multiple of diag( I  -I ).
!
!                               NOTE.  The program calling MINRES must declare
!                               Aprod and Msolve to be external.
!
!     checkA   input            If checkA = .true., an extra call of Aprod will
!                               be used to check if A is symmetric.  Also,
!                               if precon = .true., an extra call of Msolve
!                               will be used to check if M is symmetric.
!
!     precon   input            If precon = .true., preconditioning will
!                               be invoked.  Otherwise, subroutine Msolve
!                               will not be referenced; in this case the
!                               actual parameter corresponding to Msolve may
!                               be the same as that corresponding to Aprod.
!
!     shift    input            Should be zero if the system Ax = b is to be
!                               solved.  Otherwise, it could be an
!                               approximation to an eigenvalue of A, such as
!                               the Rayleigh quotient b'Ab / (b'b)
!                               corresponding to the vector b.
!                               If b is sufficiently like an eigenvector
!                               corresponding to an eigenvalue near shift,
!                               then the computed x may have very large
!                               components.  When normalized, x may be
!                               closer to an eigenvector than b.
!
!     nout     input            A file number.
!                               If nout .gt. 0, a summary of the iterations
!                               will be printed on unit nout.
!
!     itnlim   input            An upper limit on the number of iterations.
!
!     rtol     input            A user-specified tolerance.  MINRES terminates
!                               if it appears that norm(rbar) is smaller than
!                                     rtol * norm(Abar) * norm(xbar),
```

```
!                          where rbar is the transformed residual vector,
!                                rbar = bbar - Abar xbar.
!
!                          If shift = 0 and precon = .false., MINRES
!                          terminates if norm(b - A*x) is smaller than
!                                rtol * norm(A) * norm(x).
!
!     istop    output      An integer giving the reason for termination...
!
!                -1        beta2 = 0 in the Lanczos iteration; i.e. the
!                          second Lanczos vector is zero.  This means the
!                          rhs is very special.
!                          If there is no preconditioner, b is an
!                          eigenvector of A.
!                          Otherwise (if precon is true), let My = b.
!                          If shift is zero, y is a solution of the
!                          generalized eigenvalue problem Ay = lambda My,
!                          with lambda = alpha1 from the Lanczos vectors.
!
!                          In general, (A - shift*I)x = b
!                          has the solution          x = (1/alpha1) y
!                          where My = b.
!
!                 0        b = 0, so the exact solution is x = 0.
!                          No iterations were performed.
!
!                 1        Norm(rbar) appears to be less than
!                          the value  rtol * norm(Abar) * norm(xbar).
!                          The solution in  x  should be acceptable.
!
!                 2        Norm(rbar) appears to be less than
!                          the value  eps * norm(Abar) * norm(xbar).
!                          This means that the residual is as small as
!                          seems reasonable on this machine.
!
!                 3        Norm(Abar) * norm(xbar) exceeds norm(b)/eps,
!                          which should indicate that x has essentially
!                          converged to an eigenvector of A
!                          corresponding to the eigenvalue shift.
!
!                 4        Acond (see below) has exceeded 0.1/eps, so
!                          the matrix Abar must be very ill-conditioned.
!                          x may not contain an acceptable solution.
!
!                 5        The iteration limit was reached before any of
!                          the previous criteria were satisfied.
!
```

```
!                6          The matrix defined by Aprod does not appear
!                           to be symmetric.
!                           For certain vectors y = Av and r = Ay, the
!                           products y'y and r'v differ significantly.
!
!                7          The matrix defined by Msolve does not appear
!                           to be symmetric.
!                           For vectors satisfying My = v and Mr = y, the
!                           products y'y and r'v differ significantly.
!
!                8          An inner product of the form  x' M**(-1) x
!                           was not positive, so the preconditioning matrix
!                           M does not appear to be positive definite.
!
!                           If istop .ge. 5, the final x may not be an
!                           acceptable solution.
!
!     itn        output     The number of iterations performed.
!
!     Anorm      output     An estimate of the norm of the matrix operator
!                           Abar = P (A - shift*I) P',    where P = C**(-1).
!
!     Acond      output     An estimate of the condition of Abar above.
!                           This will usually be a substantial
!                           under-estimate of the true condition.
!
!     rnorm      output     An estimate of the norm of the final
!                           transformed residual vector,
!                              P (b  -  (A - shift*I) x).
!
!     ynorm      output     An estimate of the norm of xbar.
!                           This is sqrt( x'Mx ).  If precon is false,
!                           ynorm is an estimate of norm(x).
!     ------------------------------------------------------------------
!
!
!     MINRES is an implementation of the algorithm described in
!     the following reference:
!
!     C. C. Paige and M. A. Saunders (1975),
!     Solution of sparse indefinite systems of linear equations,
!     SIAM J. Numer. Anal. 12(4), pp. 617-629.
!     ------------------------------------------------------------------
!
!
!     MINRES development:
!           1972: First version, similar to original SYMMLQ.
```

```
!                 Later lost @#%*!
!        Oct 1995: Tried to reconstruct MINRES from
!                  1995 version of SYMMLQ.
!     30 May 1999: Need to make it more like LSQR.
!                  In middle of major overhaul.
!     19 Jul 2003: Next attempt to reconstruct MINRES.
!                  Seems to need two vectors more than SYMMLQ.  (w1, w2)
!                  Lanczos is now at the top of the loop,
!                  so the operator Aprod is called in just one place
!                  (not counting the initial check for symmetry).
!     22 Jul 2003: Success at last.  Preconditioning also works.
!                  minres.f added to http://www.stanford.edu/group/SOL/.
!
!     FUTURE WORK: A stopping rule is needed for singular systems.
!                  We need to estimate ||Ar|| as in LSQR.  This will be
!                  joint work with Sou Cheng Choi, SCCM, Stanford.
!                  Note that ||Ar|| small => r is a null vector for A.
!
!
!     Michael A. Saunders              na.msaunders@na-net.ornl.gov
!     Department of MS&E               saunders@stanford.edu
!     Stanford University
!     Stanford, CA 94305-4026          (650) 723-1875
!     ------------------------------------------------------------------
!
!
!     Subroutines and functions
!
!     USER        Aprod , Msolve
!     BLAS1       daxpy , dcopy , ddot  , dnrm2  } These are all in
!     Utilities   daxpy2, dload2, dscal2         } the file minresblas.f

!     Functions

      external            ddot  , dnrm2
      double precision    ddot  , dnrm2

!     Local variables

      double precision    alfa  , beta  , beta1 , cs    ,
     $                    dbar  , delta , denom , diag  ,
     $                    eps   , epsa  , epsln , epsr  , epsx  ,
     $                    gamma , gbar  , gmax  , gmin  ,
     $                    oldb  , oldeps, qrnorm, phi   , phibar,
     $                    rhs1  , rhs2  , s     , sn    , t     ,
     $                    tnorm2, ynorm2, z
      integer             i
```

```
      logical            debug, prnt

      double precision   zero, one, two, ten
      parameter        ( zero = 0.0d+0,  one =  1.0d+0,
     $                   two  = 2.0d+0,  ten = 10.0d+0 )

      character*16       enter, exit
      character*52       msg(-1:8)

      data               enter /' Enter MINRES...'/,
     $                   exit  /' Exit  MINRES...'/

      data               msg
     $ / 'beta2 = 0...If M = I, b and x are eigenvectors of A',
     $   'beta1 = 0...The exact solution is  x = 0',
     $   'Requested accuracy achieved, as determined by rtol',
     $   'Reasonable accuracy achieved, given eps',
     $   'x has converged to an eigenvector',
     $   'Acond has exceeded 0.1/eps',
     $   'The iteration limit was reached',
     $   'Aprod   does not define a symmetric matrix',
     $   'Msolve does not define a symmetric matrix',
     $   'Msolve does not define a pos-def preconditioner' /
!     ------------------------------------------------------------------

      debug = .false.

!     ------------------------------------------------------------------
!     Compute eps, the machine precision.  The call to daxpy is
!     intended to fool compilers that use extra-length registers.
!     31 May 1999: Hardwire eps so the debugger can step thru easily.
!     ------------------------------------------------------------------
      eps    = 2.22d-16     ! Set eps = zero here if you want it computed.

      if (eps .le. zero) then
         eps    = two**(-12)
   10    eps    = eps / two
         x(1)   = eps
         y(1)   = one
         call daxpy ( 1, one, x, 1, y, 1 )
         if (y(1) .gt. one) go to 10
         eps    = eps * two
      end if

!     ------------------------------------------------------------------
!     Print heading and initialize.
!     ------------------------------------------------------------------
```

```
      if (nout .gt. 0) then
         write(nout, 1000) enter, n, checkA, precon,
     $                     itnlim, rtol, shift
      end if
      istop  = 0
      itn    = 0
      Anorm  = zero
      Acond  = zero
      rnorm  = zero
      ynorm  = zero
      call dload2( n, zero, x )

!     ------------------------------------------------------------------
!     Set up y and v for the first Lanczos vector v1.
!     y  =  beta1 P' v1,  where  P = C**(-1).
!     v is really P' v1.
!     ------------------------------------------------------------------
      call dcopy ( n, b, 1, y , 1 )          ! y  = b
      call dcopy ( n, b, 1, r1, 1 )          ! r1 = b
      if ( precon ) call Msolve( n, b, y )
      beta1  = ddot  ( n, b, 1, y, 1 )

      if (beta1 .lt. zero) then     ! M must be indefinite.
         istop = 8
         go to 900
      end if

      if (beta1 .eq. zero) then     ! b = 0 exactly.  Stop with x = 0.
         istop = 0
         go to 900
      end if

      beta1  = sqrt( beta1 )        ! Normalize y to get v1 later.

!     ------------------------------------------------------------------
!     See if Msolve is symmetric.
!     ------------------------------------------------------------------
      if (checkA  .and.  precon) then
         call Msolve( n, y, r2 )
         s      = ddot  ( n, y, 1, y, 1 )
         t      = ddot  ( n,r1, 1,r2, 1 )
         z      = abs( s - t )
         epsa   = (s + eps) * eps**0.33333d+0
         if (z .gt. epsa) then
            istop = 7
            go to 900
         end if
```

```
      end if

!     ------------------------------------------------------------------
!     See if Aprod  is symmetric.
!     ------------------------------------------------------------------
      if (checkA) then
         call Aprod ( n, y, w )
         call Aprod ( n, w, r2 )
         s       = ddot  ( n, w, 1, w, 1 )
         t       = ddot  ( n, y, 1,r2, 1 )
         z       = abs( s - t )
         epsa    = (s + eps) * eps**0.33333d+0
         if (z .gt. epsa) then
            istop = 6
            go to 900
         end if
      end if

!     ------------------------------------------------------------------
!     Initialize other quantities.
!     ------------------------------------------------------------------
      oldb   = zero
      beta   = beta1
      dbar   = zero
      epsln  = zero
      qrnorm = beta1
      phibar = beta1
      rhs1   = beta1
      rhs2   = zero
      tnorm2 = zero
      ynorm2 = zero
      cs     = - one
      sn     = zero
      call dload2( n, zero, w  )          ! w  = 0
      call dload2( n, zero, w2 )          ! w2 = 0
      call dcopy ( n, r1, 1, r2, 1 )      ! r2 = r1

      if (debug) then
         write(*,*) '␣'
         write(*,*) 'b␣␣␣␣', b
         write(*,*) 'beta␣', beta
         write(*,*) '␣'
      end if

!     ------------------------------------------------------------------
!     Main iteration loop.
!     ------------------------------------------------------------------
```

```
100 itn    = itn   + 1                  ! k = itn = 1 first time through
    if (istop .ne. 0) go to 900

    !-------------------------------------------------------------------
    ! Obtain quantities for the next Lanczos vector vk+1, k = 1, 2,...
    ! The general iteration is similar to the case k = 1 with v0 = 0:
    !
    !   p1      = Operator * v1  -  beta1 * v0,
    !   alpha1  = v1'p1,
    !   q2      = p2  -  alpha1 * v1,
    !   beta2^2 = q2'q2,
    !   v2      = (1/beta2) q2.
    !
    ! Again, y = betak P vk,  where  P = C**(-1).
    ! .... more description needed.
    !-------------------------------------------------------------------
    s      = one / beta             ! Normalize previous vector (in y).
    call dscal2( n, s, y, v )       ! v = vk if P = I

    call Aprod ( n, v, y )
    call daxpy ( n, (- shift), v, 1, y, 1 )
    if (itn .ge. 2) then
       call daxpy ( n, (- beta/oldb), r1, 1, y, 1 )
    end if

    alfa   = ddot  ( n, v, 1, y, 1 )      ! alphak

    call daxpy ( n, (- alfa/beta), r2, 1, y, 1 )
    call dcopy ( n, r2, 1, r1, 1 )
    call dcopy ( n,  y, 1, r2, 1 )
    if ( precon ) call Msolve( n, r2, y )

    oldb   = beta                         ! oldb = betak
    beta   = ddot  ( n, r2, 1, y, 1 )     ! beta = betak+1^2
    if (beta .lt. zero) then
       istop = 6
       go to 900
    end if

    beta   = sqrt( beta )                 ! beta = betak+1
    tnorm2 = tnorm2 + alfa**2 + oldb**2 + beta**2

    if (itn .eq. 1) then                  ! Initialize a few things.
       if (beta/beta1 .le. ten*eps) then  ! beta2 = 0 or ~ 0.
          istop = -1                      ! Terminate later.
       end if
      !tnorm2 = alfa**2
```

```
    gmax   = abs( alfa )                  ! alpha1
    gmin   = gmax                         ! alpha1
end if

! Apply previous rotation Qk-1 to get
!   [deltak epslnk+1] = [cs  sn][dbark    0   ]
!   [gbar k dbar k+1]   [sn -cs][alfak betak+1].

oldeps = epsln
delta  = cs * dbar  +  sn * alfa ! delta1 = 0          deltak
gbar   = sn * dbar  -  cs * alfa ! gbar 1 = alfa1      gbar k
epsln  =               sn * beta ! epsln2 = 0          epslnk+1
dbar   =            -  cs * beta ! dbar 2 = beta2      dbar k+1

! Compute the next plane rotation Qk

gamma  = sqrt( gbar**2 + beta**2 )    ! gammak
cs     = gbar / gamma                 ! ck
sn     = beta / gamma                 ! sk
phi    = cs * phibar                  ! phik
phibar = sn * phibar                  ! phibark+1

if (debug) then
   write(*,*) '␣'
   write(*,*) 'v␣␣␣␣', v
   write(*,*) 'alfa␣', alfa
   write(*,*) 'beta␣', beta
   write(*,*) 'gamma', gamma
   write(*,*) 'delta', delta
   write(*,*) 'gbar␣', gbar
   write(*,*) 'epsln', epsln
   write(*,*) 'dbar␣', dbar
   write(*,*) 'phi␣␣', phi
   write(*,*) 'phiba', phibar
   write(*,*) '␣'
end if

! Update  x.

denom = one/gamma

do i = 1, n
   w1(i) = w2(i)
   w2(i) = w(i)
   w(i)  = ( v(i) - oldeps*w1(i) - delta*w2(i) ) * denom
   x(i)  =   x(i) +   phi * w(i)
end do
```

```
! Go round again.

gmax   = max( gmax, gamma )
gmin   = min( gmin, gamma )
z      = rhs1 / gamma
ynorm2 = z**2  +  ynorm2
rhs1   = rhs2  -  delta * z
rhs2   =       -  epsln * z

! Estimate various norms and test for convergence.

Anorm  = sqrt( tnorm2 )
ynorm  = sqrt( ynorm2 )
epsa   = Anorm * eps
epsx   = Anorm * ynorm * eps
epsr   = Anorm * ynorm * rtol
diag   = gbar
if (diag .eq. zero) diag = epsa

qrnorm = phibar
rnorm  = qrnorm

! Estimate  cond(A).
! In this version we look at the diagonals of  R  in the
! factorization of the lower Hessenberg matrix,  Q * H = R,
! where H is the tridiagonal matrix from Lanczos with one
! extra row, beta(k+1) e_k^T.

Acond  = gmax / gmin

! See if any of the stopping criteria are satisfied.
! In rare cases, istop is already -1 from above (Abar = const*I).

if (istop .eq. 0) then
   if (itn    .ge. itnlim    ) istop = 5
   if (Acond  .ge. 0.1d+0/eps) istop = 4
   if (epsx   .ge. beta1     ) istop = 3
   if (qrnorm .le. epsx      ) istop = 2
   if (qrnorm .le. epsr      ) istop = 1
end if


! See if it is time to print something.

if (nout .gt. 0) then
   prnt   = .false.
```

```
         if (n      .le. 40          ) prnt = .true.
         if (itn    .le. 10          ) prnt = .true.
         if (itn    .ge. itnlim - 10) prnt = .true.
         if (mod(itn,10)  .eq.     0) prnt = .true.
         if (qrnorm .le.  ten * epsx) prnt = .true.
         if (qrnorm .le.  ten * epsr) prnt = .true.
         if (Acond  .ge. 1.0d-2/eps ) prnt = .true.
         if (istop  .ne.  0         ) prnt = .true.

         if ( prnt ) then
            if (    itn     .eq. 1) write(nout, 1200)
            write(nout, 1300) itn, x(1), qrnorm, Anorm, Acond
            if (mod(itn,10) .eq. 0) write(nout, 1500)
         end if
      end if

      go to 100
!     ------------------------------------------------------------------
!     End of main iteration loop.
!     ------------------------------------------------------------------

      ! Display final status.

  900 if (nout  .gt. 0) then
         write(nout, 2000) exit, istop, itn,
     $                     exit, Anorm, Acond,
     $                     exit, rnorm, ynorm
         write(nout, 3000) exit, msg(istop)
      end if

      return


 1000 format(// 1p,    a, 5x, 'Solution of symmetric   Ax = b'
     $       / ' n       =', i7, 5x, 'checkA =', l4, 12x,
     $         'precon =', l4
     $       / ' itnlim  =', i7, 5x, 'rtol    =', e11.2, 5x,
     $         'shift   =', e23.14)
 1200 format(// 5x, 'itn', 8x, 'x(1)', 10x,
     $         'norm(r)', 3x, 'norm(A)', 3X, 'cond(A)')
 1300 format(1p, i8, e19.10, 3e10.2)
 1500 format(1x)
 2000 format(/ 1p, a, 5x, 'istop  =', i3,   14x, 'itn     =', i8
     $       /     a, 5x, 'Anorm  =', e12.4, 5x, 'Acond  =', e12.4
     $       /     a, 5x, 'rnorm  =', e12.4, 5x, 'ynorm  =', e12.4)
 3000 format(      a, 5x, a )
```

```
      end ! subroutine MINRES
```

### Β΄.1.2 Βοηθητικές υπορουτίνες για τη MINRES.

Στη συνέχεια παραθέτουμε κάποιες βοηθητικές υπορουτίνες, τις οποίες καλεί η υπορουτίνα MINRES.

```
*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*
*     minresblas.f
*
*     This file contains Level 1 BLAS from netlib, Thu May 16 1991
*     (with declarations of the form dx(1) changed to dx(*)):
*        daxpy    dcopy    ddot
*     Also
*        dnrm2    (from NAG, I think).
*
*     Also a few utilities to avoid some of the
*     loops in MINRES (so the debugger can step past them quickly):
*        daxpy2   dload2   dscal2
*
* 15 Jul 2003: dnrm2  is now the NAG version.
*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      subroutine daxpy(n,da,dx,incx,dy,incy)
c
c     constant times a vector plus a vector.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
```

```
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue

      end ! subroutine daxpy

*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      subroutine  dcopy(n,dx,incx,dy,incy)
c
c     copies a vector, x, to a vector, y.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(*),dy(*)
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dx(ix)
        ix = ix + incx
```

```
         iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,7)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dx(i)
   30 continue
      if( n .lt. 7 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,7
        dy(i) = dx(i)
        dy(i + 1) = dx(i + 1)
        dy(i + 2) = dx(i + 2)
        dy(i + 3) = dx(i + 3)
        dy(i + 4) = dx(i + 4)
        dy(i + 5) = dx(i + 5)
        dy(i + 6) = dx(i + 6)
   50 continue

      end ! subroutine dcopy

*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      double precision function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
```

```
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     *   dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp

      end ! function ddot

*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      double precision   function dnrm2 ( n, x, incx )

      implicit           double precision (a-h,o-z)
      integer            incx, n
      double precision   x(*)

*     ==================================================================
*     dnrm2 returns the Euclidean norm of a vector via the function
*     name, so that dnrm2 := sqrt( x'*x ).
*
*     15 Jul 2003: dnrm2 obtained from SNOPT src (probably from NAG).
*                  s1flmx replaced by safe large number.
*     ==================================================================
```

```
*     double precision   s1flmx
      parameter          (one = 1.0d+0, zero = 0.0d+0)
      double precision   norm
      intrinsic          abs
*     ------------------------------------------------------------------
*     flmax = s1flmx( )
      flmax = 1.0d+50

      if ( n .lt. 1) then
         norm = zero

      else if ( n .eq. 1) then
         norm = abs( x(1) )

      else
         scale = zero
         ssq   = one

         do 10, ix = 1, 1+(n-1)*incx , incx

            if ( x(ix) .ne. zero) then
               absxi = abs( x(ix) )

               if ( scale .lt. absxi) then
                  ssq   = one + ssq*(scale/absxi)**2
                  scale = absxi
               else
                  ssq   = ssq + (absxi/scale)**2
               end if
            end if
   10    continue

         sqt = sqrt( ssq )
         if ( scale .lt. flmax/sqt) then
            norm = scale*sqt
         else
            norm = flmax
         end if
      end if

      dnrm2  = norm

      end ! function dnrm2

*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      subroutine daxpy2( n, a, x, y, z )
```

```
      implicit none
      integer            n
      double precision   a, x(n), y(n), z(n)

*     -------------------------------------------------------------------
*     daxpy2 sets z = a*x + y.
*     31 May 1999: First version written for MINRES.
*     -------------------------------------------------------------------

      integer            i

      do i = 1, n
         z(i) = a*x(i) + y(i)
      end do

      end ! subroutine daxpy2

*     ===================================================================

      subroutine dload2( n, const, x )

      implicit none
      integer            n
      double precision   const, x(n)

*     -------------------------------------------------------------------
*     dload2 loads all elements of x with const.
*     -------------------------------------------------------------------

      integer            i

      do i = 1, n
         x(i) = const
      end do

      end ! subroutine dload2

*     ===================================================================

      subroutine dscal2( n, a, x, y )

      implicit none
      integer            n
      double precision   a, x(n), y(n)

*     -------------------------------------------------------------------
```

```
*      dscal2 sets y = a*x.
*      ------------------------------------------------------------------

       integer              i

       do i = 1, n
          y(i) = a*x(i)
       end do

       end ! subroutine dscal2
```

### Β΄.1.3 Η υπορουτίνα για την κατασκευή του προρυθμιστή της MINRES.

Στη συνέχεια παραθέτουμε μια υπορουτίνα για την κατασκευή του προρυθμιστή της MINRES (βλέπε [4]).

```
c234567
      subroutine syprec( n, la, liw, a, iw, neg1, neg2)
      implicit double precision (a-h, o-z)
      double precision a(la)
      integer iw(liw)
      integer neg1, neg2
C     ------------------------------------------------------------------
C     syprec (SYmmetric PREConditioner) takes the factors
C           A = U" D U
C     from Duff and Reid"s Harwell subroutine MA27BD and changes the
C     block-diagonal matrix D to be a positive-definite matrix Dbar with
C     the same 1x1 and 2x2 block-diagonal structure.
C
C     The eigensystem D = Q E Q" is used to define Dbar = Q |E| Q",
C     where |E| contains the absolute values of the eigenvalues of D.
C     The matrix
C           Abar = U" Dbar U
C     is then an exact preconditioner for A, in the sense that SYMMLQ
C     would take only 2 iterations to solve Ax = b (or 1 iteration if
C     D = Dbar is already positive definite).
C
C     If the original matrix A is close to some other matrix K,
C     Abar should be a good preconditioner for solving K x = b.
C
C     Note that MA27 stores the elements of D(inverse) and ( - U )
C     within A and IW. However, modifying a 2x2 block of D(inverse)
C     looks the same as modifying the 2x2 block itself.
C
C     10 Mar 1989: First version.
C     Systems Optimization Laboratory, Stanford University.
```

```
C     ------------------------------------------------------------------
      intrinsic abs,sqrt
      integer alen, apos
      logical single
      parameter ( zero = 0.0d+0, one = 1.0d+0, two = 2.0d+0 )

      neg1 = 0
      neg2 = 0
      nblk = abs(iw(1))
      ipos = 2
      apos = 1

      do 100, iblk = 1, nblk
         ncols    =  iw(ipos)

         if (ncols .lt. 0) then
            nrows =  1
            ncols = - ncols
         else
            ipos  =  ipos + 1
            nrows =  iw(ipos)
         end if

C        Process the diagonals in this block.

         alen   =  ncols
         single =  .true.

         do 50, k = ipos + 1, ipos + nrows
            if ( single ) then
               alpha  =  a(apos)
               j      =  iw(k)
               single =  j .gt. 0

               if ( single ) then
                  if ( alpha .lt. zero ) then
C                 ------------------------------------------------------
C                 The 1x1 diagonal is negative.
C                 ------------------------------------------------------
                     neg1    =  neg1 + 1
                     a(apos) = - alpha
                  end if
                  else
                     beta    =  a(apos+1)
                     gamma   =  a(apos+alen)

                     if (alpha * gamma .lt. beta**2 ) then
```

```
C                  ------------------------------------------------------
C                  The 2x2 diagonal is indefinite.
C                  Find its eigensystem in the form
C
C                  ( alpha  beta  )  =  ( c   s ) ( e1   ) ( c  s )
C                  ( beta   gamma )     ( s  -c ) (   e2 ) ( s -c )
C                  ------------------------------------------------------
                      tau    =  ( gamma - alpha ) / ( two * beta )
                      t      =  abs( tau ) + sqrt( tau**2 + one )
                      t      = - one / t
                      if ( tau .lt. zero ) t = - t
                      c      =  one / sqrt( t**2 + one )
                      s      =  t * c
                      e1     =  alpha + beta * t
                      e2     =  gamma - beta * t

C                     Change e1 and e2 to their absolute values
C                     and then multiply the three 2x2 matrices
C                     to get the modified alpha, beta and gamma.


                      if ( e1 .lt. zero ) then
                         neg2   =  neg2 + 1
                         e1     = - e1
                      end if
                      if ( e2 .lt. zero ) then
                         neg2   =  neg2 + 1
                         e2     = - e2
                      end if

                      alpha        =  c**2 * e1 + s**2 * e2
                      beta         =  c*s  *(e1 - e2)
                      gamma        =  s**2 * e1 + c**2 * e2
                      a(apos     ) =  alpha
                      a(apos+1   ) =  beta
                      a(apos+alen) =  gamma
                   end if
                end if
             else
                single    =  .true.
             end if

             apos   =      apos + alen
             alen   =      alen - 1
  50      continue

          ipos   =    ipos   +  ncols + 1
```

```
  100   continue

C       end of syprec
        end
```

## Β΄.2 Wrappers για συναρτήσεις δυναμικής δέσμευσης μνήμης.

Στη συνέχεια παραθέτουμε κάποιους wrappers για συναρτήσεις δυναμικής δέσμευσης μνήμης της γλώσσας προγραμματισμού C, που υλοποιήθηκαν αρχικά από τον Michael Plexousakis (βλέπε [31]).

### Β΄.2.1 Το header file για τους Wrappers.

```
/*
 * File    : memory.h
 * Author : Michael Plexousakis, plex@tem.uoc.gr
 * Purpose: Wrappers for the memory allocation/de-allocation routines
 * */

/* Modification: Wrappers for file processing were added
 * Author       : Manos Psycharis,
 *                epsychar@math.uoc.gr , epsychar@tem.uoc.gr
 * Date         : 02/06/09
 * */

#ifndef MEMORY_INCLUDED
#define MEMORY_INCLUDED

#define MALLOC(nbytes)          m_malloc((nbytes), __FILE__, __LINE__)
#define CALLOC(count,nbytes)    m_calloc((count),(nbytes),__FILE__,__LINE__)
#define FREE(ptr)               m_free((ptr), __FILE__, __LINE__)
#define REALLOC(ptr,nbytes)     m_realloc((ptr),(nbytes),__FILE__,__LINE__)
#define STRDUP(s)               m_strdup((s), __FILE__, __LINE__)
#define FOPEN(filename,type)    m_fopen((filename),(type),__FILE__,__LINE__)
#define FCLOSE(fp)              m_fclose((fp),__FILE__,__LINE__)

extern void *m_malloc(int nbytes, const char *file, int line);
extern void *m_calloc(int count, int nbytes, const char *file, int line);
extern void  m_free(void *ptr, const char *file, int line);
extern void *m_realloc(void *ptr, int nbytes, const char *file, int line);
extern char *m_strdup(const char *src, const char *file, int line);
extern FILE *m_fopen(char *filename,char *type,const char *file,int line);
extern void m_fclose(FILE *fp,const char *file,int line);

#endif
```

### Β΄.2.2 Υλοποίηση των Wrappers.

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

void *m_malloc(int nbytes, const char *file, int line)
{
        void *ptr = 0;

        if (nbytes <= 0) {
                fprintf(stderr, "Invalid argument to MALLOC() in file %s,
line %d\n", file, line);
                exit(2);
        }

        ptr = malloc(nbytes);
        if (!ptr) {
                fprintf(stderr, "MALLOC() failed in file %s, line %d\n",
file, line);
                exit(2);
        }

        return ptr;
}

void *m_calloc(int count, int nbytes, const char *file, int line)
{
        void *ptr;

        if (count <= 0 || nbytes <= 0) {
                fprintf(stderr, "Invalid argument(s) to CALLOC() in
file %s, line %d\n", file, line);
                exit(2);
        }

        ptr = calloc(count, nbytes);
        if (!ptr) {
                fprintf(stderr, "CALLOC() failed in file %s, line %d\n",
file, line);
                exit(2);
        }

        return ptr;
}
```

```
void m_free(void *ptr, const char *file, int line)
{
        if (ptr) free(ptr);
}

void *m_realloc(void *ptr, int nbytes, const char *file, int line)
{
        void *newptr;

        if (!ptr || nbytes <= 0) {
                fprintf(stderr, "Invalid argument(s) to REALLOC() in
file %s, line %d\n", file, line);
                exit(2);
        }

        newptr = realloc(ptr, nbytes);
        if (!newptr) {
                fprintf(stderr, "REALLOC() failed in file %s, line %d\n",
file, line);
                exit(2);
        }

        return newptr;
}

char *m_strdup(const char *src, const char *file, int line)
{
        char *dst = malloc(strlen(src)+1);

        if (!dst) {
                fprintf(stderr, "STRUDP() failed in file %s, line %d\n",
file, line);
                exit(2);
        }

        return strcpy(dst, src);
}

FILE *m_fopen(char *filename,char *type,const char *file,int line)
{
        FILE *fp;
        fp=fopen(filename,type);
        if(fp == NULL){
                fprintf(stderr,"FOPEN() failed in file %s, line %d\n",
file,line);
                exit(2);
        }
```

```
        return fp;
}

void m_fclose(FILE *fp,const char *file,int line)
{
        if(fp) fclose(fp);
}
```

# Βιβλιογραφία


[1] M.R. Hestenes and E.Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research, National Bureau of Standards, 49, pp. 409-436, 1952.

[2] J.R. Bunch and B.N. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal. 8, pp. 639-655, 1971.

[3] C.C. Paige and M.A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal. 12, pp. 617-629, 1975.

[4] P.E. Gill, W. Murray, D.B. Ponceleón, and M.A. Saunders, *Preconditioners for indefinite systems arising in optimization*, SIAM J. Matrix Anal. Appl. 13, pp. 292-311, 1992.

[5] R.W. Freund and N.M. Nachtigal, *A new Krylov-subspace method for symmetric indefinite linear systems*, in Proceedings of the 14th IMACS World Congress on Computational and Applied Mathematics, W.F. Ames, ed., IMACS, 1994, pp. 1253-1256.

[6] R.Bramley, *An orthogonal projection algorithm for linear Stokes problems*, Technical Report 1190, Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois, 1992.

[7] R.Bramley, X.Wang and D.Pelletier, *Orthogonalization based iterative methods for generalized Stokes problems*, in: W.G. Habashi (Ed.), Solution Techniques for Large-Scale CFD Problems, Wiley, New York, pp. 217-238, 1995.

[8] M.Benzi and A.J. Wathen, *Some Preconditioning Techniques for Saddle Point Problems*, in W. Schilders, H. A. van der Vorst and J. Rommes, eds., Model Order Reduction: Theory, Research Aspects and Applications, Springer-Verlag (Series: Mathematics in Industry), pp. 195-211, 2008.

[9] Y.Saad and H.A. van der Vorst, *Iterative solution of linear systems in the 20th century*, J. Comput. Appl. Math., 123, pp.1-33, 2000.

[10] A.Greenbaum, *Iterative methods for solving linear systems*, vol. 17 of Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[11] Y.Saad, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second ed., 2003.

[12] H.A. van der Vorst, *Iterative Krylov methods for large linear systems*, vol. 13 of Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, 2003.

[13] R.Barrett, M.Berry, T.Chan, J.Demmel, J.Donato, J.Dongarra, V.Eijkhout, R.Pozo, C.Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[14] G.Meurant, *Computer Solution of Large Linear Systems*, North-Holland, Amsterdam,1999.

[15] C.Brezinski, *Projection Methods for Systems of Equations*, North-Holland, Amsterdam,1997.

[16] H.C. Elman, D.J. Silvester and A.J. Wathen, *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford, UK, 2005.

[17] I.S. Duff and J.K. Reid, *MA*27 – *A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Technical Report AERE R10533, Her Majesty's Stationery Office, London, 1982.

[18] I.S. Duff, *MA*57 – *A new code for the solution of sparse symmetric definite and indefinite systems*, ACM Trans. Math. Softw. **30**, pp.118-154, 2004

[19] I.S. Duff, *Sparse system solution and the HSL Library*, Technical Report RAL-TR-2006-014, Rutherford Appleton Laboratory, 2006.

[20] G. Karypis and V. Kumar, *MeTiS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version* 4.0, University of Minnesota, 1998.

[21] *HSL, A collection of Fortran codes for large-scale scientific computation*, 2007. See: `http://www.hsl.rl.ac.uk/`

[22] G.L.G. Sleijpen, H.A. van der Vorst and J. Modersitzki, *Effects of rounding errors in determining approximate solutions in Krylov solvers for symmetric indefinite linear systems*, SIAM J. Matrix Anal. Appl., 22, pp. 726-751, 2000.

[23] M. Benzi, G.H. Golub, and J. Liesen, *Numerical Solution of Saddle Point Problems*, Acta Numerica, 14, pp. 1-137, 2005.

[24] H.C. Elman, A.R. Ramage, D.J. Silvester and A.J. Wathen, `http://www.cs.umd.edu/~elman/ifiss.html`

[25] R. Bramley, D. Pelletier, *Iterative Methods and Formulations for Incompressible FEM Flow Solvers*, AIAA, Aerospace Sciences Meeting and Exhibit, 34th, 1996.

[26] O. Axelsson, M. Neytcheva, *Preconditioning methods for linear systems arising in constrained optimization problems*, Numer. Linear Algebra Appl., 10, pp. 3-31, 2003.

[27] A.J. Chorin, J.E. Marsden, *A Mathematical Introduction to Fluid Dynamics*, 3rd ed. Springer-Verlag, 1993.

[28] J.Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 3rd ed. Springer-Verlag, 1993.

[29] D. Yang, *C++ and Object-Oriented Numeric Computing for Scientists and Engineers*, Springer-Verlag, New York, January 2001.

[30] Derek Capper, *Introducing C++ for Scientists, Engineers, and Mathematicians*, Springer, 2001.

[31] Michael Plexousakis, `http://www.tem.uoc.gr/~plex`

[32] Michael Saunders, `http://www.stanford.edu/~saunders`

[33] Systems Optimization Laboratory, Stanford University, Dept of Management Science and Engineering, *Optimization Software*, `http://stanford.edu/group/SOL/software/minres.html`

[34] Y.Shapira, *Solving PDEs in C++ : Numerical Methods in a Unified Object-Oriented Approach*, O. Ghattas, Ed. Philadelphia: SIAM, 2006.