ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΧΗΜΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

ΤΟΜΕΑΣ ΙΙ
ΑΝΑΛΥΣΗΣ,ΣΧΕΔΙΑΣΜΟΥ & ΑΝΑΠΤΥΞΗΣ ΔΙΕΡΓΑΣΙΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

Επίλυση μη γραμμικών προβλημάτων και εφαρμογή μεθόδων παραμετρικού βηματισμού με τη μέθοδο πεπερασμένων στοιχείων στην υπολογιστική πλατφόρμα FEniCS

Διπλωματική εργασία
του

Αλέξανδρου-Γεώργιου Σουραή

Επιβλέπων καθηγητής

Ανδρέας Μπουντουβής

Αθήνα,2019

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή της διπλωματικής μου, Ανδρέα Μπουντουβή που μου έδωσε την ευκαιρία να ασχοληθώ εκτενέστερα με το αντικείμενο της Υπολογιστικής Μηχανικής. Το ενδιαφέρον, η εμπιστοσύνη και οι εύστοχες παρατηρήσεις του διαμόρφωσαν το κατάλληλο περιβάλλον για την ολοκλήρωση αυτής της εργασίας.

Φυσικά δε θα μπορούσα να μην ευχαριστήσω τον Δρ. Αντώνη Σπυρόπουλο για την πολύτιμη και διαρκή υποστήριξή του σε θέματα παράλληλης επεξεργασίας, προγραμματισμού και αριθμητικής ανάλυσης. Η συνεργασία μας ήταν άρτια και αποτέλεσε κινητήρια δύναμη για την εκπόνηση της διπλωματικής εργασίας.

Περιεχόμενα

Περίληψη

Στη παρούσα διπλωματική εργασία εξετάστηκαν οι υπολογιστικές δυνατότητες του λογισμικού FEniCS με έμφαση στην επίλυση και ανάλυση του χώρου των λύσεων μη γραμμικών προβλημάτων. Το FEniCS είναι μια δημοφιλής υπολογιστική πλατφόρμα ανοιχτού λογισμικού που χρησιμοποιεί τη μέθοδο Πεπερασμένων Στοιχείων για τη διακριτοποίηση και αριθμητική επίλυση μερικών διαφορικών εξισώσεων. Το FEniCS διαφέρει έναντι άλλων λογισμικών καθώς είναι δομημένο υπό τη μορφή βιβλιοθηκών σε γλώσσα C++/Python δίνοντας τη δυνατότητα εξατομίκευσης των υπολογιστικών μοντέλων του χρήστη. Με βάση αυτή τη δυνατότητα του λογισμικού επιλύθηκε αρχικά ένα δι-διάστατο γραμμικό πρόβλημα κατανομής θερμοκρασίας σε πτερύγιο και εξετάστηκε η δυνατότητα εισαγωγής πλέγματος από το λογισμικό Gmsh. Στη συνέχεια επιλύθηκε ένα τρι-διάστατο μη γραμμικό πρόβλημα Bratu με ομογενείς συνοριακές συνθήκες Dirichlet. Εξετάστηκε η εφαρμογή της μεθόδου Newton και η αυτόματη εξαγωγή της Ιακωβιανής σε μεταβολική μορφή. Επίσης εξετάστηκε η σύγκλιση της Newton στην αυτοματοποιημένη μορφή που παρέχει το FEniCS. Τα παραπάνω προβλήματα επιλύθηκαν παράλληλα σε υπολογιστικό σύστημα κοινής μνήμης προκειμένου να αξιολογηθεί η παράλληλη επιτάχυνση και η δέσμευση μνήμης. Ο υπολογιστικός χρόνος επίλυσης του 3D προβλήματος Bratu με $4 \cdot 10^6$ βαθμούς ελευθερίας μειώθηκε έως και 75% σε 8 πυρήνες ενώ η μέγιστη δέσμευση μνήμης δε ξεπέρασε τα 22 GB. Στο τελευταίο κεφάλαιο της εργασίας αναπτύχθηκε αναλυτικά η διαδικασία εφαρμογής μεθόδων ανάλυσης του χώρου των λύσεων μη γραμμικών προβλημάτων με τρόπο φιλικό προς το FEniCS προκειμένου να είναι δυνατή η παράλληλη επεξεργασία. Εφαρμόστηκε η μέθοδος παραμετρικού βηματισμού πρώτης τάξης για ομαλές λύσεις του προβλήματος Bratu ενώ για την ιχνηλάτηση του κλάδου λύσεων πέρα ιδιάζοντων σημείων όπως σημείων στροφής χρησιμοποιήθηκαν η παραμετροποίηση μήκους τόξου με τη μορφή αλγορίθμου Block-elimination και η μέθοδος αποπληθωρισμού (deflation). Για την καλύτερη αξιολόγηση της μεθόδου αποπληθωρισμού στο FEniCS επιλύθηκε επίσης ένα δι-διάστατο πρόβλημα ροής ρευστού σε κανάλι με απότομη διαστολή. Κατέστη δυνατός ο εντοπισμός 5 εκ των 6 μη συμμετρικών κλάδων που προκύπτουν από διακλάδωση (bifurcation) του ασταθούς συμμετρικού κλάδου μετά από ένα κρίσιμο αριθμό Reynolds χρησιμοποιώντας μόνο τη μέθοδο αποπληθωρισμού.

Λέξεις-Κλειδιά: FEniCS, Μέθοδος Πεπερασμένων Στοιχείων, μη γραμμικά προβλήματα, Gmsh, 3D Bratu, παράλληλη επεξεργασία, παραμετροποίηση μήκους τόξου, παραμετρικός βηματισμός με αποπληθωρισμό

Solution of nonlinear problems and application of parametric continuation with the Finite Element method using FEniCS computational platform

Diploma thesis
Alexander G. Sourais

Advisor
Andreas G. Boudouvis, Professor NTUA

Abstract

In this diploma thesis the computational capabilities of FEniCS software were examined, with emphasis on nonlinear problems and solution space analysis. FEniCS is a popular open-source computational platform for discretization and solution of partial differential equations using the Finite Element method. FEniCS differs from other software available because it is built as a collection of C++/Python libraries allowing user customization of computational models. Based on this capability a two dimensional problem of calculating the temperature distribution in a fin was solved first using the Gmsh software for mesh generation. Then a 3D Bratu problem with homogeneous Dirichlet boundary conditions was solved. The application of Newton's method, the automatic derivation of the Jacobian as a variational form as well as the convergence of Newton's method in an automated form were examined. The two problems were solved with parallel computing in a shared memory multiprocessor system so that the parallel speedup and memory usage could be evaluated. The computational time required for the solution of 3D Bratu problem with $4 \cdot 10^6$ degrees of freedom was reduced up to 75 % using 8 cores and the maximum memory usage did not exceed 22 GB. In the last chapter of this work is described in detail the implementation of parameter continuation methods for analyzing the solution space of nonlinear problems with a FEniCS-friendly format suitable for parallel computations. A first-order continuation scheme was implemented for regular solutions of 3D Bratu problem while arc-length continuation as a block-elimination type of algorithm and deflation continuation were used for computing branches of solutions past singular points, namely turning points. For a better evaluation of the deflation method in FEniCS, a problem of fluid flow in a two-dimensional channel with sudden expansion was also solved. This type of flow undergoes an one-sided bifurcation and the symmetric solution is unstable past a critical Reynolds number. The discovery of 5 out of 6 nonsymmetrical solution branches was achieved using only the deflation method.

Key-words: FEniCS, Finite Element Method, nonlinear problems, Gmsh, 3D Bratu, parallel processing, arc-length continuation, deflation continuation

Εισαγωγή

Η χρήση υπολογιστικών πακέτων για την επίλυση μαθηματικών μοντέλων που προκύπτουν από ρεαλιστικά φυσικά προβλήματα είναι πλέον επιτακτική. Ειδικότερα η ραγδαία εξέλιξη των υπολογιστικών συστημάτων και της παράλληλης επεξεργασίας, επιτρέπει την εκτέλεση πολύ απαιτητικών υπολογισμών μεγάλης κλίμακας και την αύξηση της πολυπλοκότητας των μοντέλων. Ταυτόχρονα υπάρχει μια αναπτυσσόμενη τάση χρήσης λογισμικού ανοιχτού κώδικα ως εναλλακτική λύση σε υπολογιστικούς προσομοιωτές κλειστού κώδικα. Τα πλεονεκτήματα που παρουσιάζει το ανοιχτό λογισμικό είναι η δυνατότητα εύκολης τροποποίησης, αλληλεπίδρασης και επιπλέον ανάπτυξης του από πολλούς χρήστες παγκοσμίως. Επιπλέον ένα μεγάλο πλήθος από λογισμικά ανοιχτού κώδικα είναι διαθέσιμα ελεύθερα στο διαδίκτυο. Επομένως μια ερευνητική προσπάθεια μπορεί να απαλλαγεί από το κόστος αγοράς λογισμικού ή την ανάγκη ανάπτυξης λογισμικού από την αρχή.

Το FEniCS project αποτελεί ένα παράδειγμα ανοιχτού λογισμικού καθώς αναπτύσσεται και συντηρείται ως ελεύθερα διαθέσιμο ανοιχτό λογισμικό από μια παγκόσμια ομάδα επιστημόνων. Το FEniCS είναι μια υπολογιστική πλατφόρμα ανοιχτού λογισμικού για την επίλυση μερικών διαφορικών εξισώσεων με τη μέθοδο Πεπερασμένων Στοιχείων. Το λογισμικό είναι δομημένο με τη μορφή βιβλιοθηκών σε γλώσσα προγραμματισμού C++/Python και επιτρέπει σε μεγάλο βαθμό την εξατομίκευση της λειτουργίας του από το χρήστη. Επίσης μπορεί να συνδυαστεί με άλλες βιβλιοθήκες Python ή πηγαίο κώδικα.

Στα πλαίσια αυτής της διπλωματικής εργασίας εξετάστηκαν διάφορες δυνατότητες του λογισμικού σε προβλήματα φαινομένων μεταφοράς. Μεγαλύτερη έμφαση δώθηκε στην ανάλυση του χώρου των λύσεων μη γραμμικών προβλημάτων συναρτήσει των παραμέτρων. Στη πράξη η γνώση της συμπεριφοράς της λύσης καθώς μεταβάλλονται γεωμετρικές ή λειτουργικές παράμετροι είναι ίσως πιο σημαντική από την ίδια τη λύση. Εξετάστηκε η εφαρμογή αλγορίθμων και μεθόδων παραμετρικού βηματισμού σε δύο μη γραμμικά προβλήματα αξιοποιώντας τα υπολογιστικά εργαλεία του FEniCS. Κυριότερος στόχος ήταν να διατηρηθεί η δυνατότητα παράλληλης επεξεργασίας που προσφέρει το λογισμικό. Εκτός από την εφαρμογή αυτών των μεθόδων αξιολογήθηκαν τόσο οι υπολογιστικές επιδόσεις όσο και η ευκολία και ευελιξία διαμόρφωσης του υπολογιστικού κώδικα κατά την επίλυση γραμμικών και μη γραμμικών προβλημάτων, σειριακά ή παράλληλα.

Κεφάλαιο 1

Η Μέθοδος Πεπερασμένων Στοιχείων

Σε αυτό το κεφάλαιο αναπτύσσεται η μέθοδος Πεπερασμένων Στοιχείων (Finite-Element method) για τη διακριτοποίηση και επίλυση διαφορικών εξισώσεων που αναδύονται από διάφορα φυσικά προβλήματα .

Η μέθοδος Πεπερασμένων Στοιχείων ή FEM (Finite Element Method) [1] είναι μια αριθμητική μέθοδος που χρησιμοποιείται για την επίλυση διαφορικών εξισώσεων που υπεισέρχονται σε μεγάλο πλήθος φυσικών προβλημάτων. Τυπικά πεδία ενδιαφέροντος από όπου αναδύονται τέτοια προβλήματα είναι η δομική ανάλυση, η μεταφορά θερμότητας, η ροή των ρευστών, η μεταφορά μάζας και ο ηλεκτρομαγνητισμός. Η αναλυτική επίλυση τέτοιων προβλημάτων συνεπάγεται την επίλυση προβλημάτων συνοριακών τιμών για μερικές διαφορικές εξισώσεις. Ωστόσο στις περισσότερες περιπτώσεις η επίλυση μερικών διαφορικών εξισώσεων αναλυτικά απαιτεί την απλοποίηση των φυσικών μοντέλων κάτι που μπορεί να οδηγήσει σε μη ρεαλιστικά αποτελέσματα. Η σημαντική εξέλιξη των υπολογιστικών συστημάτων επιτρέπει την αριθμητική επίλυση περίπλοκων προβλημάτων συνοριακών τιμών με ικανοποιητική ακρίβεια [2, pp. 2-3].

Η μέθοδος αναπτύχθηκε αρχικά από την ανάγκη επίλυσης περίπλοκων προβλημάτων ελαστικότητας και δομικής ανάλυσης στις αρχές της δεκαετίας του 1940 από τους A. Hrennikoff και R.Courant. Αν και η προσέγγιση της μεθόδου μεταξύ των δύο πρωτοπόρων ήταν σαφώς διαφορετική, μοιράζονταν ένα κοινό χαρακτηριστικό:τη διακριτοποίηση ενός συνεχούς χωρίου Ω σε ένα σύνολο από μικρότερα διακριτά υποχωρία που συνήθως αποκαλούνται στοιχεία (elements). Μάλιστα η συνεισφορά του Courant βασίστηκε στην εξέλιξη αποτελεσμάτων από παλιότερες προσπάθειες στην επίλυση μερικών διαφορικών εξισώσεων από τους Rayleigh, Ritz και Galerkin. Η μέθοδος των πεπερασμένων στοιχείων έλαβε την πραγματική της μορφή τις δεκαετίες του 1960-1970 από τον J. H. Argyris και τους συνεργάτες του στο Πανεπιστήμιο της Στουτγκάρδης, τον R. W. Clough και τους συνεργάτες του στο πανεπιστήμιο UC Berkeley και άλλους.

Η μεγάλη διεθνής αποδοχή και επιτυχία της μεθόδου στην επιστημονική κοινότητα οφείλεται κυρίως στη γενικότητα και τη κομψότητα της, επιτρέποντας ένα ευρύ φάσμα από διαφορικές εξισώσεις που προκύπτουν από όλες τις περιοχές της επιστήμης να αναλυθούν και να επιλυθούν υπό ένα κοινό υπόβαθρο. Επιπλέον ένας παράγοντας που συνεισφέρει στην επιτυχία της μεθόδου των πεπερασμένων στοιχείων είναι η ευελιξία όσον αφορά τη διαμόρφωση των προβλημάτων, επιτρέποντας τον έλεγχο των ιδιοτήτων της διακριτοποίησης από την επιλογή του κατάλληλου προσεγγιστικού χώρου πεπερασμένων στοιχείων [3, p. 73]. Άλλο ένα πλεονέκτημα της μεθόδου είναι η στενή σχέση μεταξύ της μεταβολικής διαμόρφωσης των διαφορικών εξισώσεων (weak or variational formulation) και της διαμόρφωσης του αριθμητικού προβλήματος. Για παράδειγμα η θεωρία από μόνη της επιτρέπει την εκτίμηση του σφάλματος ή των ορίων μεταξύ των οποίων αυτό βρίσκεται όταν το αριθμητικό μοντέλο επιλύεται σε υπολογιστή[1] *. Επίσης η διαίρεση του χωρίου σε απλούστερα υποχωρία επιτρέπει:

- την ακριβή αναπαράσταση σύνθετων γεωμετριών
- τη συμπερίληψη μη ομοιογενών ιδιοτήτων του υλικού μέσου
- την εύκολη αναπαράσταση της λύσης
- την καταγραφή τοπικών φαινομένων

* Παραπομπές με άνω δείκτη αντιστοιχούν σε αναφορές με ηλεκτρονικό σύνδεσμο (βλ. Βιβλιογραφία)

Στόχος της μεθόδου Πεπερασμένων Στοιχείων είναι η διακριτοποίηση του προβλήματος ώστε η λύση του να προκύψει από ένα σύστημα αλγεβρικών εξισώσεων. Τα τυπικά βήματα που περιλαμβάνει η μέθοδος είναι:

- Διαίρεση του χωρίου του προβλήματος σε ένα σύνολο από υποχωρία το καθένα από τα οποία συνοδεύονται από ένα σύνολο εξισώσεων που ανάγονται στο αρχικό πρόβλημα
- Συστηματικός μετασχηματισμός όλων των τοπικών εξισώσεων σε ένα ολικό σύνολο εξισώσεων για την τελική επίλυση. Αυτό το σύστημα εξισώσεων έχει γνωστές τεχνικές επίλυσης.

Οι τοπικές εξισώσεις που αντιστοιχούν στα υποχωρία είναι μια προσέγγιση των αρχικών διαφορικών εξισώσεων. Συνήθως η κατασκευή αυτών των εξισώσεων γίνεται με τη μέθοδο σταθμισμένων υπολοίπων Galerkin [2] (Galerkin's method of weighted residuals) . Η μαθηματική αυτή διαδικασία απαιτεί τη κατασκευή ενός ολοκληρώματος του εσωτερικού γινομένου της αρχικής εξίσωσης με μια συνάρτηση στάθμισης (ή συνάρτηση βάσης). Αυτό το ολοκλήρωμα πρέπει να ισούται με το μηδέν. Με άλλα λόγια αυτή η διαδικασία ελαχιστοποιεί το σφάλμα προσαρμόζοντας συναρτήσεις γνωστής μορφής στις διαφορικές εξισώσεις. Οι εξισώσεις αυτές οδηγούν:

- Σε ένα σύστημα αλγεβρικών εξισώσεων για προβλήματα σε μόνιμη κατάσταση
- Σε ένα σύστημα συνήθων διαφορικών εξισώσεων για χρονικά-μεταβαλλόμενα προβλήματα

Οι εξισώσεις είναι γραμμικές αν οι μερικές διαφορικές εξισώσεις είναι γραμμικές ή μη γραμμικές αν οι μερικές διαφορικές εξισώσεις είναι μη γραμμικές. Τα γραμμικά συστήματα αλγεβρικών εξισώσεων επιλύονται με αριθμητικές μεθόδους γραμμικής άλγεβρας ενώ οι σύνηθεις διαφορικές εξισώσεις με μεθόδους όπως οι Euler και Runge-Kutta. Στη περίπτωση της μη γραμμικότητας απαιτείται επιπλέον η εφαρμογή κάποιας επαναληπτικής μεθόδου (Newton, Picard κλπ).

Στο δεύτερο βήμα που αναφέρθηκε παραπάνω το ολικό σύστημα εξισώσεων προκύπτει από τις τοπικές εξισώσεις με μετασχηματισμό των συντεταγμένων των κόμβων του υποχωρίου στις συντεταγμένες των κόμβων στο συνολικό χωρίο. Αυτή η διαδικασία ονομάζεται ισοπαραμετρική απεικόνιση [2, p. 26].

Έστω

$$\mathcal{L}u - f = 0 \ \sigma\tau o \ \Omega$$

$$u = u_0 \ \sigma\varepsilon \ \kappa\acute{\alpha}\pi o\iota o \ \upsilon\pi o\sigma\acute{\upsilon}\nu o\rho o \ \Gamma_D \ \subset \ \partial\Omega$$

$$\frac{\partial^m u}{\partial n^m} = g \ \ \sigma\varepsilon \ \kappa\acute{\alpha}\pi o\iota o \ \upsilon\pi o\sigma\acute{\upsilon}\nu o\rho o \ \Gamma_N \ \subset \ \partial\Omega \qquad (1.1)$$

ένα πρόβλημα συνοριακών τιμών του οποίου αναζητείται η λύση u στο χωρίο Ω με τη μέθοδο πεπερασμένων στοιχείων. $\mathcal{L}$ είναι ένας διαφορικός τελεστής (γραμμικός ή μη) που δρα στη συνάρτηση u και m η τάξη της παραγώγου που εμφανίζεται στη γενικευμένη συνοριακή συνθήκη Neumann.

Για τη διακριτοποίηση του προβλήματος (1.1) ολοκληρώνεται το εσωτερικό γινόμενο της διαφορικής εξίσωσης με μια γνωστή συνάρτηση (test function) έστω υ:

$$\int_{\Omega} (\mathcal{L}u - f)\upsilon \, dx = 0 \Rightarrow$$

$$\Rightarrow \int_{\Omega} \mathcal{L}u \, \upsilon \, dx - \int_{\Omega} f \, \upsilon \, dx = 0 \tag{1.2}$$

Ας υποθέσουμε για λόγους απλότητας ότι ο διαφορικός τελεστής $\mathcal{L}$ είναι γραμμικός και η μεγαλύτερη τάξη διαφόρισης που εμφανίζεται είναι m+1. Ορίζουμε $\mathcal{L}_m$ τον διαφορικό τελεστή που περιέχει όλους τους διαφορικούς όρους του $\mathcal{L}$ εκτός της διαφόρισης m+1 τάξης. Με ολοκλήρωση κατά παράγοντες του όρου m+1 τάξης προκύπτει για την (1.2):

$$\int_{\partial\Omega} \frac{\partial^m u}{\partial n^m} \upsilon \, ds - \int_{\Omega} \nabla^m u \cdot \nabla \upsilon \, dx \; + \int_{\Omega} \mathcal{L}_m u \, \upsilon \, dx - \int_{\Omega} f \, \upsilon \, dx = 0 \Rightarrow$$

$$\Rightarrow \int_{\partial\Omega} g \, \upsilon \, ds - \int_{\Omega} \nabla^m u \cdot \nabla \upsilon \, dx \; + \int_{\Omega} \mathcal{L}_m u \, \upsilon \, dx - \int_{\Omega} f \, \upsilon \, dx = 0$$

Επομένως σε μεταβολική μορφή αναζητείται λύση $u \in V$ στο παρακάτω πρόβλημα:

$$\int_{\Gamma_N} g \, \upsilon \, ds - \int_{\Omega} \nabla^m u \cdot \nabla \upsilon \, dx \; + \int_{\Omega} \mathcal{L}_m u \, \upsilon \, dx - \int_{\Omega} f \, \upsilon \, dx = 0 \quad \forall \, \upsilon \in \hat{V} \tag{1.3}$$

Οι συναρτησιακοί χώροι $V, \hat{V}$ ορίζονται ως εξής:

$$V = \{\upsilon \in H^1(\Omega) : \upsilon = u_0 \; στο \; \Gamma_D \} \tag{1.4}$$

$$\hat{V} = \{\upsilon \in H^1(\Omega) : \upsilon = 0 \; στο \; \Gamma_D \} \tag{1.5}$$

Για την επίλυση του διακριτού μεταβολικού προβλήματος θα πρέπει να γίνει κατάλληλη διακριτοποίηση των χώρων (1.4), (1.5). Εν συντομία ορίζουμε μια βάση $\{\varphi_j\}_{j=1}^{N}$ για το V και μια βάση $\{\hat{\varphi}_i\}_{i=1}^{N}$ για τον $\hat{V}$. Ν είναι η διάσταση των χώρων που ταυτίζεται με το πλήθος των βαθμών ελευθερίας (ή των κόμβων του πλέγματος).

Η λύση u προσεγγίζεται ως ένας γραμμικός συνδυασμός των συναρτήσεων βάσης :

$$u = \sum_{j=1}^{N} U_j \varphi_j(x) \tag{1.6}$$

όπου $\mathrm{U} \in \mathbb{R}^{\mathrm{N}}$ το διάνυσμα των βαθμών ελευθερίας ή διαφορετικά των διακριτών τιμών της λύσης στους κόμβους του πλέγματος.

Το πρόβλημα (1.3) μετά τη διακριτοποίηση και με βάση την εξίσωση (1.6) παίρνει τη μορφή:

$$\int_{\Gamma_N} g\,\hat{\varphi}_i\,ds - \sum_{j=1}^{N} U_j \int_{\Omega} \nabla^m \varphi_j \cdot \nabla\hat{\varphi}_i\,dx + \sum_{j=1}^{N} U_j \int_{\Omega} \mathcal{L}_m \varphi_j\,\hat{\varphi}_i\,dx - \int_{\Omega} f\,\hat{\varphi}_i\,dx = 0$$

$$i = 1,2,\dots N \tag{1.7}$$

Η προσεγγιστική λύση u της μορφής (1.6) μπορεί να υπολογιστεί με την επίλυση του γραμμικού συστήματος [3, pp. 74-76]:

$$AU = b \tag{1.8}$$

όπου

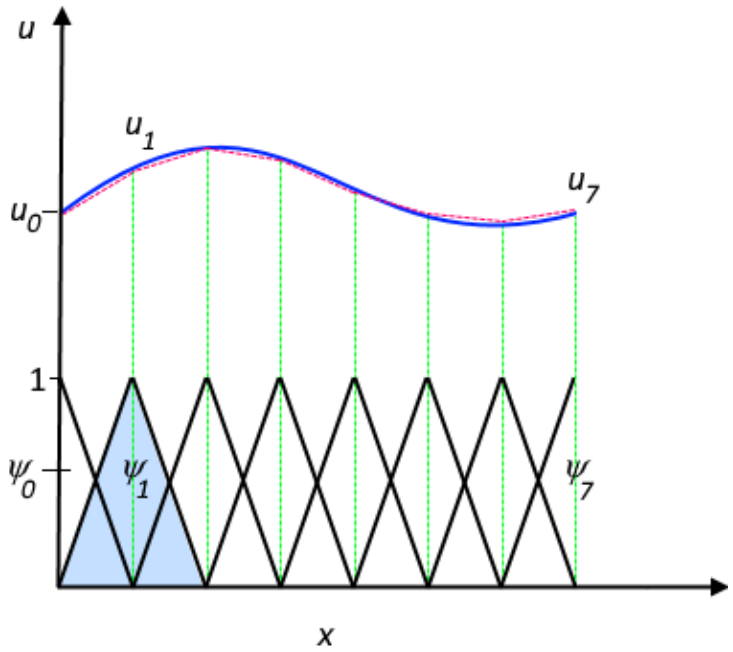
$$A_{ij} = -\int_{\Omega} \nabla^m \varphi_j \cdot \nabla\hat{\varphi}_i\,dx + \int_{\Omega} \mathcal{L}_m \varphi_j\,\hat{\varphi}_i dx \tag{1.9}$$

$$b_i = \int_{\Omega} f\,\hat{\varphi}_i\,dx - \int_{\Gamma_N} g\,\hat{\varphi}_i\,ds \tag{1.10}$$

Οι συναρτήσεις βάσης φ είναι πολυώνυμα (συνήθως 1ου ή 2ου βαθμού) των συντεταγμένων του διανύσματος θέσης. Κάθε μία από τις συναρτήσεις βάσης είναι μη μηδενική μόνο σε μια περιοχή του χωρίου Ω. Συγκεκριμένα μια συνάρτηση βάσης φ_j είναι ίση με τη μονάδα μόνο σε ένα κόμβο έστω k και μηδενική σε όλους τους άλλους κόμβους [2, p. 3]:

$$\varphi_j(x_k) = 1\ j = k \tag{1.11}$$

$$\varphi_j(x_k) = 0\ j \neq k \tag{1.12}$$



Εικόνα[2] 1.1 Η συνάρτηση u (μπλε χρώμα) προσεγγίζεται (κόκκινο χρώμα) με ένα γραμμικό συνδυασμό από συναρτήσεις βάσης (μαύρο χρώμα).

Κεφάλαιο 2

Η Πλατφόρμα ανοιχτού λογισμικού FEniCS

Σε αυτό το κεφάλαιο παρουσιάζονται κάποια γενικά στοιχεία για το FEniCS. Περιγράφεται συνοπτικά η δομή και η λειτουργία του λογισμικού με βάση τις βιβλιοθήκες από τις οποίες απαρτίζεται.

Το FEniCS[3] είναι μια δημοφιλής υπολογιστική πλατφόρμα ανοιχτού λογισμικού για την επίλυση μερικών διαφορικών εξισώσεων (PDEs). Το FEniCS επιτρέπει στους χρήστες να μετατρέψουν γρήγορα και αποτελεσματικά επιστημονικά μοντέλα σε κώδικα πεπερασμένων στοιχείων. Οι υψηλού επιπέδου διεπαφές C++ και Python διευκολύνουν το χρήστη κατά τη γνωριμία του με το λογισμικό αλλά ταυτόχρονα παρέχουν πολύ ισχυρές δυνατότητες σε προχωρημένους προγραμματιστές. Το FEniCS μπορεί να εγκατασταθεί και να εκτελεστεί σε ένα πλήθος από πλατφόρμες: από προσωπικούς υπολογιστές μέχρι υψηλής απόδοσης υπολογιστικές συστοιχίες.

2.1 Μερικά ιστορικά στοιχεία

Το FEniCS project ξεκίνησε το 2013 ως μια προσπάθεια να αυτοματοποιηθεί η επίλυση μαθηματικών προβλημάτων που προκύπτουν από διαφορικές εξισώσεις με τη συνεργασία των Πανεπιστημίων του Σικάγο[4] και του Τεχνολογικού Πανεπιστημίου Chalmers[5] στη Σουηδία [3, p. xi]. Η ονομασία του λογισμικού προκύπτει από τα ακρωνύμια FE (Finite Elements) και CS (Computational Software). Το λογισμικό αρχικά είχε 2 κύριες βιβλιοθήκες:τη DOLFIN και τη FIAT. Στη συνέχεια με την εξέλιξη του λογισμικού προστέθηκαν και άλλες βιβλιοθήκες όπως οι FFC, Instant, UFC and UFL.

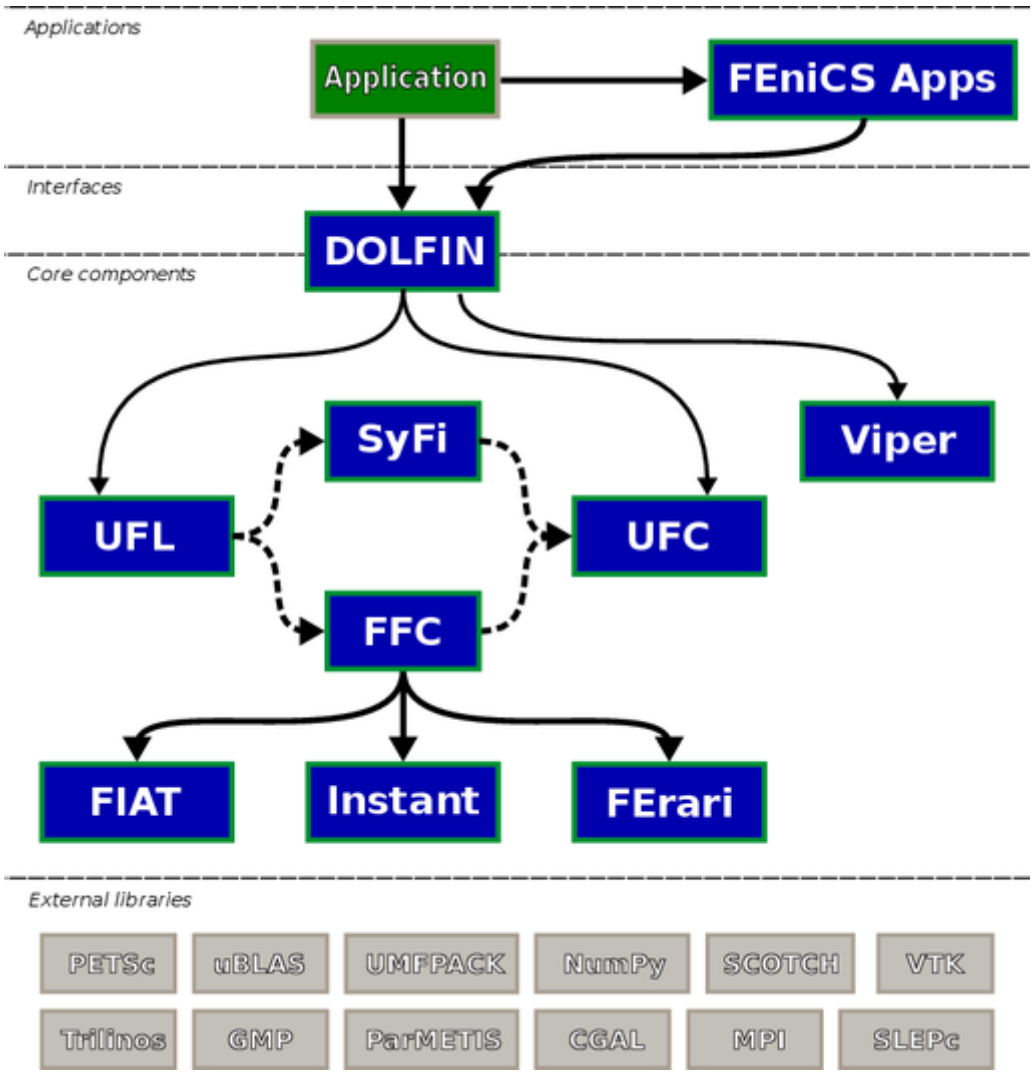
Εικόνα 2.1 Το λογότυπο του FEniCS.

2.2 Βιβλιοθήκες

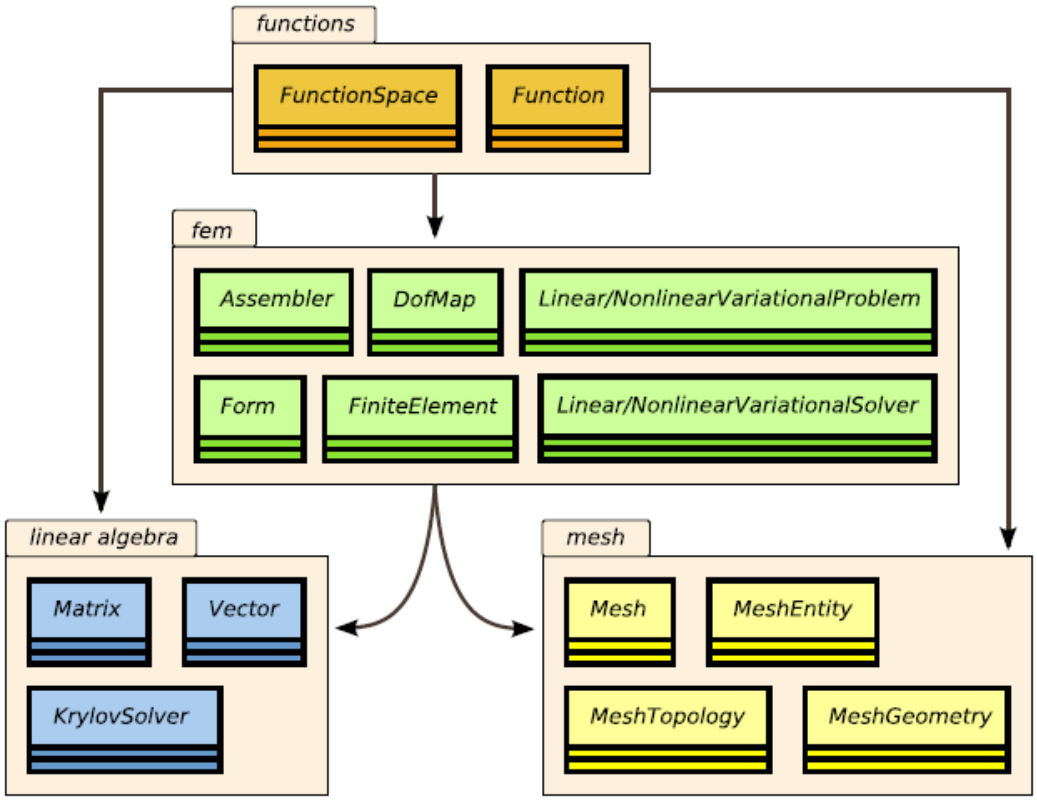
Το λογισμικό FEniCS έχει σχεδιαστεί με πρότυπο τύπου “ομπρέλας” (umbrella project) για μια συλλογή από διαλειτουργικά στοιχεία ή βιβλιοθήκες. Η δομή και η λειτουργία των βασικότερων βιβλιοθηκών περιγράφεται στις παρακάτω παραγράφους. Άλλες βιβλιοθήκες που δεν αναφέρονται εδώ μπορούν να βρεθούν στο παράρτημα ή με περισσότερες λεπτομέρειες στη βιβλιογραφία [3].

2.2.1 DOLFIN

Το DOLFIN [4] είναι μια βιβλιοθήκη C++/Python που αποτελεί τη κυρία διεπαφή χρήστη στο FEniCS. Ένα μεγάλο μέρος της λειτουργικότητας του FEniCS βασίζεται στο DOLFIN καθώς παρέχει ένα περιβάλλον επίλυσης για μοντέλα που βασίζονται σε μερικές διαφορικές εξισώσεις και υλοποιεί βασικές λειτουργίες όπως διαχείριση δομών δεδομένων και αλγορίθμων για υπολογιστικά πλέγματα και διακριτοποίηση πεπερασμένων στοιχείων. Το DOLFIN παρέχει κλάσεις για την αναπαράσταση πινάκων, διανυσμάτων, συναρτησιακών χώρων και πλεγμάτων. Επίσης περιλαμβάνει κάποιες ελεύθερες συναρτήσεις ως εναλλακτική επιλογή για τη κλήση επιλυτών. Άλλες λειτουργίες βασίζονται στην επικοινωνία με εξωτερικές βιβλιοθήκες όπως πακέτα γραμμικής άλγεβρας (PETSc[6], Trilinos[7] κ.α) και επιμερισμού πλέγματος (ParMETIS[8], SCOTCH[9]). Ολόκληρη η επικοινωνία μεταξύ ενός προγράμματος από το χρήστη, άλλες βιβλιοθήκες του FEniCS και εξωτερικό λογισμικό υλοποιείται μέσω συνδετικών στρωμάτων (wrapper layers) που αποτελούν μέρος του DOLFIN (εικόνα 2.2).



Εικόνα 2.2 Το DOLFIN αποτελεί τη κύρια διεπαφή FEniCS - χρήστη και διαχειρίζεται την επικοινωνία μεταξύ των κυριότερων βιβλιοθηκών και εξωτερικού λογισμικού.



Εικόνα 2.3 Σχηματική αναπαράσταση μερικών εκ των σημαντικότερων περιεχομένων και κλάσεων του DOLFIN. Τα βέλη υποδηλώνουν εξάρτηση.

2.2.2 UFL

Η γλώσσα ενοποιημένης μορφής UFL (Unified Form Language) είναι μια γλώσσα ειδικής μορφής για διακριτοποίηση μεταβολικών και συναρτησιακών μορφών με τη μέθοδο πεπερασμένων στοιχείων. Πιο συγκεκριμένα η γλώσσα παρέχει μια ευέλικτη διεπαφή στο χρήστη για τον ορισμό χώρων πεπερασμένων στοιχείων και εκφράσεων σε μεταβολική μορφή κοντά στο μαθηματικό συμβολισμό. Η UFL μοιάζει με μια γλώσσα συμβολικού προγραμματισμού αλλά οι στόχοι και οι προτεραιότητές της διαφέρουν από λογισμικά συμβολικού προγραμματισμού γενικής μορφής. Για παράδειγμα η UFL δεν είναι κατάλληλη για μεγάλης κλίμακας συμβολικό προγραμματισμό.

Η UFL παρέχει εργαλεία για:

- Την έκφραση σύνθετων γραμμικών ή μη γραμμικών μεταβολικών μορφών
- Την αυτόματη παραγώγιση εκφράσεων και μεταβολικών μορφών
- Μαθηματικές συναρτήσεις
- Αλγεβρικούς τελεστές
- Δημιουργία τελεστών και συναρτήσεων ως python functions από το χρήστη

Η UFL είναι σχεδιασμένη να αναγνωρίζει εκφράσεις στη παρακάτω γενική μορφή:

$$a(v;w) = \sum_{k=1}^{n_c} \int_{\Omega_k} I_k^c\,(v;w)dx + \sum_{k=1}^{n_e} \int_{\partial\Omega_k} I_k^e\,(v;w)ds + \sum_{k=1}^{n_i} \int_{\Gamma_k} I_k^i\,(v;w)dS$$

όπου $v = [u_1, u_2, \dots u_r]$ ένα διάνυσμα που περιέχει όλες τις άγνωστες συναρτήσεις ενώ

$w = [w_1, w_2, \dots w_r]$ ένα διάνυσμα που περιέχει όλα τα Test Functions.

Τα ολοκληρώματα ορίζονται πολλαπλασιάζοντας μια έκφραση με ένα διαφορικό μέγεθος που αναφέρεται σε ολοκλήρωση :

- Στο εσωτερικό του χωρίου Ω_k (dx, cell integral)
- Στο σύνορο του χωρίου $\partial\Omega_k$ (ds, exterior facet integral)
- Στα εσωτερικά σύνορα των στοιχείων Γ_k (dS, interior facet integral)

2.2.2 FFC

Ο μεταγλωττιστής FFC (FEniCS Form Compiler) έχει σχεδιαστεί να παράγει κώδικα από την συμβολική έκφραση των μεταβολικών μορφών της UFL. Η επίλυση μεταβολικών προβλημάτων πεπερασμένων στοιχείων βασίζεται σε 2 διαδικασίες: την εξαγωγή του αλγεβρικού συστήματος από τις διακριτοποιημένες εξισώσεις και την επίλυσή του. Ως αποτέλεσμα αυτού οι περισσότεροι κώδικες πεπερασμένων στοιχείων είναι παρόμοιοι στο σχεδιασμό τους. Ένα κεντρικό κομμάτι σε αυτούς τους κώδικες είναι η εξαγωγή πινάκων σε αραιή μορφή από τις μεταβολικές εξισώσεις. Η μορφή αυτών των πινάκων καθώς και η αντιστοίχιση από τοπικές σε ολικές συντεταγμένες διαφέρει ανάλογα με τις μεταβολικές μορφές και την επιλογή του είδους των πεπερασμένων στοιχείων. Προκειμένου αυτή η διαδικασία να είναι γενική και αυτοματοποιημένη χρησιμοποιείται ένας form compiler όπως ο FFC. Ο FFC λαμβάνει ως είσοδο μια μεταβολική έκφραση σε γλώσσα UFL και τη μετατρέπει σε κώδικα C++ σε UFC format. Περισσότερες πληροφορίες για τη βιβλιοθήκη UFC μπορούν να βρεθούν στο παράρτημα Γ .

Εικόνα 2.4 Ο FFC παράγει κώδικα C++ σε UFC format με είσοδο μια μεταβολική έκφραση σε γλώσσα UFL

Κεφάλαιο 3

Επίλυση γραμμικών προβλημάτων στο FEniCS

Σε αυτό το κεφάλαιο γίνεται μια εισαγωγή στην επίλυση προβλημάτων στο FEniCS. Περιγράφεται η διαδικασία επίλυσης ενός απλού γραμμικού προβλήματος από την έκφραση σε μεταβολική μορφή μέχρι την εφαρμογή στο FEniCS.

Μια πρώτη γνωριμία με το λογισμικό στη παρούσα εργασία γίνεται επιλύοντας ένα απλό δι-διάστατο γραμμικό πρόβλημα κατανομής θερμοκρασίας με αγωγή σε πτερύγιο. Το FEniCS έχει το πλεονέκτημα ότι ο υπολογιστικός κώδικας σε γλώσσα python/C++ παραμένει αρκετά συμπαγής, σύντομος και κοντά στο μαθηματικό πρόβλημα. Στη περίπτωση ενός γραμμικού προβλήματος ο χρήστης πρέπει να ορίσει τη γεωμετρία και τον αριθμό των στοιχείων του πλέγματος, τις συνοριακές συνθήκες Dirichlet, το πρόβλημα σε μεταβολική μορφή και να καλέσει ένα από τους διαθέσιμους γραμμικούς επιλύτες. Η κλήση του επιλύτη μπορεί να γίνει είτε με την ελεύθερη συνάρτηση solve που προσφέρει το DOLFIN είτε με τη κλάση LinearVariationalSolver. Στη δεύτερη περίπτωση παρέχονται περισσότερες δυνατότητες για τον έλεγχο των παραμέτρων της σύγκλισης και στην επιλογή των γραμμικών επιλυτών και προσταθεροποιητών (preconditioners). Ανάλογα με την επιλογή του πακέτου γραμμικής άλγεβρας που θα χρησιμοποιηθεί υπάρχει αντίστοιχα και ένα πλήθος από διαθέσιμους επιλύτες και προσταθεροποιητές. Περισσότερες πληροφορίες μπορούν να βρεθούν στο παράρτημα Α.

3.1 Κατανομή θερμοκρασίας σε πτερύγιο σε μόνιμη κατάσταση

Η κατανομή θερμοκρασίας σε μόνιμες συνθήκες σε πτερύγιο περιγράφεται από την εξίσωση Laplace. Στη γενική περίπτωση το πρόβλημα συνοριακών τιμών περιλαμβάνει συνοριακές συνθήκες Dirichlet (κάποια επιφάνεια σταθερής θερμοκρασίας), συνοριακές συνθήκες Neumann (μόνωση) και μεικτές συνοριακές συνθήκες Robin (συναγωγή με το περιβάλλον). Για ένα δι-διάστατο χωρίο Ω με απλή γεωμετρία το πρόβλημα συνοριακών τιμών είναι το παρακάτω:

$$\nabla^2 T = 0 \quad (x,y) \in \Omega, \quad \Omega = [0,4]x[0,2] \tag{3.1}$$

$$\frac{\partial T}{\partial x} = 0\,, x = 0 \tag{3.2}$$

$$\frac{\partial T}{\partial y} = -h(T - T_o), y = 2 \tag{3.3}$$

$$T = T_1\,, y = 0\ \kappa\alpha\iota\ x = 4 \tag{3.4}$$

$$T[\text{°C}]\ , T_o = 20\ \text{°C}\ , T_1 = 200\ \text{°C}\,, h = 1\ m^{-1}$$

3.1.1 Μοντελοποίηση σε μεταβολική μορφή

Για την επίλυση στο περιβάλλον του FEniCS απαιτείται ο ορισμός του προβλήματος σε μεταβολική μορφή. Αυτό επιτυγχάνεται πολλαπλασιάζοντας κάθε όρο της εξίσωσης Laplace με μια γνωστή συνάρτηση υ (test function) και ολοκληρώνοντας σε ολόκληρο το χωρίο Ω. Για τυπικούς λόγους συμβολίζεται η άγνωστη συνάρτηση της θερμοκρασίας με u (trial function). Προκύπτει επομένως:

$$\int_{\Omega} (\nabla^2 u)\upsilon dx = 0\ \forall\ \upsilon \in V \tag{3.5}$$

Με εφαρμογή ολοκλήρωσης κατά παράγοντες:

$$-\int_{\Omega} \nabla u \cdot \nabla v \, dx + \int_{\partial\Omega} \frac{\partial u}{\partial n} v \, ds = 0 \Rightarrow$$

$$-\int_{\Omega} \nabla u \cdot \nabla v \, dx + \int_{\Gamma_N} \frac{\partial u}{\partial x} v \, ds - \int_{\Gamma_R} \frac{\partial u}{\partial y} v \, ds = 0 \;\; \forall \, v \in V \tag{3.6}$$

όπου n το κάθετο διάνυσμα στο σύνορο και Γ_N, Γ_R τα σύνορα όπου εφαρμόζονται συνοριακές συνθήκες Neumann και Robin αντίστοιχα.

Τελικά προκύπτει το παρακάτω πρόβλημα:

$$F = -\int_{\Omega} \nabla u \cdot \nabla v \, dx - \int_{\Gamma_R} h(u - T_o) \, v \, ds = 0 \; \forall \, v \in V \tag{3.7}$$

Το FEniCS παρέχει τη δυνατότητα έκφρασης της μεταβολικής μορφής ως μια ενιαία εξίσωση από όπου μπορούν στη συνέχεια να εξαχθούν αυτόματα οι όροι που περιέχουν γνωστές (linear form, $L(v)$) και άγνωστες συναρτήσεις (bilinear form, $a(u, v)$).

3.1.2 Εφαρμογή στο FEniCS

Το πρόβλημα κατανομής θερμοκρασίας μπορεί να επιλυθεί με βάση τον παρακάτω κώδικα σε γλώσσα Python αξιοποιώντας τις βιβλιοθήκες του FEniCS.

Κλήση των βιβλιοθηκών του FEniCS. Ορισμός γεωμετρίας πλέγματος και πλήθους (τετράπλευρων) πεπερασμένων στοιχείων κατά τη x,y διεύθυνση:

Python Code

```
from fenics import *
mesh=RectangleMesh(Point(0,0),Point(4,2),8,4,"right")
```

Δημιουργία συναρτησιακού χώρου με πεπερασμένα στοιχεία τύπου Lagrange και διωνυμικές συναρτήσεις βάσης. Ορισμός trial-test functions:

Python Code

```
V=FunctionSpace(mesh,"Lagrange",2)
u=TrialFunction(V)
v=TestFunction(V)
```

Μαρκάρισμα των υποσυνόρων ορίζοντας υποχωρία. Με αυτό το τρόπο μπορεί να γίνει διάκριση των Γ_N, Γ_R

Python Code

```
# Define boundary subdomains
class BoundaryX0(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], 0, tol)

class BoundaryX1(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[0], 4, tol)

class BoundaryY0(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[1], 0, tol)

class BoundaryY1(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and near(x[1], 2, tol)

# Mark boundaries
boundary_markers = FacetFunction('size_t', mesh)
boundary_markers.set_all(9999)
bx0 = BoundaryX0()
bx1 = BoundaryX1()
by0 = BoundaryY0()
by1 = BoundaryY1()
bx0.mark(boundary_markers, 0)
bx1.mark(boundary_markers, 1)
by0.mark(boundary_markers, 2)
by1.mark(boundary_markers, 3)
```

Εφαρμογή συνοριακών συνθηκών τύπου Dirichlet στα σύνορα $y = 0\ \kappa\alpha\iota\ x = 4$:

Python Code

```
#define Dirichlet Boundary conditions
u_D=Constant(200)
bc1=DirichletBC(V,u_D,boundary_markers,1)
bc2=DirichletBC(V,u_D,boundary_markers,2)
bc=[bc1,bc2]
```

Τροποποίηση του διαφορικού ds ώστε η ολοκλήρωση να πραγματοποιείται στο επιθυμητό υποσύνορο και όχι σε ολόκληρο το σύνορο.

Python Code

```
# Redefine boundary integration measure
ds = Measure('ds', domain=mesh, subdomain_data=boundary_markers)
```

Ορισμός του προβλήματος σε μεταβολική μορφή και αυτόματη εξαγωγή των όρων $a(u,v)$, $L(v)$. Ορισμός της συνάρτησης που θα αποθηκεύει τη λύση:

Python Code

```
h=Constant(1)
To=Constant(20)
F=-dot(grad(u),grad(v))*dx-h*(u-To)*v*ds(3)
a, L=lhs(F), rhs(F)
u = Function(V)
```

Κλήση της συνάρτησης επίλυσης του DOLFIN για τον υπολογισμό της λύσης. Σε αυτή τη περίπτωση χρησιμοποιούνται οι προκαθορισμένοι επιλύτες και παράμετροι σύγκλισης του FEniCS:

Python Code

```
solve(a==L,u,bc)
```

Αποθήκευση της λύσης υπό τη μορφή αρχείου vtk (Visualization Toolkit File format) για τη μετέπειτα επεξεργασία ή ανάγνωση από εξωτερικό λογισμικό:

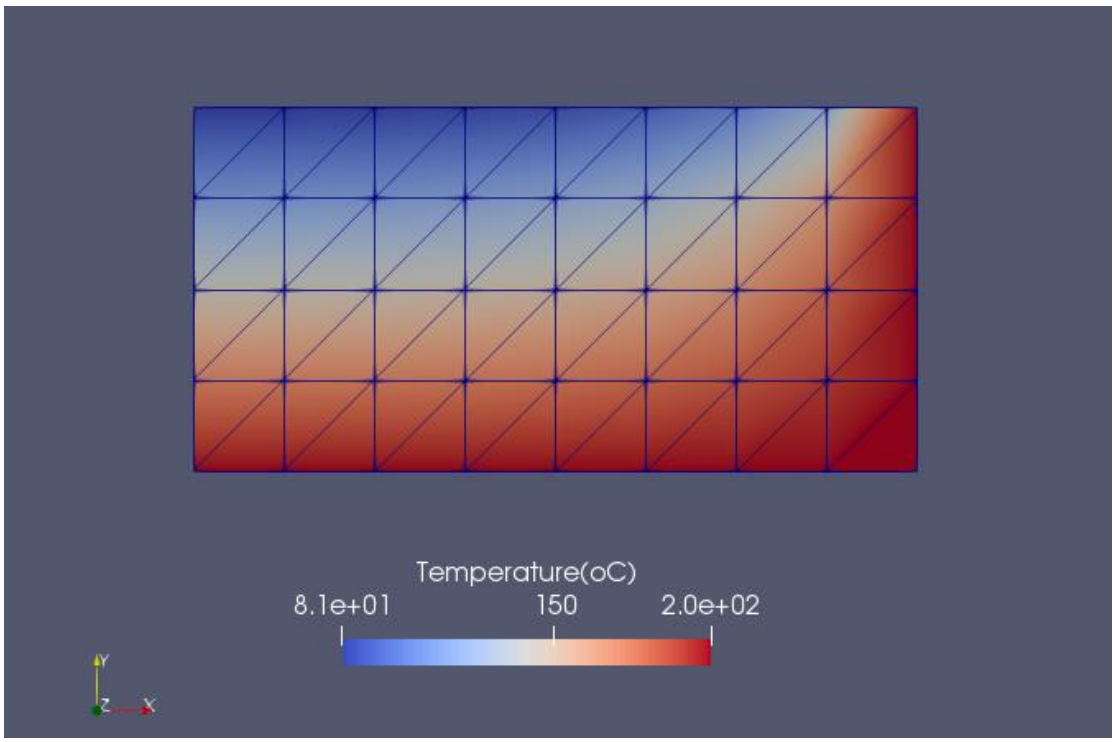
Python Code

```
# Save solution to file in VTK format
vtkfile = File("Temp_distribution.pvd")
vtkfile << u
```

3.1.3 Απεικόνιση της λύσης

Η αναπαράσταση λύσεων από το FEniCS μπορεί να γίνει από κάποιο εξωτερικό λογισμικό όπως το Paraview[10] [5]. Το συγκεκριμένο λογισμικό παρέχει τη δυνατότητα αναπαράστασης τόσο βαθμωτών όσο και διανυσματικών πεδίων. Επιπλέον διαθέτει πολλά εργαλεία για τον έλεγχο της απεικόνισης. Άλλο ένα πλεονέκτημα είναι η δυνατότητα ανάγνωσης πολλαπλών αρχείων που έχουν δημιουργηθεί κατά τη παράλληλη επίλυση και περιέχουν τμήματα της γεωμετρίας και της λύσης.

Η κατανομή της θερμοκρασίας σε μόνιμη κατάσταση όπως προέκυψε από την επίλυση στο FEniCS παρουσιάζεται στην εικόνα 3.1 .



Εικόνα 3.1 Κατανομή θερμοκρασίας στο δι-διάστατο πτερύγιο. Διακρίνεται το πλέγμα που αποτελείται από τριγωνικά στοιχεία.

Κεφάλαιο 4

Δημιουργία πλεγμάτων από εξωτερικό λογισμικό

Σε αυτό το κεφάλαιο περιγράφεται η διαδικασία κατασκευής ενός πλέγματος με το λογισμικό Gmsh και η εισαγωγή του στο FEniCS. Στη συνέχεια γίνεται απλή εφαρμογή στο πρόβλημα κατανομής θερμοκρασίας που αναλύθηκε στο κεφάλαιο 3.

Το FEniCS παρέχει στο χρήστη μεγάλο έλεγχο ως προς την κατάστρωση των μερικών διαφορικών εξισώσεων, την επιλογή προσταθεροποιητών, επιλυτών, υπολογισμό συναρτήσεων (function evaluations) καθώς και μεταεπεξεργασία (post-processing) της λύσης. Ωστόσο οι δυνατότητες του ως προς τη παραγωγή πλεγμάτων περιορίζονται προς το παρόν σε απλές δισδιάστατες και τρισδιάστατες γεωμετρίες (τετράπλευρα, κύβους) ενώ ο έλεγχος της πύκνωσης δεν είναι τόσο εύχρηστος. Γι' αυτό το λόγο όπως αναφέρεται και στην επίσημη βιβλιογραφία [3] είναι προτιμότερο να χρησιμοποιηθεί ένα εξωτερικό λογισμικό για την παραγωγή και τον έλεγχο του πλέγματος.

4.1 Το λογισμικό παραγωγής πλεγμάτων Gmsh

Ένα από τα διαθέσιμα ανοικτά λογισμικά είναι το Gmsh[11] [6] το οποίο μπορεί να παράγει τρισδιάστατα πλέγματα πεπερασμένων στοιχείων παρέχοντας εσωτερικά σχεδιαστικά εργαλεία τύπου CAD διατηρώντας όσο το δυνατόν ένα φιλικό περιβάλλον προς το χρήστη. Τα βήματα που ακολουθούνται για τη κατασκευή ενός απλού πλέγματος στο Gmsh είναι:

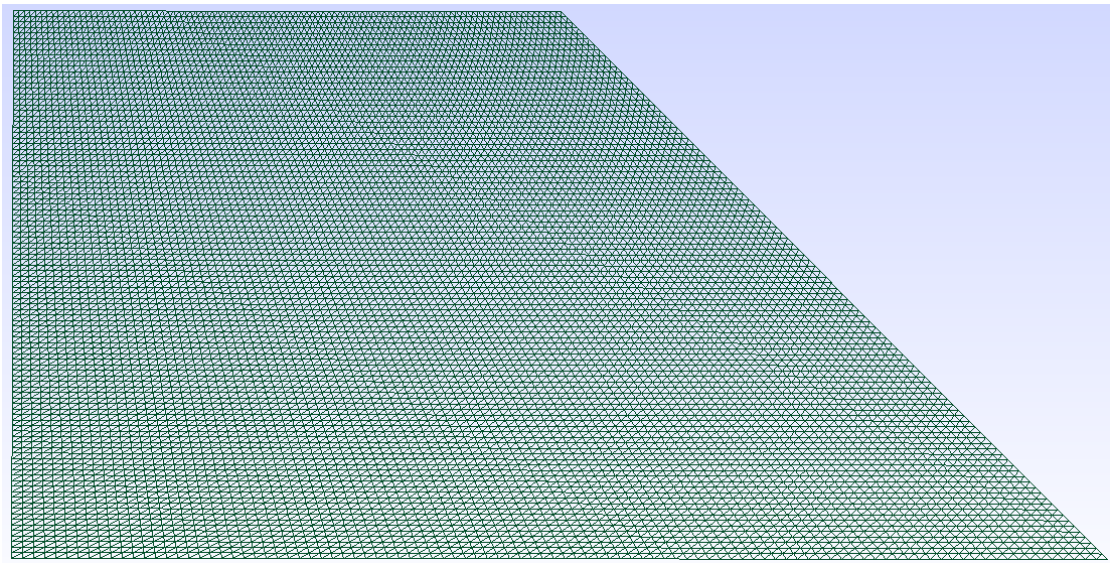
- Καθορίζονται τα σημεία που ορίζουν το σύνορο της γεωμετρίας σε καρτεσιανές συντεταγμένες. Για παράδειγμα σε ένα κύβο καθορίζονται οι συντεταγμένες των ακμών του.
- Κατασκευάζονται τα σύνορα του χωρίου.
- Μαρκάρεται η επιφάνεια που ορίζεται εσωτερικά του συνόρου.
- Ορίζονται τα φυσικά σύνορα όπου θα εφαρμοστούν οι συνοριακές συνθήκες και η φυσική επιφάνεια όπου θα «γεννηθεί» το πλέγμα.
- Δημιουργία πλέγματος.

Το Gmsh εκτός από το γραφικό περιβάλλον παρέχει τη δυνατότητα κατασκευής πλεγμάτων με προγραμματιστικά εργαλεία. Ο χρήστης μπορεί να επεξεργαστεί αρχεία πλεγμάτων και να τροποποιήσει τη γεωμετρία ή τη διαδικασία κατασκευής του πλέγματος. Αυτή είναι μια εξαιρετικά χρήσιμη δυνατότητα για σύνθετες γεωμετρίες ή επαναλαμβανόμενες γεωμετρικές δομές.

4.2 Εφαρμογή στο γραμμικό πρόβλημα κατανομής θερμοκρασίας

Προκειμένου να εξεταστεί η διαδικασία παραγωγής ενός πλέγματος με το Gmsh και της εισαγωγής του στο FEniCS γίνεται εφαρμογή στο πρόβλημα κατανομής θερμοκρασίας που μελετήθηκε παραπάνω στο κεφάλαιο 3.

Η γεωμετρία που δημιουργήθηκε με τη βοήθεια του Gmsh είναι ένα ορθογώνιο τραπέζιο ABCD όπου AB=2 το ύψος του τραπεζίου και BC=2, AD=4 οι δύο βάσεις του. Η διακριτοποίηση του χωρίου έγινε με τριγωνικά στοιχεία κυρίως για λόγους συμφωνίας με τα πλέγματα που δημιουργούνται στο FEniCS (δεν υποστηρίζονται επίσημα μέχρι στιγμής τετράπλευρα πεπερασμένα στοιχεία). Για τη δημιουργία του πλέγματος χρησιμοποιήθηκαν 100 σημεία κατά τη x και 100 σημεία κατά την y διεύθυνση. Επομένως προκύπτουν 99 στοιχεία κατά τη x και 99 στοιχεία κατά την y διεύθυνση. Η μορφή του πλέγματος που κατασκευάστηκε παρουσιάζεται στην εικόνα 4.1 .



Εικόνα 4.1 Το πλέγμα που δημιουργήθηκε στο Gmsh

Το αρχείο του πλέγματος εξάγεται με τη μορφή της επέκτασης -.msh . Έπειτα μετατρέπεται σε αναγνώσιμη από το FEniCS μορφή .xml με τη βοήθεια του DOLFIN:

Bash code: dolphin-convert filename.msh filename.xml

Στη συνέχεια πρέπει να γίνουν κάποιες απαραίτητες τροποποιήσεις στο python script προκειμένου να γίνει εισαγωγή του πλέγματος στο FEniCS και να οριστούν οι συνοριακές συνθήκες στη νέα μη συμμετρική γεωμετρία. Οι αλλαγές παρουσιάζονται παρακάτω.

Εισαγωγή του πλέγματος στο FEniCS. Η κλήση των βιβλιοθηκών του FEniCS μπορεί να γίνει ισοδύναμα καλώντας το DOLFIN:

Python Code

```
from dolfin import*

#import mesh
mesh=Mesh("trapmesh.xml")
```

Αξιοποίηση της δυνατότητας που παρέχει το Gmsh στο μαρκάρισμα συνόρων και υποχωρίων με ακέραιους δείκτες. Η αντιστοίχιση αυτών είναι διαθέσιμη στο χρήστη στο αρχείο του πλέγματος msh. Τα στοιχεία αυτά εξάγονται σε διαφορετικά αρχεία μετά το dolphin-convert:

Python Code

```
#import subdomain and boundary markers
subdomains=MeshFunction("size_t",mesh,"trapmesh_physical_region.xml")
boundaries=MeshFunction("size_t",mesh,"trapmesh_facet_region.xml")
```

Ορισμός συναρτησιακού χώρου και συναρτήσεων:

Python Code

```
V=FunctionSpace(mesh,"Lagrange",2)
u=TrialFunction(V)
v=TestFunction(V)
```

Ορισμός συνοριακών συνθηκών Dirichlet. Αυτό υλοποιείται αρκετά εύκολα καθώς έχει γίνει ήδη μαρκάρισμα των συνόρων με ακέραιους δείκτες:

Python Code

```
u_D=Constant(200)
#define Dirichlet Boundary conditions
bc1=DirichletBC(V,u_D,boundaries,5)
bc2=DirichletBC(V,u_D,boundaries,2)
bc=[bc1,bc2]
```

Και σε αυτή τη περίπτωση επαναορίζεται το διαφορικό ds καθώς ο συνοριακός όρος της μεταφοράς θερμότητας με συναγωγή εφαρμόζεται μόνο σε ένα υποσύνορο όπως και η συνθήκη Neumann:

Python Code

```
# Redefine boundary integration measure
ds = Measure('ds', domain=mesh, subdomain_data=boundaries)
```

Η έκφραση των μεταβολικών μορφών και η επίλυση δε διαφέρει από τη περίπτωση της ενότητας 3.1:

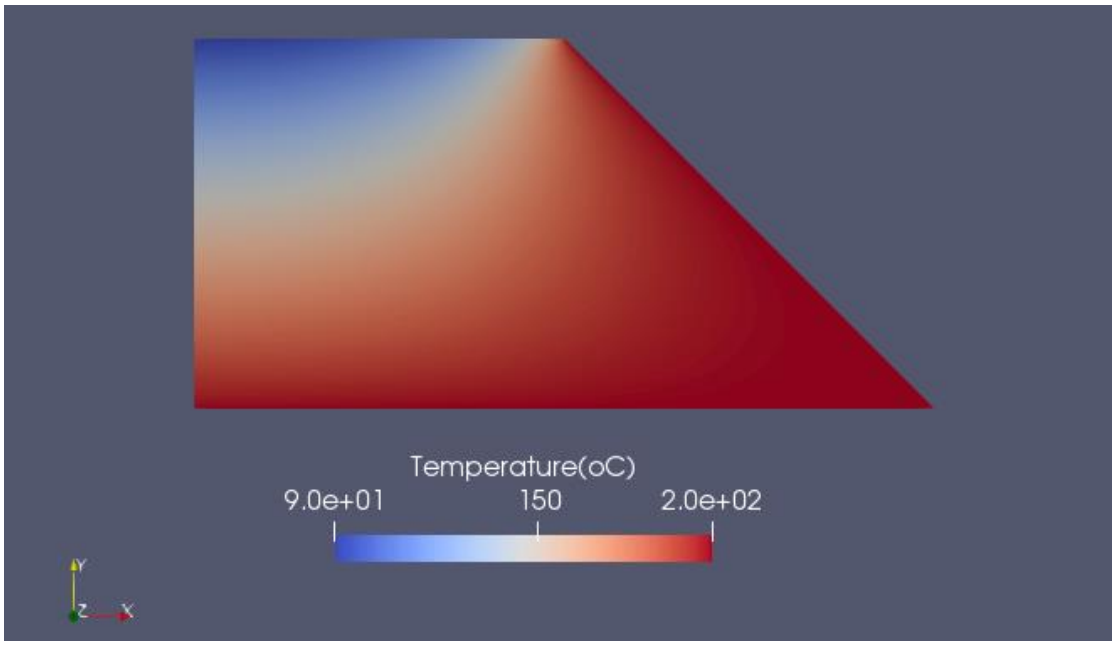
Python Code

```
h=Constant(1)
To=Constant(20)
F=-dot(grad(u),grad(v))*dx-h*(u-To)*v*ds(6)
a, L=lhs(F), rhs(F)
u = Function(V)

solve(a == L,u,bc)

vtkfile = File("Temp2d_gmsh.pvd")
vtkfile << u
```

Η λύση που προκύπτει μπορεί να παρουσιαστεί με τη βοήθεια του Paraview (εικόνα 4.2).



Εικόνα 4.2 Αναπαράσταση της λύσης στη περίπτωση πτερυγίου σχήματος τραπεζίου

Κεφάλαιο 5

Επίλυση μη γραμμικών προβλημάτων στο FEniCS

Στο κεφάλαιο 5 αναπτύσσεται η μέθοδος επίλυσης μη γραμμικών προβλημάτων στο FEniCS με εφαρμογή σε ένα τρι-διάστατο ελλειπτικό πρόβλημα. Επίσης εξετάζεται η σύγκλιση της αυτοματοποιημένης μεθόδου Newton που περιλαμβάνει το λογισμικό.

Ο τρόπος αντιμετώπισης ενός μη γραμμικού προβλήματος στο FEniCS δεν έχει σημαντικές διαφορές σε σχέση με ένα γραμμικό πρόβλημα. Μάλιστα το λογισμικό παρέχει τη δυνατότητα για αυτόματη εξαγωγή της Ιακωβιανής σε μεταβολική μορφή. Επομένως ο χρήστης δεν υποχρεούται να κάνει κάποια άλλη ενέργεια εκτός από το να διαμορφώσει την αρχική εξίσωση (residual) σε μεταβολική μορφή. Από την άλλη πλευρά αν υπάρχει απαίτηση για έλεγχο στο τρόπο υπολογισμού της Ιακωβιανής το FEniCS παρέχει τη κλάση μη γραμμικού επιλύτη NonLinearVariationalSolver. Σε αυτή τη περίπτωση είναι δυνατή η επιλογή των χρησιμοποιούμενων γραμμικών επιλυτών ή παραμέτρων σύγκλισης της μεθόδου Newton. Εάν τίποτα από όλα αυτά δεν καλύπτουν το χρήστη και απαιτείται απόλυτος έλεγχος του επαναληπτικού βρόχου ή είναι η επιθυμητή η χρήση κάποιας άλλης επαναληπτικής μεθόδου (πχ Picard iteration) θα πρέπει να κατασκευαστεί το αντίστοιχο γραμμικό πρόβλημα που θα επιλύεται σε κάθε επανάληψη με τα εργαλεία του FEniCS εντός κάποιου επαναληπτικού βρόχου της python.

5.1 Η μέθοδος Newton

Για την επίλυση του συστήματος μη γραμμικών αλγεβρικών εξισώσεων :

$$\underline{F}(\underline{u}) = \underline{0} \tag{5.1}$$

που προκύπτει από τη διακριτοποίηση των μερικών διαφορικών εξισώσεων χρησιμοποιείται συνήθως η επαναληπτική μέθοδος Newton (ή Newton-Raphson). Στην εξίσωση (5.1) $\underline{F}$ είναι το διάνυσμα μετά τη διακριτοποίηση και $\underline{u}$ οι άγνωστες (διακριτές) τιμές της λύσης. Με δεδομένη μια αρχική προσέγγιση της λύσης $\underline{u}^{(0)}$, υπολογίζεται η (k+1) προσέγγιση $\underline{u}^{(k+1)}$ από την επίλυση του παρακάτω γραμμικού αλγεβρικού συστήματος:

$$\underline{\underline{J}}\,\underline{du}^{(k+1)} = -\underline{F}^{(k)} \tag{5.2}$$

όπου

$$\underline{u}^{(k+1)} = \underline{u}^{(k)} + \underline{du}^{(k+1)} \tag{5.3}$$

Ο πίνακας $\underline{\underline{J}}$ είναι η Ιακωβιανή του συστήματος (5.1) και τα στοιχεία του προκύπτουν με μερική παραγώγιση των στοιχείων του $\underline{F}$ ως προς τα στοιχεία του $\underline{u}$:

$$J_{ij} = \frac{\partial F_i}{\partial u_j} \tag{5.4}$$

5.2 Το τρι-διάστατο ελλειπτικό πρόβλημα συνοριακών τιμών Bratu

Το Bratu είναι ένα ελλειπτικό μη γραμμικό πρόβλημα συνοριακών τιμών με μια παράμετρο λ, σε ένα τρι-διάστατο πεδίο ορισμού (3D) σε καρτεσιανές συντεταγμένες, με ομογενείς οριακές συνθήκες Dirichlet και Neumann. Τέτοιου είδους προβλήματα προκύπτουν από την απλοποίηση του μοντέλου καύσης στερεών καυσίμων και από διάφορες άλλες φυσικές εφαρμογές όπως η θεωρία χημικών αντιδράσεων και η μεταφορά θερμότητας με ακτινοβολία.

Έστω το πρόβλημα Bratu με τις παρακάτω συνοριακές συνθήκες σε ένα τρι-διάστατο χωρίο Ω:

$$\nabla^2 u + \lambda e^u = 0 \quad , (x,y,z) \in \Omega \quad , \quad \Omega = [0,1]x[0,1]x[0,0.5]$$
$$u|_{x=0} = 0 \ , \ u|_{x=1} = 0 \qquad \forall (y,z) \in \Omega$$
$$u|_{y=0} = 0 \ , \ u|_{y=1} = 0 \qquad \forall (x,z) \in \Omega$$
$$\left.\frac{\partial u}{\partial z}\right|_{z=0} = 0 \ , \left.\frac{\partial u}{\partial z}\right|_{z=0.5} = 0 \qquad \forall (x,y) \in \Omega \tag{5.5}$$

Στη συνέχεια διαμορφώνεται κατάλληλα το μαθηματικό πρόβλημα όπως και στη περίπτωση του γραμμικού προβλήματος για την εισαγωγή του στο FEniCS αξιοποιώντας τη βιβλιοθήκη του UFL για την έκφραση των μεταβολικών μορφών.

5.2.1 Μοντελοποίηση σε μεταβολική μορφή

Παρόλο που η μέθοδος Newton κανονικά εφαρμόζεται σε επίπεδο αλγεβρικών εξισώσεων μετά τη διακριτοποίηση όπως αναφέρθηκε παραπάνω, είναι δυνατό να εφαρμοστεί και σε επίπεδο διαφορικών εξισώσεων. Η δεύτερη περίπτωση είναι προτιμότερη για την εφαρμογή της στο FEniCS. Το αποτέλεσμα που προκύπτει σε επίπεδο μεταβολικών μορφών ταυτίζεται με αυτό από την εφαρμογή της μεθόδου σε αλγεβρική μορφή [3, pp. 40-41].

Αρχικά διαμορφώνεται σε μεταβολική μορφή το πρόβλημα:

$$\int_\Omega (\nabla^2 u + \lambda e^u) v dx = 0 \Rightarrow$$
$$-\int_\Omega \nabla u \cdot \nabla v \, dx + \int_{\partial\Omega} \frac{\partial u}{\partial n} v \, ds + \lambda \int_\Omega e^u v \, dx = 0 \ \forall \ v \in V \tag{5.6}$$

Η τεχνική αυτή απαιτεί αρχικά τη γραμμικοποίηση των διαφορικών εξισώσεων πριν τη διακριτοποιησή τους. Με δεδομένη την k προσέγγιση της λύσης $u^{(k)}$ αναζητείται μια διόρθωση δu τέτοια ώστε:

$$u^{(k+1)} = u^{(k)} + \delta u \tag{5.7}$$

Ωστόσο το πρόβλημα παραμένει μη γραμμικό. Υποθέτοντας ότι το δu είναι αρκετά μικρό μπορεί να γραμμικοποιηθεί το πρόβλημα ως προς το δu με προσέγγιση τύπου Taylor. Αντικαθιστώντας την λύση από την εξίσωση (5.7) προκύπτει για το μη γραμμικό όρο:

$$e^{u^{(k+1)}} \cong e^{u^{(k)}} + \delta u \cdot e^{u^{(k)}} \tag{5.8}$$

Αντίστοιχα ο γραμμικός όρος μετατρέπεται στη μορφή:

$$\nabla u^{(k+1)} \cdot \nabla v = \nabla(u^{(k)} + \delta u) \cdot \nabla v = \nabla u^{(k)} \cdot \nabla v + \nabla \delta u \cdot \nabla v \tag{5.9}$$

Η εξίσωση (5.6) λόγω των (5.8), (5.9) ⇒:

$$-\int_{\Omega} \nabla \delta u \nabla v \, dx + \lambda \int_{\Omega} e^{u^{(k)}} \delta u \, v \, dx = \int_{\Omega} \nabla u^{(k)} \nabla v \, dx - \lambda \int_{\Omega} e^{u^{(k)}} v \, dx \; \forall \, v \in V \tag{5.10}$$

$$a(\delta u, v) = -\int_{\Omega} \nabla \delta u \nabla v \, dx + \lambda \int_{\Omega} e^{u^{(k)}} \delta u \, v \, dx \tag{5.11}$$

$$L(v) = \int_{\Omega} \nabla u^{(k)} \nabla v \, dx - \lambda \int_{\Omega} e^{u^{(k)}} v \, dx \tag{5.12}$$

Ο όρος $a(\delta u, v)$ (εξίσωση 5.11) δεν είναι τίποτα άλλο από την Ιακωβιανή του προβλήματος σε μεταβολική μορφή. Σε αυτή τη περίπτωση δε χρειάστηκε να εφαρμοστεί κάποια παραγώγιση. Η Ιακωβιανή προέκυψε έμμεσα με τη γραμμικοποίηση χρησιμοποιώντας τους 2 πρώτους όρους του αναπτύγματος Taylor και αντικαθιστώντας την άγνωστη λύση με τη διόρθωση δu.

Η μέθοδος Newton απαιτεί μια αρχική προσέγγιση της λύσης έστω $u^{(0)}$. Μάλιστα η επιλογή αυτής της προσέγγισης μπορεί να είναι καθοριστική για τη σύγκλιση της μεθόδου ειδικά σε εφαρμογές παραμετρικής ανάλυσης όπου εμφανίζονται ιδιάζοντα σημεία όπως θα εξεταστεί σε επόμενο κεφάλαιο. Γενικά η αρχική προσέγγιση πρέπει να βρίσκεται στη κοντινή γειτονιά της λύσης [7, p. 40] .Μια αποδεκτή αρχική προσέγγιση είναι η λύση του αντίστοιχου γραμμικού προβλήματος δηλαδή για λ=0. Το πρόβλημα που προκύπτει ταυτίζεται με την εξίσωση Laplace.

5.2.2 Εφαρμογή στο FEniCS

Όπως και στη περίπτωση του γραμμικού προβλήματος γίνεται μια παρουσίαση του κώδικα σε γλώσσα python για την επίλυση του προβλήματος (5.5) με το FEniCS.

Κατασκευή πλέγματος και ορισμός συναρτησιακού χώρου:

Python Code

```
from dolfin import*

#Domain omega [0,1]x[0,1]x[0,0.5]
mesh=BoxMesh(Point(0,0,0),Point(1,1,0.5),20,20,20)
V=FunctionSpace(mesh,"Lagrange",2)
```

Ορισμός συνοριακών συνθηκών Dirichlet για το πρόβλημα συνοριακών τιμών. Σε αυτή τη περίπτωση ο χρήστης πρέπει να είναι προσεκτικός. Οι συνοριακές συνθήκες του προβλήματος Newton γενικά δε ταυτίζονται με αυτές του αντίστοιχου προβλήματος συνοριακών τιμών. Εφόσον η λύση επιβάλλεται στα σύνορα Dirichlet απαιτείται η διόρθωση δu να μηδενίζεται στα σύνορα αυτά. Ωστόσο τόσο η συνάρτηση solve όσο και η κλάση του μη γραμμικού επιλύτη δέχεται ως είσοδο τις συνοριακές συνθήκες του κανονικού προβλήματος ώστε να τις εφαρμόσει στην λύση που τροφοδοτείται ως αρχική εκτίμηση. Στη συνέχεια εφαρμόζει μηδενικές συνοριακές συνθήκες Dirichlet στα κατάλληλα σύνορα εντός της επανάληψης Newton για τη διόρθωση δu. Αν η εφαρμογή της μεθόδου δε γίνει αυτόματα θα πρέπει όλη αυτή η διαδικασία να εφαρμοστεί από το χρήστη. Παρόλα αυτά στο συγκεκριμένο πρόβλημα οι συνοριακές συνθήκες Dirichlet είναι εξ αρχής μηδενικές.Επίσης η λύση του αντίστοιχου γραμμικού προβλήματος με αυτές τις συνοριακές συνθήκες είναι η μηδενική επομένως δεν έχει πρακτικό νόημα να επιλυθεί το γραμμικό πρόβλημα.

Python Code

```
#Dirichlet boundary conditions
u_D0=Constant(0)
tol = 1e-14

def boundary_D0(x,on_boundary):
    return on_boundary and (near(x[0],0,tol) or near(x[0],1,tol) \
    or near(x[1],0,tol) or near(x[1],1,tol))

bc0=DirichletBC(V,u_D0,boundary_D0)
```

Ορισμός παραμέτρου λ και συναρτήσεων. Ορισμός του μη γραμμικού εκθετικού όρου ως python function το οποίο μπορεί να χρησιμοποιηθεί στην διατύπωση των μεταβολικών μορφών:

Python Code

```
# parameter
lamda=1.0

u=TrialFunction(V)
v = TestFunction(V)
# most recently computed solution
u_k=Function(V)

# define nonlinear term
def fun(u,lamda):
    return lamda*exp(u)
```

Εισαγωγή της Ιακωβιανής και του υπολοίπου (residual) σε μεταβολική μορφή. Με την εντολή action η άγνωστη λύση u αντικαθίσταται από τη τρέχουσα προσέγγιση της λύσης $u^{(k)}$. Η μεταβολική μορφή της Ιακωβιανής προκύπτει άμεσα παραγωγίζοντας την έκφραση F ως προς την άγνωστη συνάρτηση u στην k επανάληψη. Εναλλακτικά θα μπορούσε να χρησιμοποιηθεί η έκφραση της εξίσωσης (5.11):

Python Code

```
# define nonlinear variational problem

# Residual
F = inner(grad(u),grad(v))*dx-fun(u,lamda)*v*dx
F=action(F, u_k)

# Jacobian
J = derivative(F,u_k,u)
```

Ορισμός του μη γραμμικού προβλήματος με την αντίστοιχη κλάση (NonlinearVariationalProblem) του FEniCS. Δημιουργία μη γραμμικού επιλύτη (NonlinearVariationalSolver). Έλεγχος παραμέτρων σύγκλισης της Newton και επίλυση. Ως επαναληπτικός επιλύτης Krylov των αλγεβρικών συστημάτων που προκύπτουν χρησιμοποιείται ο GMRES [8]. Σε αυτή τη περίπτωση μπορεί να χρησιμοποιηθεί ως επαναληπτική μέθοδος επίλυσης του γραμμικού προβλήματος σε κάθε επανάληψη της Newton και ο conjugate gradient καθώς εφαρμόζονται συμμετρικές συνοριακές συνθήκες Dirichlet .

Python Code

```
#solve nolinear problem
problem = NonlinearVariationalProblem(F, u_k, bc0, J)
solver  = NonlinearVariationalSolver(problem)

#solver parameters
prm = solver.parameters
prm['newton_solver']['absolute_tolerance'] = 1E-8
prm['newton_solver']['relative_tolerance'] = 1E-7
prm['newton_solver']['maximum_iterations'] = 25
prm['newton_solver']['relaxation_parameter'] = 1.0
prm['newton_solver']["convergence_criterion"]='incremental'
solver.parameters["newton_solver"]["linear_solver"] = "gmres"
solver.parameters["newton_solver"]["preconditioner"] ="hypre_euclid"

solver.solve()
```

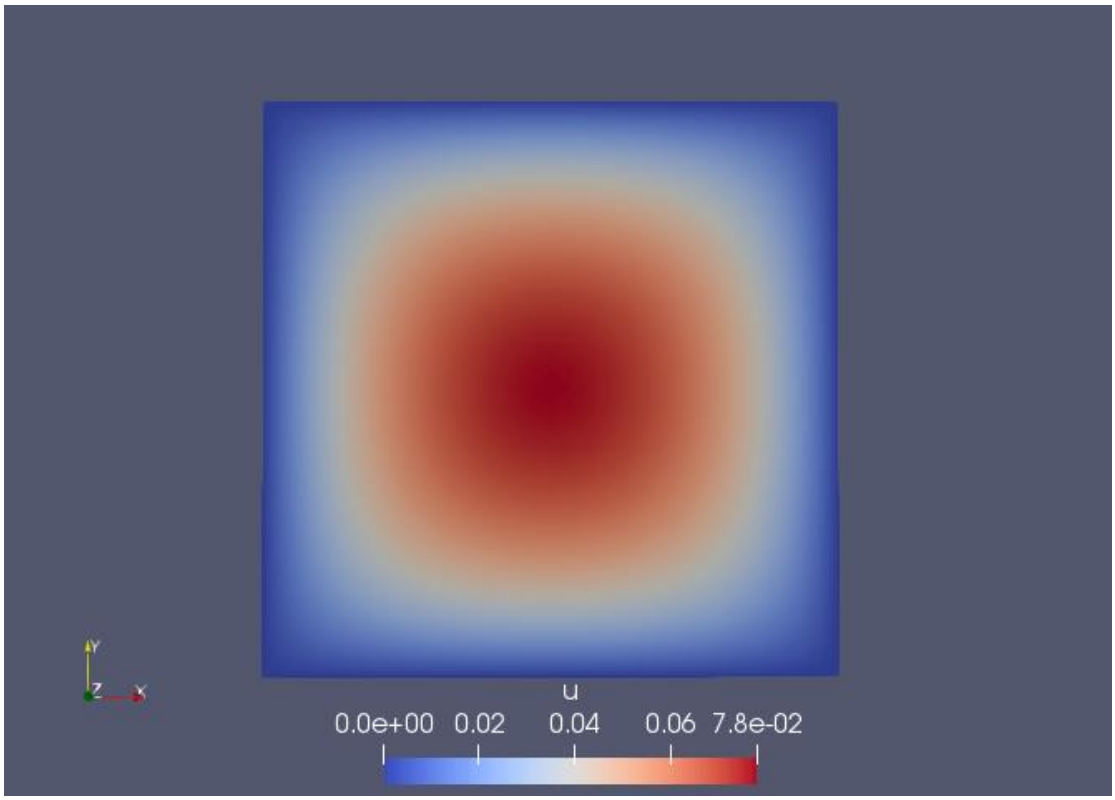
Αποθήκευση λύσης σε μορφή vtk αρχείου:

Python Code

```
# Save solution to file in VTK format
vtkfile = File("Bratu_3D.pvd")
vtkfile << u_k
```

5.2.3 Απεικόνιση της λύσης

Παρουσιάζεται η λύση της εξίσωσης Bratu για λ=1 με κώδικα χρώματος σε z-επιφάνεια. Επειδή το πρόβλημα είναι συμμετρικό προκύπτει πανομοιότυπη λύση για οποιαδήποτε τέτοια επιφάνεια.



Εικόνα 5.1 Απεικόνιση της λύσης του προβλήματος Bratu ως z-επιφάνεια.

5.3 Σύγκλιση της αυτοματοποιημένης μεθόδου Newton

Αν και η εφαρμογή της αυτοματοποιημένης μεθόδου Newton σε μορφή μη γραμμικού επιλύτη είναι αρκετά εύχρηστη θα ήταν ενδιαφέρον να εξεταστεί η πορεία σύγκλισής της και να συγκριθεί με την αντίστοιχη της μη αυτοματοποιημένης μεθόδου. Για το σκοπό αυτό καταστρώνεται κώδικας που επιλύει το ίδιο πρόβλημα χωρίς τη χρήση του έτοιμου επιλύτη. Στη συνέχεια παρουσιάζονται μόνο οι τροποποιήσεις στον κώδικα της ενότητας 5.2.

Ορισμός διόρθωσης της λύσης du:

Python Code

```
#solution correction
du=TrialFunction(V)
```

Ορισμός Ιακωβιανής μήτρας και υπολοίπου δεξιού μέλους:

Python Code

```
#jacobian matrix
J=-inner(grad(du),grad(v))*dx+fun(u_k,lamda)*du*v*dx
#residual
F=inner(grad(u_k),grad(v))*dx-fun(u_k,lamda)*v*dx
du=Function(V)
u=Function(V)
```

Ορισμός παραμέτρων μεθόδου Newton:

Python Code

```
#Newton's Method parameters
omega = 1.0       # relaxation parameter
eps = 1.0
tol = 1.0E-8
k = 0
maxiter = 20
```

Επαναληπτικός βρόγχος Newton. Σε κάθε επανάληψη λύνεται το αντίστοιχο γραμμικό πρόβλημα και διορθώνεται η τρέχουσα εκτίμηση της λύσης. Ως κριτήριο σύγκλισης χρησιμοποιείται η ευκλείδεια νόρμα της διόρθωσης du (L-2 norm) η οποία απαιτείται να λάβει μια τιμή μικρότερη της ακρίβειας tol.

Python Code

```
while eps > tol and k < maxiter:
   k += 1
   #solve linear problem at k iteration
   solve(J == F,du,bc0,solver_parameters={'linear_solver':'gmres',
'preconditioner':'hypre_euclid'})

#calculate L-2 norm
   eps = norm(du.vector(),'l2')
   print 'L-2 Norm:', eps
   #print 'Inf norm',norm(du.vector(),'linf')

   #solution update
   u.vector()[:] = u_k.vector() + omega*du.vector()
   u_k.assign(u)
```

Στη περίπτωση του αυτοματοποιημένου μη γραμμικού επιλύτη της ενότητας (5.2) το κριτήριο σύγκλισης μπορεί να αναφέρεται είτε στη διόρθωση du είτε στα υπόλοιπο F. Η επιλογή γίνεται με βάση τη παράμετρο:

Python Code

```
prm['newton_solver']["convergence_criterion"]='incremental'
```

Η πορεία σύγκλισης όπως ήταν αναμενόμενο ταυτίζεται και για τους δύο τρόπους εφαρμογής της μεθόδου. Παρατηρείται μόνο μια διαφορά στο format κατά τη εκτύπωση του σφάλματος η οποία εξαρτάται από τη μέθοδο print.

Επανάληψη	L-2 Norm , automated Newton	L-2 Norm , manual Newton
0	1.113e+01	1.1128e+01
1	1.801e-02	1.8010e-02
2	6.305e-08	6.3053e-08

Πίνακας 5.1 Πορεία σύγκλισης των 2 τρόπων εφαρμογής της μεθόδου Newton

Η σύγκλιση είναι πολύ γρήγορη καθώς το σφάλμα μέσα σε 3 επαναλήψεις μειώνεται κατά 8 τάξεις μεγέθους. Επομένως η εφαρμογή της μεθόδου Newton έχει γίνει σωστά.

Η παράμετρος λ καθορίζει σημαντικά την ταχύτητα σύγκλισης. Όπως θα αναλυθεί σε επόμενο κεφάλαιο το πρόβλημα Bratu πέρα από μια κρίσιμη τιμή της παραμέτρου ($\lambda \cong 6.8$) δεν έχει λύση. Όσο προσεγγίζεται η κρίσιμη τιμή τόσο αυξάνονται οι απαιτούμενες επαναλήψεις και η σύγκλιση από τετραγωνική μπορεί να γίνει ακόμα και γραμμική. Για τιμές μεγαλύτερης της κρίσιμης η μέθοδος αποκλίνει όπως είναι αναμενόμενο.

Κεφάλαιο 6

Υπολογιστικές απαιτήσεις κατά την επίλυση

Σε αυτό το κεφάλαιο εξετάζονται οι υπολογιστικές απαιτήσεις/επιδόσεις σε χρόνο εκτέλεσης και δέσμευσης μνήμης κατά την επίλυση ενός προβλήματος στο FEniCS σειριακά και παράλληλα. Συγκρίνονται οι απαιτήσεις μεταξύ του γραμμικού προβλήματος κατανομής θερμοκρασίας και του μη γραμμικού προβλήματος Bratu.

Τα περισσότερα προβλήματα πεπερασμένων στοιχείων που προκύπτουν από τη διακριτοποίηση ρεαλιστικών προβλημάτων μηχανικής είναι μεγάλης κλίμακας. Για αυτό το λόγο είναι απαραίτητη η εφαρμογή μεθόδων παράλληλης επεξεργασίας για την εκμετάλλευση των υπολογιστικών συστημάτων με πολλούς επεξεργαστές. Στη παρούσα εργασία χρησιμοποιούνται οι δυνατότητες του FEniCS στην παραλληλοποίηση και παράλληλη εκτέλεση προβλημάτων και γίνεται μια αξιολόγηση της απόδοσής του.

6.1 Το υπολογιστικό σύστημα tyranistar

Η εκτέλεση του FEniCS στη παρούσα εργασία γίνεται ως Docker[12] image. Το Docker είναι εγκατεστημένο σε υπολογιστικό σύστημα με λειτουργικό Linux[13] το οποίο παρέχεται από το υπολογιστικό κέντρο της Σχολής Χημικών Μηχανικών Ε.Μ.Π. Το υπολογιστικό σύστημα tyranistar έχει αρχιτεκτονική κοινής μνήμης μη ομοιόμορφης προσπέλασης με πολλαπλές μονάδες επεξεργαστών (non uniform memory access multiprocessor system[14,15]). Συγκεκριμένα περιλαμβάνει 4 μονάδες επεξεργαστών (sockets) AMD Opteron[16](tm) 6380 (2.5 GHZ, 2 threads/core, 8 cores/socket), συνολικά 64 CPUs αρχιτεκτονικής x86-64 bit και συνολική μνήμη RAM 528 GB (517 GB διαθέσιμα). Η πρόσβαση στο tyranistar γίνεται απομακρυσμένα από προσωπικό υπολογιστή με την υπηρεσία SSH[17] (Secure Shell Client) χρησιμοποιώντας VPN[18] του Ε.Μ.Π.

6.2 Παράλληλη επίλυση στο FEniCS

Το DOLFIN υποστηρίζει παράλληλη επεξεργασία από συστήματα με πολλαπλούς επεξεργαστές μέχρι παράλληλους υπερυπολογιστές. Είναι σχεδιασμένο με τέτοιο τρόπο ώστε οι χρήστες να μπορούν να εκτελούν παράλληλους υπολογισμούς χρησιμοποιώντας τον ίδιο κώδικα που έχουν κατασκευάσει για σειριακούς υπολογισμούς. Υποστηρίζονται 2 υλοποιήσεις, μια για multithreading σε αρχιτεκτονικές κοινής μνήμης και μία για αρχιτεκτονικές κατανεμημένης μνήμης. Και στις 2 περιπτώσεις απαιτείται κάποια προεπεξεργασία του πλέγματος. Για παραλληλοποίηση τύπου multithreading ακολουθείται μια διαδικασία ‘’χρωματισμού’’ του πλέγματος έτσι ώστε να αναγνωρίζονται επικαλυπτόμενα κελιά. Για παραλληλοποίηση κατανεμημένης μνήμης χρησιμοποιείται μια τεχνική επιμερισμού του πλέγματος (mesh partitioning). Το multithreading υποστηρίζεται στο FEniCS από το OpenMP[19] αλλά οι δυνατότητες του είναι περιορισμένες καθώς μπορεί να χρησιμοποιηθεί μόνο για την επιτάχυνση της διαδικασίας εξαγωγής του αλγεβρικού συστήματος και όχι κατά τη διάρκεια της επίλυσής του. Αντιθέτως η παράλληλη επεξεργασία σε αρχιτεκτονική κατανεμημένης μνήμης υποστηρίζεται εξ ολοκλήρου και υλοποιείται με το MPI[20](Message Passing Interface). Η παράλληλη επεξεργασία με το MPI μπορεί να εφαρμοστεί αποτελεσματικά ωστόσο και σε αρχιτεκτονικές κοινής μνήμης μεταξύ μεμονωμένων πυρήνων ή κόμβων NUMA ως μια εναλλακτική λύση του multithreading. Το FEniCS δεν υποστηρίζει ακόμα υβριδικό παράλληλο προγραμματισμό που θα μπορούσε να εκμεταλλευτεί πλήρως την αρχιτεκτονική των νέων υπολογιστικών συστημάτων κοινής μνήμης. Το MPI στο FEniCS θα πρέπει να συνδυαστεί με ένα πακέτο γραμμικής άλγεβρας που επιτρέπει τη παράλληλη επεξεργασία (PETSc, Trilinos) και ένα πακέτο λογισμικού για τον επιμερισμό του πλέγματος (SCOTCH, ParMETIS). Αυτά τα πακέτα όπως αναφέρθηκε και στην ενότητα 2.2 είναι προσβάσιμα στο χρήστη με την επιλογή κατάλληλων παραμέτρων του DOLFIN.

Οι μόνες αλλαγές που θα πρέπει να γίνουν από το χρήστη ώστε να εκτελεστεί παράλληλα μια εφαρμογή στο FEniCS με το MPI είναι:

- Εκτέλεση του DOLFIN με το MPI

Bash code: mpirun –n number_of_processes python filename.py

- Επιλογή κατάλληλου πακέτου γραμμικής άλγεβρας. Δεν είναι απαραίτητη κάποια ενέργεια αν ο χρήστης θέλει να χρησιμοποιήσει το PETSc που είναι το προκαθορισμένο πακέτο που χρησιμοποιεί το FEniCS.

Επίσης ένας άμεσος επιλύτης μειώνει σημαντικά την απόδοση κατά την παράλληλη επεξεργασία.Είναι προτιμότερο επομένως να χρησιμοποιηθεί ένας επαναληπτικός επιλύτης σε συνδυασμό με κατάλληλο προσταθεροποιητή (preconditioner) για την επίλυση των αλγεβρικών προβλημάτων.Ο προσταθεροποιητής είναι ένας πίνακας που μετασχηματίζει κατάλληλα τον πίνακα που προκύπτει από τη διακριτοποίηση μειώνοντας τον αριθμό κατάστασής του προκειμένου να συγκλίνει γρηγορότερα ο επαναληπτικός επιλύτης.

6.3 Παράλληλη επίλυση του προβλήματος κατανομής θερμοκρασίας

Στα παρακάτω αποσπάσματα παρουσιάζονται μερικές αλλαγές στο python script της ενότητας 3.1 για την επίλυση του γραμμικού προβλήματος κατανομής θερμοκρασίας παράλληλα.

Επιλογή του κατάλληλου επαναληπτικού επιλύτη και προσταθεροποιητή. Σε αυτή τη περίπτωση χρησιμοποιείται η μέθοδος GMRES [8] καθώς εφαρμόζονται μη συμμετρικές συνοριακές συνθήκες Dirichlet (ο πίνακας δεν είναι θετικά ορισμένος). Ως προσταθεροποιητής επιλέγεται ο hypre_euclid (parallel ILU decomposition):

Python Code

```
solve(a == L,u,bc,solver_parameters={'linear_solver':'gmres',
'preconditioner':'hypre_euclid'})
```

Παρουσίαση μέσου υπολογιστικού χρόνου για κάθε διαδικασία (διακριτοποίηση πλέγματος, επίλυση γραμμικού συστήματος κλπ) σε όλες τις παράλληλες διεργασίες:

Python Code

```
list_timings(TimingClear_keep,[TimingType_wall,TimingType_system])
```

6.3.1 Παράλληλη επιτάχυνση

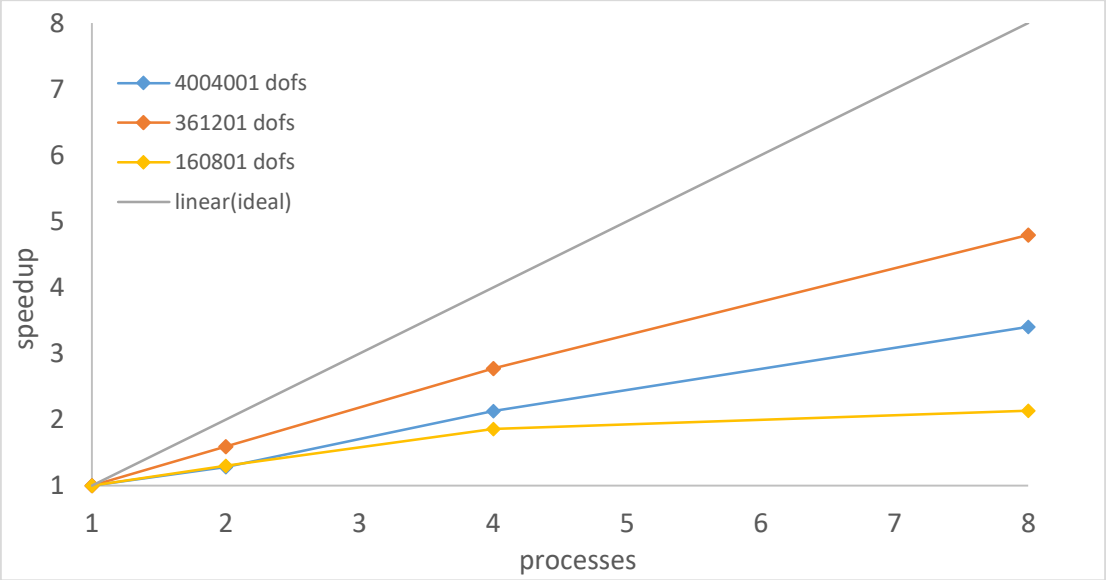
Ένα από τα σημαντικότερα μεγέθη που περιγράφουν τις παράλληλες επιδόσεις ενός προβλήματος ως προς τον υπολογιστικό χρόνο είναι η παράλληλη επιτάχυνση.

Ως παράλληλη επιτάχυνση S συνήθως ορίζεται ο λόγος του υπολογιστικού χρόνου ενός σειριακού αλγορίθμου T_S, που ιδανικά είναι βελτιστοποιημένος, προς τον υπολογιστικό χρόνο επίλυσης του ίδιου προβλήματος T_P, παράλληλα σε p επεξεργαστές [9, p. 12] :

$$S = \frac{T_S}{T_P} \tag{6.1}$$

Αυτός ο ορισμός αντιστοιχεί σε περιπτώσεις όπου το πρόβλημα δεν είναι υπερβολικά μεγάλο σε μέγεθος και μπορεί να εκτελεστεί σειριακά σε ένα επεξεργαστή.

Στο διάγραμμα 6.1 παρουσιάζεται η παράλληλη επιτάχυνση συναρτήσει του αριθμού των διεργασιών και του πλήθους των βαθμών ελευθερίας.



Διάγραμμα 6.1 Παράλληλη επιτάχυνση για το γραμμικό πρόβλημα

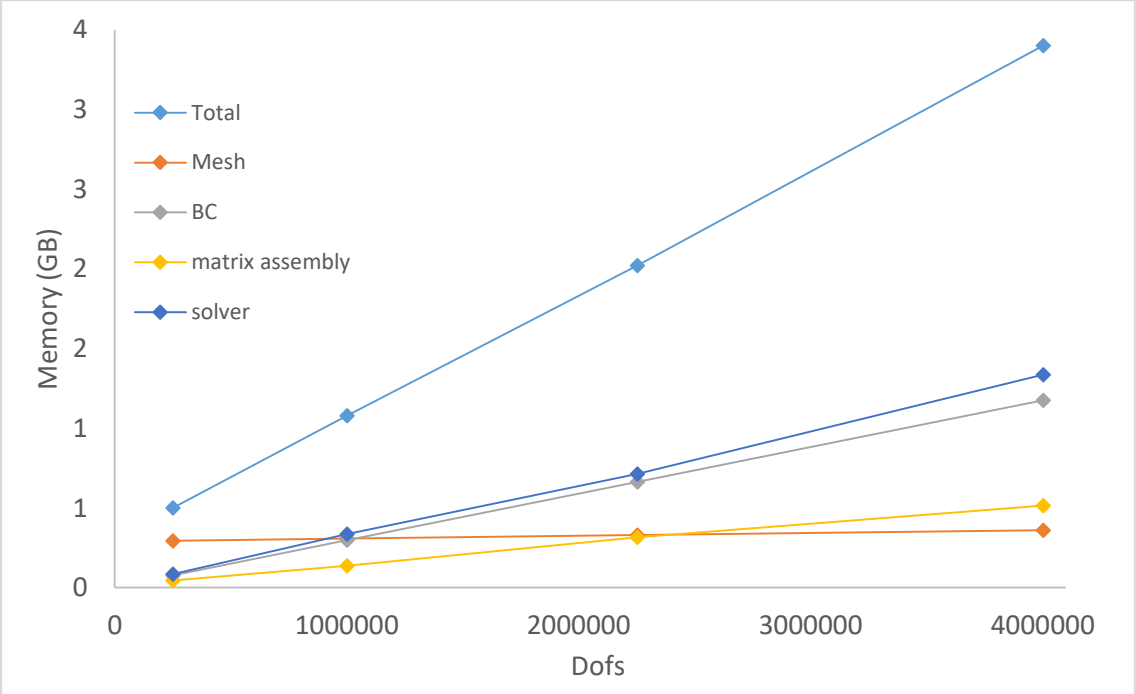
Ανάλογα με το πλήθος των βαθμών ελευθερίας (ισοδύναμα με το πλήθος των κόμβων) προκύπτει και διαφορετική παράλληλη επιτάχυνση. Για μικρό πλήθος βαθμών ελευθερίας η παράλληλη επίλυση δεν επιφέρει σημαντική μείωση στον υπολογιστικό χρόνο. Αυτό είναι αναμενόμενο και οφείλεται κατά κύριο λόγο στην επικοινωνία μεταξύ των επεξεργαστών διαμέσω του διαύλου επικοινωνίας και τη διαμέριση του πλέγματος. Στη συνέχεια παρατηρείται μια περιοχή όπου η παράλληλη επιτάχυνση είναι μέγιστη ($3 \cdot 10^5 - 5 \cdot 10^5$ Dofs) και σχεδόν γραμμική. Περαιτέρω αύξηση στο μέγεθος του συστήματος επιφέρει εκ νέου μείωση της επιτάχυνσης. Ιδανικά η παράλληλη επιτάχυνση θα αυξανόταν γραμμικά p φορές για p διεργασίες (και αντίστοιχο αριθμό πυρήνων) κάτι που είναι γνωστό ως strong scaling. Ωστόσο ο νόμος του Amdahl προβλέπει ότι η επιτάχυνση δεν αυξάνεται γραμμικά αλλά συγκλίνει ασυμπτωτικά σε μια σταθερά 1/α όπου α το κλάσμα του προβλήματος που εκτελείται σειριακά. Στη πράξη αντί να επιδιώκεται η μείωση του υπολογιστικού χρόνου εξετάζεται η επίλυση N φορές μεγαλύτερων προβλημάτων σε N πυρήνες για σχετικά σταθερό χρόνο επίλυσης[21] (weak scaling) [9, pp. 14-16]. Στη περίπτωση του FEniCS προτείνεται η κατανομή 500000 βαθμών ελευθερίας (Dofs) ανά πυρήνα που χρησιμοποιείται. Κάτι τέτοιο εξαρτάται από τον επιλύτη και τον προσταθεροποιητή που θα χρησιμοποιηθεί. Η ανατροπή της παράλληλης επιτάχυνσης που παρατηρείται και η οποία δε συμφωνεί με τις προβλέψεις του weak scaling οφείλεται κυρίως στον προσταθεροποιητή καθώς αυξάνοντας το μέγεθος το προβλήματος δε παρατηρείται κάποια μείωση στον υπολογιστικό χρόνο κατά την κλήση του επιλύτη παρά την αξιοποίηση περισσότερων πυρήνων.

Τα παραπάνω συμπεράσματα δεν αναιρούν τη χρησιμότητα των προσταθεροποιητών στους επαναληπτικούς επιλύτες όπου μειώνουν δραστικά τον υπολογιστικό χρόνο επίλυσης του προβλήματος. Η αύξηση της παράλληλης επιτάχυνσης δεν είναι πάντοτε το ζητούμενο και θα πρέπει να συνυπολογίζεται και ο χρόνος επίλυσης.

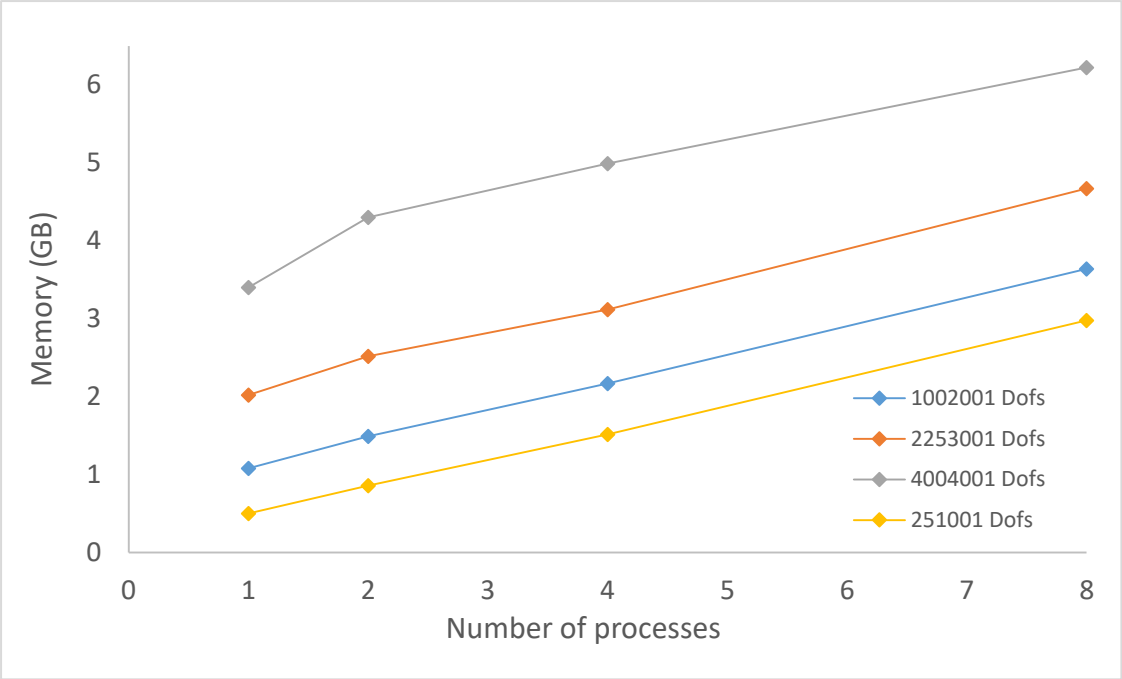
Κατά τη σειριακή επίλυση ο περισσότερος υπολογιστικός χρόνος αναλώνεται στην επίλυση του γραμμικού συστήματος και στην αντιστοίχιση των βαθμών ελευθερίας (λύσεις στους κόμβους) με τους κόμβους του πλέγματος από τοπική σε ολική αρίθμηση. Κατά τη παράλληλη επίλυση σημαντικό χρόνο καταναλώνει η επίλυση του συστήματος και η διαμέριση του πλέγματος. Τόσο η επίλυση όσο και η διαμέριση του πλέγματος γίνονται από εξωτερικές βιβλιοθήκες όπου ο χρήστης δεν έχει άμεση πρόσβαση. Επομένως οι προσπάθειες βελτίωσης της παράλληλης επιτάχυνσης σε ένα γραμμικό πρόβλημα περιορίζονται αρχικά στη κατάλληλη επιλογή επαναληπτικού επιλύτη και προσταθεροποιητή.

6.3.2 Δέσμευση μνήμης

Εκτός από τον υπολογιστικό χρόνο κατά τη σειριακή και παράλληλη επίλυση σημαντικό ρόλο παίζει επίσης η δέσμευση της μνήμης. Μάλιστα σε αρκετά μεγάλα προβλήματα με εκατομμύρια βαθμούς ελευθερίας (Dofs) όπως τα τρισδιάστατα μη γραμμικά προβλήματα η διαθέσιμη μνήμη ενός απλού προσωπικού υπολογιστή συνήθως δεν είναι επαρκής κάτι που καθιστά την επίλυση αδύνατη. Εξετάστηκαν αναλυτικά στο γραμμικό πρόβλημα της κατανομής θερμοκρασίας οι μέγιστες απαιτήσεις σε μνήμη. Επίσης εξετάστηκε κατά την παράλληλη επίλυση πόσο αυξάνεται η δέσμευση συναρτήσει του αριθμού των διεργασιών που καλούνται μέσω του MPI. Τα αποτελέσματα παρουσιάζονται στα διαγράμματα 6.2 και 6.3 .



Εικόνα 6.2 Μέγιστη δέσμευση μνήμης στο γραμμικό πρόβλημα από τις διάφορες διαδικασίες συναρτήσει του πλήθους των βαθμών ελευθερίας



Εικόνα 6.3 Μέγιστη δέσμευση μνήμης συναρτήσει του πλήθους των παράλληλων διεργασιών για διάφορα μεγέθη προβλήματος

Ο υπολογισμός της μέγιστης δέσμευσης εικονικής (Virtual) μνήμης γίνεται σχετικά έυκολα σε λειτουργικά συστήματα Linux με χρήση της συνάρτησης memory_usage του DOLFIN. Εικονική είναι η μνήμη RAM που δεσμεύεται από το πρόγραμμα και είναι άμεσα διαθέσιμη χωρίς αυτό να σημαίνει απαραίτητα ότι χρησιμοποιείται εξ ολοκλήρου.

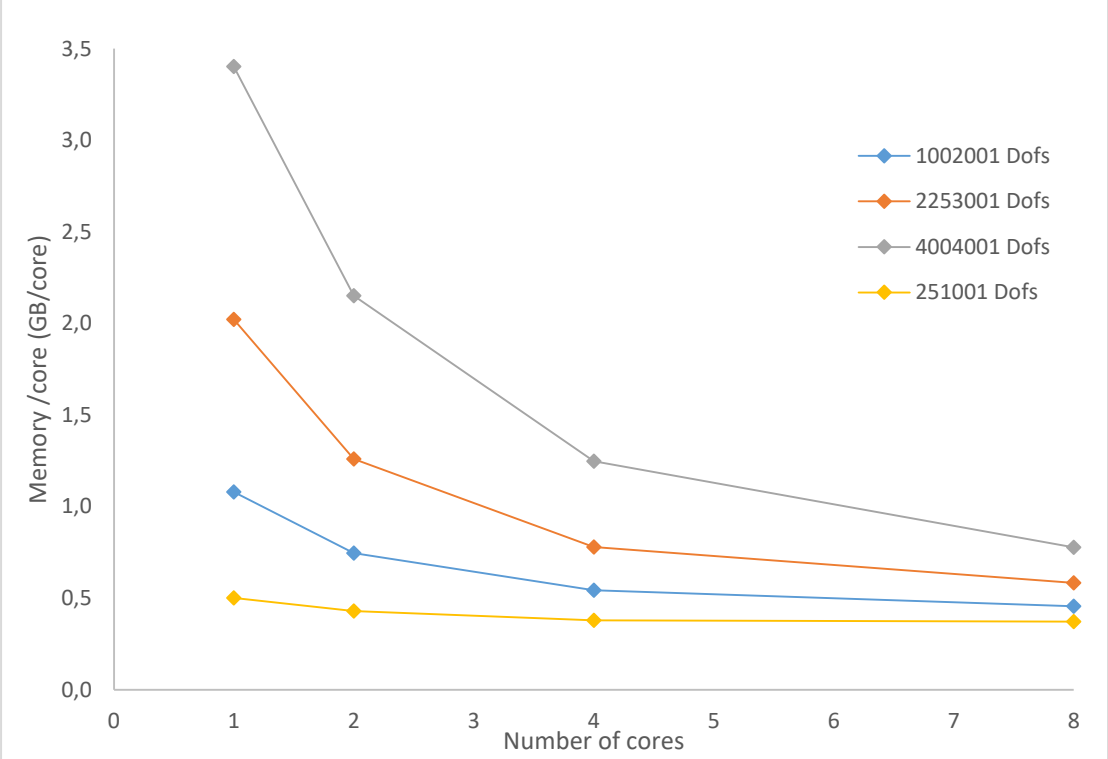
Python Code

```
print dolfin.common.memory.memory_usage(as_string=True)
```

Η συνολική δέσμευση εικονικής μνήμης είναι ανάλογη του πλήθους των κόμβων (ή βαθμών ελευθερίας) και σχεδόν διπλασιάζεται για διπλάσιο αριθμό κόμβων. Η μνήμη που απαιτείται για τη δημιουργία του πλέγματος παραμένει πρακτικά σταθερή σε σχέση με το μέγεθος του προβλήματος. Σύμφωνα με τους κατασκευαστές του λογισμικού δεν αναμένεται σημαντική μεταβολή στη δέσμευση μνήμης από το πλέγμα ακόμα και για προβλήματα με δισεκατομμύριους κόμβους. Αντίστοιχα η δημιουργία των πινάκων (matrix assembly) από τις εξισώσεις διακριτοποίησης καταναλώνει μικρό ποσόστο της μνήμης καθώς τα συστήματα που προκύπτουν είναι πολύ αραιά και απαιτείται να αποθηκευτεί πολύ μικρότερη πληροφορία για την αναπαράσταση του συστήματος. Η αναπαράσταση αραιών πινάκων στο FEniCS είναι μια διαδικασία που διαχειρίζεται το πακέτο γραμμικής άλγεβρας. Το PETSc για παράδειγμα χρησιμοποιεί το CSR format[22].
Η περισσότερη μνήμη δεσμεύεται κυρίως:

- Κατά το μαρκάρισμα των συνορών και την εφαρμογή των συνοριακών συνθηκών Dirichlet
- Κατά την κλήση του γραμμικού επιλύτη και την επίλυση του συστήματος

Κατά τη παράλληλη επίλυση παρατηρείται αύξηση της δεσμευόμενης μνήμης όσο αυξάνει το πλήθος των διεργασιών (και ισοδύναμα το πλήθος των επεξεργαστών). Ιδανικά η μνήμη που δεσμεύεται συνολικά από όλες τις διεργασίες θα ήταν ίση με τη μνήμη που δεσμεύεται κατά τη σειριακή επίλυση. Ωστόσο η διαδικασία επιμερισμού του πλέγματος (mesh partitioning) δημιουργεί επικαλυπτόμενες περιοχές μεταξύ των τμημάτων του πλέγματος που πρέπει να αποθηκευτούν εκ νέου για όλες τις διεργασίες. Επιπλέον δεσμεύεται μνήμη για την επικοινωνία μεταξύ των διεργασιών με το MPI[23]. Κάτι τέτοιο μπορεί να αποτελέσει πρόβλημα μόνο για υπολογιστικά συστήματα με αρχιτεκτονική κοινής μνήμης. Σε μια υπολογιστική συστοιχία κατανεμημένης μνήμης κάθε υπολογιστικός κόμβος διαθέτει ξεχωριστές μονάδες μνήμης. Επομένως η αύξηση του πλήθους των χρησιμοποιούμενων κόμβων ουσιαστικά διαμοιράζει την απαιτούμενη μνήμη σε όλους τους κόμβους. Αυτό το συμπέρασμα είναι εμφανές όταν εξετάζεται η απαιτούμενη μνήμη ανά το πλήθος των χρησιμοποιούμενων πυρήνων όπως στο διάγραμμα 6.4.

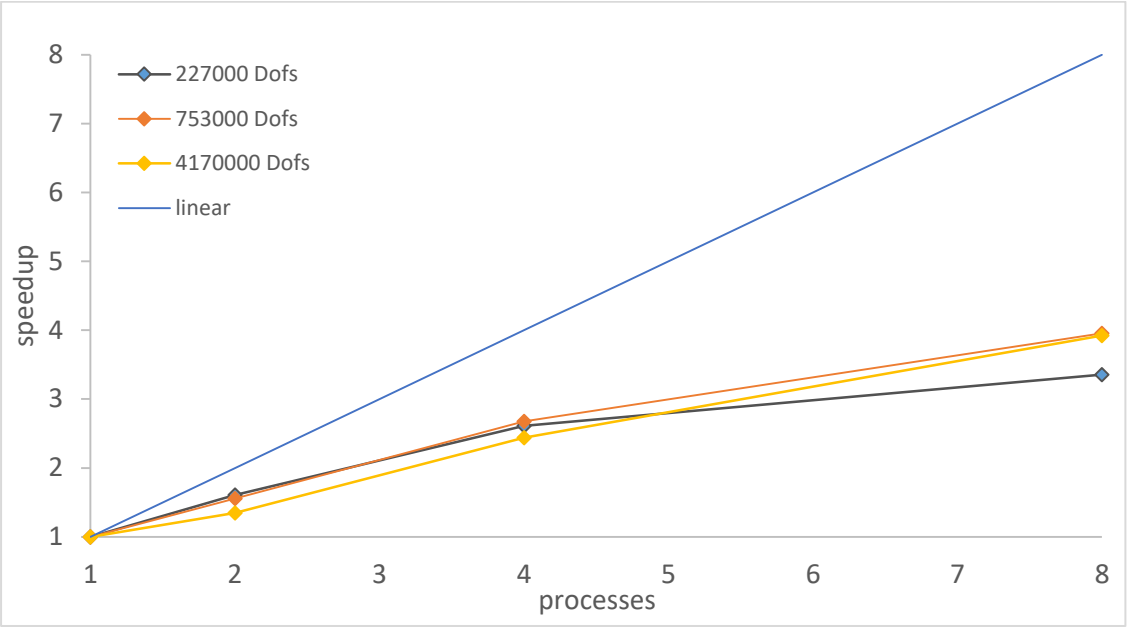


Διάγραμμα 6.4 Δέσμευση μνήμης ανά το πλήθος των χρησιμοποιούμενων πυρήνων

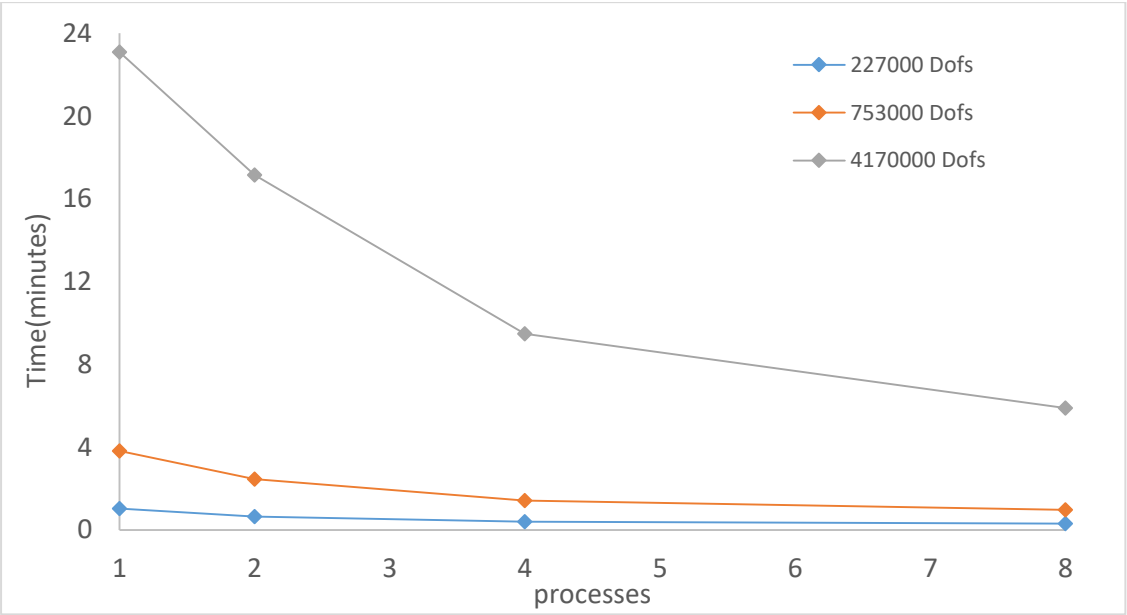
6.4 Παράλληλη επίλυση του προβλήματος Bratu

Όπως και στη περίπτωση του δι-διάστατου γραμμικού προβλήματος εξετάζονται οι απαιτήσεις σε υπολογιστικό χρόνο και δέσμευση μνήμης κατά την παράλληλη επίλυση του προβλήματος Bratu. Επειδή το πρόβλημα είναι μη γραμμικό και τρι-διάστατο αναμένεται να αυξηθούν οι απαιτήσεις αυτές συγκριτικά με το γραμμικό πρόβλημα ειδικά κατά τη πύκνωση του πλέγματος. Ωστόσο δε παρατηρήθηκε αδυναμία σειριακής επίλυσης ακόμα και για το πιο πυκνό πλέγμα που χρησιμοποιήθηκε. Επομένως μπορεί να οριστεί κανονικά η παράλληλη επιτάχυνση όπως και στην ενότητα 6.3 ως ένα μέτρο αξιολόγησης της υπολογιστικής απόδοσης κατά τη παράλληλη επεξεργασία.

6.4.1 Παράλληλη επιτάχυνση



Εικόνα 6.5 Παράλληλη επιτάχυνση συναρτήσει του πλήθος των διεργασιών για το πρόβλημα 3D Bratu.



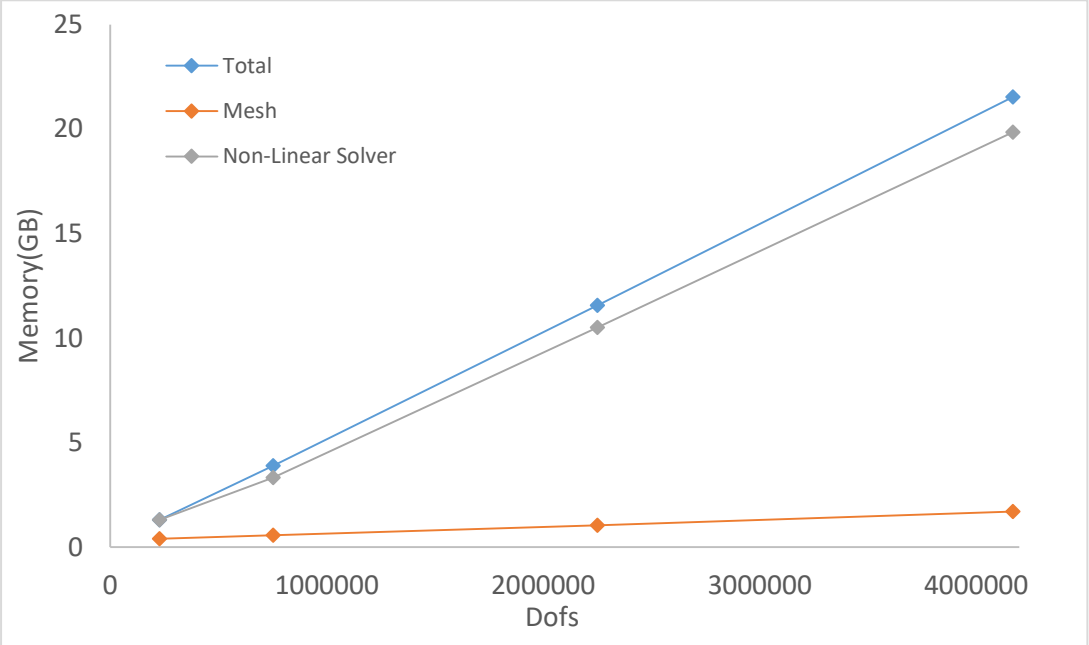
Διάγραμμα 6.6 Χρόνοι εκτέλεσης συναρτήσει του αριθμού των διεργασιών

Σύμφωνα με το διάγραμμα 6.6 οι χρόνοι εκτέλεσης αυξάνουν σημαντικά σε σχέση με το γραμμικό πρόβλημα (έως και 24 min για $4 \cdot 10^6$ κόμβους με σειριακή επίλυση). Για $4 \cdot 10^6$ κόμβους επιτυγχάνεται έως και 75 % μείωση στον υπολογιστικό χρόνο του προβλήματος όταν αυτό εκτελείται σε 8 πυρήνες. Τα αποτελέσματα αυτά ισχύουν για τιμές της παραμέτρου μακριά από το σημείο στροφής (συγκεκριμένα για λ=1) όπου η Newton συγκλίνει τετραγωνικά. Επειδή στο μη γραμμικό πρόβλημα απαιτούνται πολύ περισσότερα assembly στο σύστημα και αντίστοιχα πολλαπλές κλήσεις του επαναληπτικού επιλύτη Krylov

η χρήση του προσταθεροποιητή δεν επιδρά όπως στη περίπτωση του γραμμικού προβλήματος. Γι αυτό το λόγο όπως προβλέπεται από το weak scaling είναι προτιμότερο να αυξάνεται ο αριθμός των πυρήνων όσο αυξάνεται το μέγεθος του προβλήματος. Το συμπέρασμα αυτό συμφωνεί και με το διάγραμμα 6.5. Καθώς αυξάνεται το μέγεθος του προβλήματος η παράλληλη επιτάχυνση βελτιώνεται με αύξηση του πλήθους των διεργασιών. Φυσικά η χρήση προσταθεροποιητή περιορίζει τη περαιτέρω βελτίωση όταν το μέγεθος του προβλήματος ξεπεράσει τους μερικούς εκατομμύριους κόμβους (σε αντίθεση με το γραμμικό πρόβλημα που αυτή η ανατροπή εμφανιζόταν για σαφώς μικρότερο μέγεθος).

6.4.2 Δέσμευση μνήμης

Η μέγιστη δέσμευση μνήμης αυξάνει γραμμικά όσο αυξάνεται το μέγεθος του προβλήματος και σε αντίθεση με το γραμμικό πρόβλημα είναι σημαντική (έως και 22 GB για $4{\cdot}10^6$ κόμβους). Ο μη γραμμικός επιλύτης που περιλαμβάνει το assembly, την εφαρμογή των συνοριακών συνθηκών Dirichlet και την επίλυση του γραμμικού συστήματος εντός των επαναλήψεων της Newton δεσμεύει το μεγαλύτερο ποσοστό μνήμης. Η δημιουργία του πλέγματος δεν επηρεάζει σημαντικά τη δέσμευση μνήμης και παραμένει σχετικά σταθερή όσο αυξάνει το μέγεθος του προβλήματος. Τα παραπάνω συμπεράσματα συνοψίζονται στο διάγραμμα 6.7.



Εικόνα 6.7 Δέσμευση μνήμης συναρτήσει του αριθμού των βαθμών ελευθερίας (κόμβων)

Επομένως όπως αναμενόταν οι υπολογιστικές απαιτήσεις για την επίλυση ενός τρι-διάστατου μη γραμμικού προβλήματος με ικανοποιητικά πυκνό πλέγμα απαιτούν παράλληλη επεξεργασία σε κάποια υπολογιστικό σύστημα με επαρκή διαθέσιμη μνήμη όπως και γίνεται στη παρούσα εργασία. Φυσικά αυτό δεν εξαρτάται τόσο από το FEniCS αλλά από την ίδια τη φύση των αριθμητικών μεθόδων που εφαρμόζονται σε μη γραμμικά προβλήματα. Οι απαιτήσεις αυτές μπορεί να αυξηθούν με την ύπαρξη ιδιάζοντων σημείων. Με βάση τα παραπάνω αποτελέσματα προκύπτει το συμπέρασμα ότι παρόλο τη διαφορετική φύση των προβλημάτων το FEniCS μπορεί να διαχειριστεί με επιτυχία την παράλληλη

επεξεργασία και να επιταχύνει σημαντικά τους υπολογισμούς χωρίς ο χρήστης να είναι απαραίτητο να κάνει οποιαδήποτε αλλαγή στον αρχικό σειριακό κώδικα. Δηλαδή το λογισμικό συνδυάζει υπολογιστική απόδοση με ευκολία χρήσης.

Κεφάλαιο 7

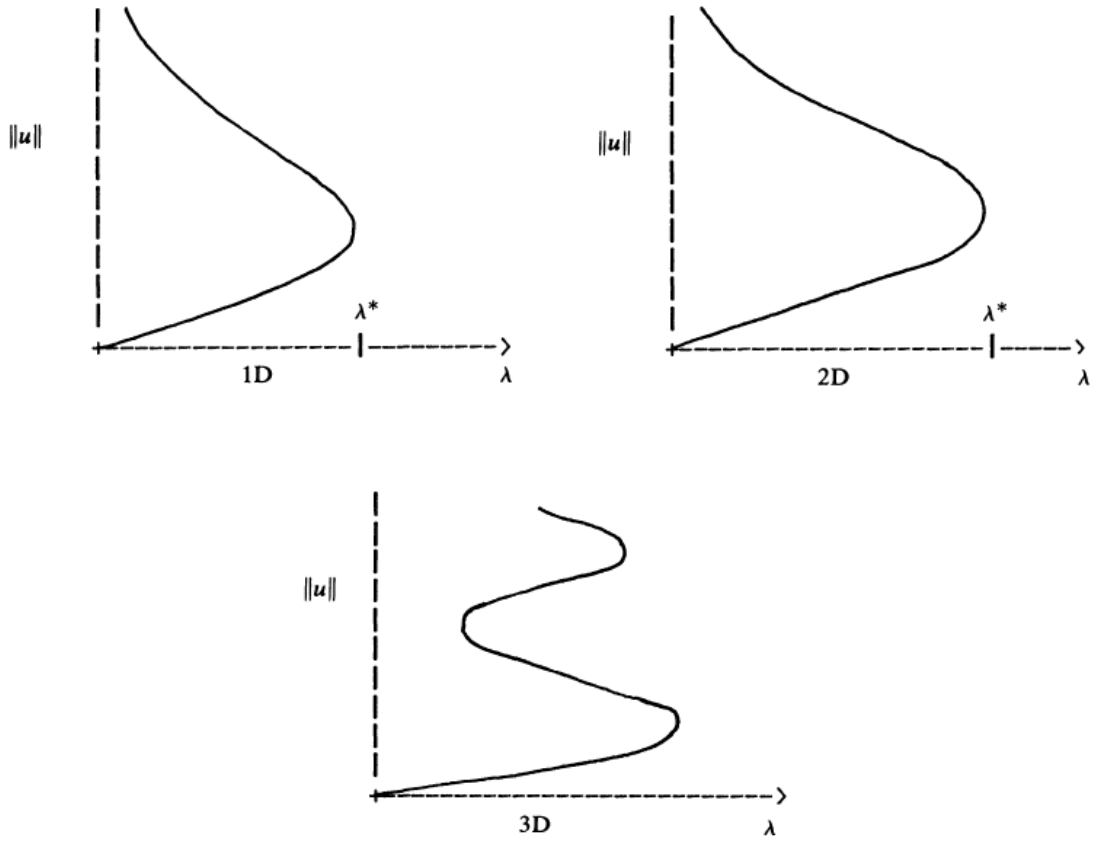
Εφαρμογή μεθόδων παραμετρικού βηματισμού στο FEniCS

Σε αυτό το κεφάλαιο παρουσιάζεται η εφαρμογή διάφορων μεθόδων παραμετρικής ανάλυσης του χώρου των λύσεων μη γραμμικών προβλημάτων στο FEniCS. Κυριότερος στόχος είναι η εφαρμογή των μεθόδων με ένα φιλικό τρόπο προς το λογισμικό προκειμένου να διατηρηθεί η αποτελεσματικότητα και η κομψότητα των υπολογισμών κατά την παράλληλη επεξεργασία.

Γενικά ο χώρος των λύσεων που προκύπτει από την επίλυση μη γραμμικών διαφορικών εξισώσεων μπορεί να είναι ιδιαίτερα περίπλοκος. Η ύπαρξη λύσης, πολλαπλότητας λύσεων και η ευστάθεια της λύσης εξαρτάται από τις τιμές των παραμέτρων. Μάλιστα σε πολλά προβλήματα μηχανικής η ανάλυση του χώρου των λύσεων είναι σημαντικότερη και από τη λύση καθεαυτή [7, p. 44]. Στόχος αυτού του κεφαλαίου είναι να εφαρμοστούν μέθοδοι παραμετρικής ανάλυσης σε μη γραμμικά προβλήματα χρησιμοποιώντας το FEniCS. Το λογισμικό προς το παρόν δεν υποστηρίζει συστηματικά την εφαρμογή μεθόδων παραμετρικού βηματισμού. Μάλιστα δεν έχει γίνει κάποια επίσημη δημοσίευση παρόλο που αρκετοί χρήστες έχουν ενδιαφερθεί για το ζήτημα και έχουν καταθέσει ερωτήσεις στη κοινότητα του FEniCS.

7.1 Ανάλυση του χώρου των λύσεων του προβλήματος Bratu

Το πρόβλημα Bratu υπό τη μορφή της ενότητας 5.1 είναι μονοπαραμετρικό. Δεν εμφανίζονται σημεία διακλάδωσης λύσεων (bifurcation points) παρά μόνο σημεία στροφής (turning points). Για 1 ή 2 διαστάσεις το πρόβλημα εμφανίζει ένα μοναδικό σημείο στροφής. Στη περίπτωση του τρι-διάστατου προβλήματος μπορούν να εμφανίζονται έως και άπειρα τέτοια σημεία και ισοδύναμα άπειρες λύσεις για μια συγκεκριμένη τιμή της παραμέτρου λ (σε σφαιρική γεωμετρία).



Εικόνα 7.1 Μορφή της λύσης του προβλήματος Bratu σε 1, 2 ή 3 διαστάσεις. Το τρι-διάστατο πρόβλημα μπορεί να εμφανίζει πολλά σημεία στροφής.

Το εξεταζόμενο πρόβλημα τίθεται στον μοναδιαίο κύβο $[0,1]x[0,1]x[0,1] \in R^3$. Παρουσιάζεται τουλάχιστον ένα σημείο στροφής για $\lambda = \lambda^* \cong 6.81$. Για τιμές της παραμέτρου $\lambda > \lambda^* \cong 6.81$ το πρόβλημα δεν έχει λύση ενώ για $\lambda < \lambda^* \cong 6.81$ έχει τουλάχιστον 2 λύσεις [10].

Η τιμή της νόρμας της λύσης $\|u\|_\infty$ για $\lambda = \lambda^* \cong 6.81$ είναι $\|u\|_\infty \cong 1.39$. Στη συνέχεια θα εφαρμοστούν μέθοδοι στο FEniCS για παραμετρικό βηματισμό τόσο γύρω από ομαλά σημεία (regular points) όσο και για σημεία πέρα από το σημείο στροφής.

7.2 Βηματισμός σε παράμετρο (natural continuation)

Ο βηματισμός σε παράμετρο χρησιμοποίεται για την ιχνηλάτηση περιοχών των κλάδων της λύσης που περιλαμβάνουν μόνο ομαλά σημεία. Η μέθοδος απαιτεί την εκτίμηση της λύσης για μια νέα γειτονική τιμή της παραμέτρου λ+δλ γνωρίζοντας τη λύση u(λ) προκειμένου αυτή να χρησιμοποιηθεί ως αρχική εκτίμηση στη μέθοδο Newton. Για βηματισμό τύπου μηδενικής τάξης (zeroth-order continuation) χρησιμοποιείται η λύση u(λ) ως αρχική εκτίμηση. Ο βηματισμός τύπου πρώτης τάξης μπορεί να προσφέρει μια καλύτερη εκτίμηση για τη νέα λύση λαμβάνοντας υπόψη την ευαισθησία της λύσης ως προς τη παράμετρο λ. Η αρχική εκτίμηση δίνεται υπό την μορφή :

$$u_0(\lambda + \delta\lambda) = u(\lambda) + \frac{\partial u}{\partial \lambda}\delta\lambda \tag{7.1}$$

Η παράγωγος της λύσης ως προς τη παράμετρο λ μπορεί να υπολογιστεί εύκολα μετά τη διακριτοποίηση με την επίλυση του παρακάτω αλγεβρικού συστήματος:

$$J(\lambda)\frac{\partial u}{\partial \lambda} = -\frac{\partial F}{\partial \lambda} \tag{7.2}$$

όπου $J(\lambda)$ είναι η Ιακωβιανή του προβλήματος η οποία είναι διαθέσιμη από την τελευταία επανάληψη Newton. Επομένως θα πρέπει απλά να υπολογιστεί η παράγωγος του υπολοίπου F ως προς τη παράμετρο λ. Επίσης η επίλυση του γραμμικού συστήματος στη περίπτωση του Gauss elimination είναι πολύ απλή καθώς η αποικοδόμηση της Ιακωβιανής είναι ήδη γνωστή από τη τελευταία επανάληψη Newton. Αντίστοιχα για επαναληπτικούς επιλύτες Krylov μπορεί να χρησιμοποιηθεί ο ίδιος προσταθεροποιητής από την τελευταία επανάληψη Newton.

Ο βηματισμός τύπου πρώτης τάξης επιτρέπει να γίνουν μεγαλύτερα βήματα στη παράμετρο λ χωρίς αυτό να οδηγήσει σε απόκλιση της μεθόδου Newton. Με αυτό το τρόπο μειώνεται ο αριθμός των επαναλήψεων και ο υπολογιστικός χρόνος επίλυσης. Ωστόσο θα πρέπει να δίνεται ιδιαίτερη προσοχή στην επιλογή του βήματος δλ όσο προσεγγίζεται το σημείο στροφής. Παρακάτω παρουσιάζεται η εφαρμογή της πρώτης τάξης βηματισμού στο FEniCS για το πρόβλημα Bratu.

7.2.1 Εφαρμογή στο πρόβλημα Bratu

Ορίζεται η γεωμετρία, ο συναρτησιακός χώρος και οι συνοριακές συνθήκες Dirichlet:

Python Code

```
from dolfin import*

#Domain omega [0,1]x[0,1]x[0,0.5]
mesh=BoxMesh(Point(0,0,0),Point(1,1,0.5),30,30,30)
V=FunctionSpace(mesh,"Lagrange",2)

#Define linear variational problem for initial guess (lamda=0)

#Dirichlet boundary conditions
u_D0=Constant(0)
tol = 1e-14

def boundary_D0(x,on_boundary):
    return on_boundary and (near(x[0],0,tol) or near(x[0],1,tol) \
    or near(x[1],0,tol) or near(x[1],1,tol))
bc0=DirichletBC(V,u_D0,boundary_D0)
```

Ορίζονται οι μη γραμμικοί όροι που εμφανίζονται στο υπόλοιπο και στη παράγωγό του ως προς τη παράμετρο και οι συναρτήσεις που αντιστοιχούν στη λύση στη προηγούμενη και τη τρέχουσα επανάληψη Newton. Χρησιμοποιείται η λύση του γραμμικού προβλήματος ως αρχική εκτίμηση της λύσης:

Python Code

```
v = TestFunction(V)

# define nonlinear variational problem
u=TrialFunction(V)

# define nonlinear term for nonlinear problem
def fun(u,lamda):
    return lamda*exp(u)


#define nonlinear term for residual derivative with respect to lamda
def fun2(u):
    return exp(u)


# most recently computed solution
u_k=Function(V)
# assign initial guess from linear problem
u_k.vector()[:]=u0.vector()[:]
```

Ορισμός παραμέτρων Newton και βήματος δλ:

Python Code

```
#Newton's Method parameters
omega = 1.0       # relaxation parameter
eps = 1.0
tol = 1.0E-5
k = 0
maxiter = 20
iterations=10

lamda=0.2
#parameter step
dlamda=(2.2-0.2)/iterations
```

Στη συνέχεια ακολουθεί ο εξωτερικού βρόχος επαναλήψεων για βηματισμό στη παράμετρο λ (paramstep loop) και ο εσωτερικός βρόγχος επαναλήψεων Newton (Newton iterations) για την επίλυση του μη γραμμικού προβλήματος. Επιλέγεται η εφαρμογή της μη αυτοματοποιημένης μεθόδου. Με αυτό το τρόπο μπορεί να χρησιμοποιηθεί η Ιακωβιανή μήτρα από την τελευταία επανάληψη Newton για την εκτίμηση της νέας λύσης u(λ+δλ). Ο επιλύτης ορίζεται απευθείας σαν επιλύτης του πακέτου PETSc προκειμένου να είναι δυνατή η επαναχρησιμοποίηση του προσταθεροποιητή:

Python Code

```
# paramstep loop
for it in range(iterations+1):

  #Newton iterations
   while eps > tol and k < maxiter:
           k += 1
           #solution correction
           du=TrialFunction(V)

           #jacobian matrix
           J=-inner(grad(du),grad(v))*dx+fun(u_k,lamda)*du*v*dx

            #residual
           F=inner(grad(u_k),grad(v))*dx-fun(u_k,lamda)*v*dx

           #assemble and solve linear system
           du=Function(V)
           u=Function(V)
           Jac,R=assemble_system(J,F,bc0)
           dU=du.vector()

           solver1=PETScKrylovSolver("gmres","hypre_euclid")
           solver1.solve(Jac,dU,R)

           #calculate L-2 norm
           eps = norm(du.vector(),'l2')
           #print 'L-2 Norm:', eps
           #print 'Inf norm',norm(du.vector(),'linf')

           #update solution
           u.vector()[:] = u_k.vector() + omega*du.vector()
           u_k.assign(u)
```

Επίλυση του γραμμικού συστήματος με άγνωστο τη παράγωγο της λύσης ως προς τη παράμετρο λ. Διόρθωση της αρχικής εκτίμησης για τη νέα λύση u(λ+δλ) με βάση το βηματισμό πρώτης τάξης. Η επιλογή της επαναχρησιμοποίησης του προσταθεροποιητή πρέπει να απενεργοποιηθεί εφόσον θα ξεκινήσει και πάλι ο βρόγχος Newton:

Python Code

```
   #first order continuation

   #residual derivative with respect to lamda

   Fl=-fun2(u_k)*v*dx

 # solution derivative with respect to lamda
   u_l=Function(V)

   Rl=assemble(Fl)
```

```
   U_l=u_l.vector()

   #reuse preconditioner from last newton iteration
   solver1.set_reuse_preconditioner(True)


   solver1.solve(Jac,U_l,Rl)

   #print ' inf norm u_l',norm(u_l.vector(),'linf')
   #update solution estimate for u(lamda+dlamda)
   u_k.vector()[:] +=dlamda*U_l
   #update lamda

   lamda +=dlamda
   k = 0
   eps=1.0
   solver1.set_reuse_preconditioner(False)


vtkfile = File("Bratu_firstordercont.pvd")
vtkfile << u
```

Προκειμένου να επαληθευτεί ότι η εφαρμογή του βηματισμού 1^{ης} τάξης έχει γίνει σωστά στο FEniCS θα πρέπει να συγκριθεί ο μέσος αριθμός επαναλήψεων Newton που απαιτούνται για σύγκλιση με κάθε σχήμα βηματισμού. Φυσικά στο συγκεκριμένο πρόβλημα εξετάζεται η λύση για ομαλά σημεία και δε παρουσιάζεται αδυναμία σύγκλισης της λύσης για οποιαδήποτε επιλογή αρχικής εκτίμησης. Σε άλλες περιπτώσεις πιθανώς να είναι απαραίτητη η εφαρμογή του σχήματος πρώτης τάξης αν η λύση για προηγούμενη τιμή της παραμέτρου ως αρχική εκτίμηση οδηγεί σε απόκλιση της μεθόδου. Ο υπολογιστικός χρόνος στο σχήμα πρώτης τάξης αναμένεται και αυτός να μειωθεί αλλά δεν εξαρτάται μόνο από τη μείωση των επαναλήψεων Newton αλλά και από τον υπολογισμό της παραγώγου της λύσης ως προς τη παράμετρο. Στο παρακάτω πίνακα 7.1 παρουσιάζονται τα αποτελέσματα συναρτήσει της επιλογής του βήματος Δλ. Η τιμή της παραμέτρου λ μεταβάλλεται μεταξύ των τιμών 0.5-5.

Δλ	1^{st} -order Newton iterations(avg)	0^{th} order Newton iterations(avg)	1^{st} –order timing (sec)	0^{th} order timing (sec)
0.9	3	3.5	75	79
0.45	2.73	3.18	128	130
0.225	2.33	3	213	233

Πίνακας 7.1 Μέσος αριθμός επαναλήψεων Newton και υπολογιστικοί χρόνοι με τα δύο διαφορετικά σχήματα βηματισμού σε παράμετρο.

Ο μέσος αριθμός επαναλήψεων Newton έως σύγκλιση είναι μικρότερος για το σχήμα $1^{ης}$ τάξης όπως ήταν αναμενόμενο καθώς λαμβάνει υπόψη την ευαισθησία της λύσης ως προς τη μεταβολή της παραμέτρου. Καθώς το βήμα Δλ μειώνεται ο μέσος αριθμός επαναλήψεων μείωνεται και για τα δύο σχήματα βηματισμού. Ο υπολογιστικός χρόνος αντίστοιχα είναι μικρότερος για το σχήμα $1^{ης}$ τάξης κάτι που σημαίνει ότι ο υπολογισμός της παραγώγου ως προς τη παράμετρο είναι μια ‘’φθηνή’’ υπολογιστικά διαδικασία σε σχέση με τις επαναλήψεις Newton που περιλαμβάνουν διακριτοποίηση και επίλυση του γραμμικού συστήματος.

7.3 Μέθοδος παραμετροποίησης μήκους τόξου

Η μέθοδο βηματισμού σε παράμετρο που αναλύθηκε παραπάνω πρέπει να τροποποιηθεί κατάλληλα όταν εφαρμόζεται σε μη γραμμικά προβλήματα που παρουσιάζουν πολλαπλότητα λύσεων για μια ή περισσότερες τιμές της παραμέτρου. Μάλιστα η ύπαρξη πολλαπλότητας λύσεων συνοδεύεται από την εμφάνιση μη ομαλών σημείων. Η ύπαρξη τέτοιων σημείων όπως το σημείο στροφής στο πρόβλημα Bratu συνδέεται με εμφάνιση ιδιάζουσας Ιακωβιανής. Μια από πιο κλασικές μεθόδους αντιμετώπισης [7] [10], [11] είναι η παραμετροποίηση των κλάδων της λύσης ως προς το μήκος τόξου ως προς το οποίο η λύση είναι γνησίως μονότονη.

Έστω s η παράμετρος μήκους τόξου. Η λύση u και η παράμετρος λ είναι πλέον συναρτήσεις του s έστω $u(s), \lambda(s)$. Η αρχική μη γραμμική εξίσωση της οποίας αναζητείται η λύση είναι:

$$F\big(u(s), \lambda(s)\big) = 0 \tag{7.3}$$

Το s κατά μήκος του κλάδου της λύσης ορίζεται με βάση την εξίσωση:

$$\left\|\frac{\partial u}{\partial s}\right\|_2^2 + \left(\frac{d\lambda}{ds}\right)^2 - 1 = 0 \tag{7.4}$$

Με γραμμικοποίηση γύρω από μια γνωστή λύση $(u(s_o), \lambda(s_o))$ προκύπτει από τη (7.3):

$$N\big(u(s), \lambda(s)\big) \equiv \langle \frac{\partial u}{\partial s}\Big|_{s_o}, \big(u(s) - u(s_o)\big)\rangle + \big(\lambda(s) - u(s_o)\big)\frac{d\lambda}{ds}\Big|_{s_o} - \Delta s = 0 \tag{7.5}$$

Επομένως το νέο επαυξημένο σύστημα εξισώσεων είναι:

$$F\big(u(s), \lambda(s)\big) = 0$$

$$N\big(u(s),\lambda(s)\big)=0$$

και σε κάθε επανάληψη Newton πρέπει να επιλυθεί το παρακάτω επαυξημένο γραμμικό σύστημα:

$$\begin{bmatrix} F_u & F_\lambda \\ N_u^T & N_\lambda \end{bmatrix}\begin{bmatrix}\delta u \\ \delta\lambda\end{bmatrix}=-\begin{bmatrix}F \\ N\end{bmatrix} \tag{7.6}$$

Η παράγωγος της αρχικής εξίσωσης F_u όταν διακριτοποιηθεί αποτελεί την Ιακωβιανή του αρχικού προβλήματος. Για τους υπόλοιπους όρους της επαυξημένης Ιακωβιανής προκύπτει:

$$N_u=\left.\frac{\partial u}{\partial s}\right|_{s_o} \tag{7.7}$$

$$N_\lambda=\left.\frac{d\lambda}{ds}\right|_{s_o} \tag{7.8}$$

$$\left.\frac{d\lambda}{ds}\right|_{s_o}=\pm\left(1+\left\|\frac{\partial u}{\partial\lambda}\right\|_2^{\,2}\right)^{-\frac{1}{2}} \tag{7.9}$$

$$\left.\frac{\partial u}{\partial s}\right|_{s_o}=\left.\frac{\partial u}{\partial\lambda}\,\frac{d\lambda}{ds}\right|_{s_o} \tag{7.10}$$

Το πρόσημο στην εξίσωση (7.9) καθορίζει αν ο βηματισμός θα γίνει για αυξανόμενες ή μειούμενες τιμές της παραμέτρου λ.

Εναλλακτικά οι παράγωγοι της λύσης και της παραμέτρου ως προς το μήκος τόξου μπορούν να υπολογιστούν με πεπερασμένες διαφορές (backward ή forward finite differences). Σε αυτή τη περίπτωση είναι απαραίτητη η γνώση μιας ακόμα λύσης.

Η παραπάνω μέθοδος ωστόσο περιλαμβάνει δημιουργία επαυξημένων συστημάτων και πράξεις μεταξύ διανυσμάτων. Το FEniCS είναι πολύ αποτελεσματικό όταν το assembly του συστήματος προκύπτει από κάποιο πρόβλημα σε μεταβολική μορφή και όχι απευθείας από τις εξισώσεις διακριτοποίησης. Όπως έχει αναφερθεί και σε προηγούμενη εργασία [12] θα έπρεπε η κατασκευή του συστήματος να γίνει σε επίπεδο python με numpy arrays[24] κάτι που μειώνει σε μεγάλο βαθμό τη λειτουργικότητα του κώδικα και δεν επιτρέπει την παράλληλη επεξεργασία παρά μόνο αν ο ίδιος ο χρήστης καλέσει το MPI εξωτερικά. Επομένως η εφαρμογή της μεθόδου στο FEniCS περιορίζεται προς το παρόν μόνο σε προβλήματα με αραιά πλέγματα που μπορούν να επιλυθούν σειριακά.

Εκτός από τα προβλήματα που παρουσιάζονται στην εφαρμογή της μεθόδου στο FEniCS η απευθείας επίλυση του επαυξημένου συστήματος διαταράσσει την αραιή δομή (sparsity structure) της αρχικής Ιακωβιανής.

Γιαυτό το λόγο προτάθηκε εναλλακτικά από τους Keller & Chan [10] ένας αλγόριθμος τύπου BE (block elimination).

Επιλύονται τα παρακάτω γραμμικά συστήματα:

$$\mathrm{F_u}y=F_\lambda \tag{7.11}$$

$$\mathrm{F_u} z = -F \tag{7.12}$$

Στη συνέχεια η διόρθωση της παραμέτρου δλ και της λύσης δu προκύπτει από τις εξής σχέσεις:

$$\delta\lambda = \frac{-N_u^T z - N}{N_\lambda - N_u^T y} \tag{7.13}$$

$$\delta u = z - \delta\lambda y \tag{7.14}$$

Ο αλγόριθμος αυτός μπορεί να χρησιμοποιηθεί όσο η Ιακωβιανή δεν είναι ιδιάζουσα. Αυτό σημαίνει ότι θα πρέπει να γίνει κατάλληλη επιλογή του βήματος στη παράμετρο του μήκους τόξου προκειμένου να ξεπεραστεί το σημείο στροφής και όχι να προσεγγιστεί μια μικρή γειτονιά γύρω από αυτό. Στην δεύτερη περίπτωση ο αλγόριθμος θα οδηγήσει σε απόκλιση τη μέθοδο Newton και επομένως ο παραμετρικός βηματισμός πρέπει να ξεκινήσει από την αρχή.

Ο παραπάνω αλγόριθμος απαιτεί την επίλυση συστημάτων που περιλαμβάνουν μόνο την αρχική Ιακωβιανή που έχει προκύψει από τη διατύπωση των μεταβολικών μορφών. Επιπροσθέτως, ο υπολογισμός της διόρθωσης της λύσης και της παραμέτρου είναι 'matrix-free'. Τα εσωτερικά γινόμενα διανυσμάτων και γενικότερα οι βαθμωτές πράξεις είναι εύκολα και ταυτόχρονα αποτελεσματικά υλοποιήσιμες με το FEniCS. Στη συνέχεια παρουσιάζεται ο κώδικας στο FEniCS όπου εφαρμόζεται η μέθοδο παραμετροποίησης μήκους τόξου στο πρόβλημα 3D Bratu με βάση τον αλγόριθμο BE.

7.3.1 Εφαρμογή στο πρόβλημα Bratu

Με βάση την αρίθμηση μεταξύ των γραμμών 1-87 ορίζεται η γεωμετρία, οι συνοριακές συνθήκες Dirichlet και επιλύεται το μη γραμμικό πρόβλημα ώστε να προσδιοριστούν 2 λύσεις πριν το σημείο στροφής για τον υπολογισμό των παραγώγων με πεπερασμένες διαφορές.

Python Code

```
from dolfin import*

#Domain omega [0,1]x[0,1]x[0,0.5]
mesh=BoxMesh(Point(0,0,0),Point(1,1,0.5),30,30,30)
#Function Space
V=FunctionSpace(mesh,"Lagrange",2)

#Dirichlet Boundaru conditions
u_D0=Constant(0)
tol = 1e-14

def boundary_D0(x,on_boundary):
    return on_boundary and (near(x[0],0,tol) or near(x[0],1,tol) \
    or near(x[1],0,tol) or near(x[1],1,tol))
bc=DirichletBC(V,u_D0,boundary_D0)

#Initial solutions u0,u1 for arc-length continuation method

# parameter
lamda=6.8

# define nonlinear terms for variational formulation

def fun(u,lamda):
    return lamda*exp(u)


def fun2(u):
    return exp(u)
```

```
30
31 #Define trial and test functions
32 v = TestFunction(V)
33 u=TrialFunction(V)
34
35 # most recently computed solution
36 u_k=Function(V)
37
38 # Residual
39 F = inner(grad(u),grad(v))*dx-fun(u,lamda)*v*dx
40 F=action(F, u_k)
41
42 # Jacobian
43 J = derivative(F,u_k,u)
44
45 #solve nonlinear problem
46 problem = NonlinearVariationalProblem(F, u_k, bc, J)
47 solver  = NonlinearVariationalSolver(problem)
48
49 #solver parameters
50 prm = solver.parameters
51 prm['newton_solver']['absolute_tolerance'] = 1E-8
52 prm['newton_solver']['relative_tolerance'] = 1E-7
53 prm['newton_solver']['maximum_iterations'] = 25
54 prm['newton_solver']['relaxation_parameter'] = 1.0
55 prm['newton_solver']["convergence_criterion"]='incremental'
56 solver.parameters["newton_solver"]["linear_solver"] = "gmres"
57 solver.parameters["newton_solver"]["preconditioner"] ="hypre_euclid"
58
59 solver.solve()
60 #solution u0
61 u0 = Function(V)
62 u0.assign(u_k)
63 print 'Inf norm',norm(u0.vector(),'linf')
64 l0 = lamda
65
66 #update lamda
67 lamda += 0.005
68
69 # Residual
70 F = inner(grad(u),grad(v))*dx-fun(u,lamda)*v*dx
71 F=action(F, u_k)
72
73 # Jacobian
74 J = derivative(F,u_k,u)
75
76 #solve nolinear problem
77 problem = NonlinearVariationalProblem(F, u_k, bc, J)
78 solver  = NonlinearVariationalSolver(problem)
79
80 solver.solve()
81
82 #solution u1
83 u1 = Function(V)
84 u1.assign(u_k)
85 print 'Inf norm',norm(u1.vector(),'linf')
86 l1 = lamda
87
```

Στη συνέχεια (γραμμή 96) ξεκινάει ο βασικός βρόγχος επαναλήψεων για το βηματισμό στη παράμετρο του μήκους τόξου με βάση τον αλγόριθμο BE. Ο αριθμός των βημάτων καθορίζεται από τη παράμετρο num_steps . Ορίζονται οι παράγωγοι της εξίσωσης (5.5) με πεπερασμένες διαφορές (γραμμές 98-103). Η έναρξη των εσωτερικών επαναλήψεων της Newton γίνεται στη γραμμή 115. Ορίζεται το μη γραμμικό μεταβολικό πρόβλημα (γραμμές 118-124). Η διατύπωση τροποποιείται προκειμένου να μην είναι απαραίτητη η κλήση του JIT compiler σε κάθε επανάληψη. Αυτό επιτυγχάνεται ορίζοντας αρχικά τη παράμετρο λ ως σταθερά του FEniCS. Στη συνέχεια επιλέγεται η εξαγωγή της Ιακωβιανής και του υπολοίπου να γίνει με αυτοματοποιημένο τρόπο. Επίσης ο εκθετικός όρος δεν ορίζεται με python function όπως προηγουμένως.

Python Code

```
88 #linear solver for manually implemented Newton
89 solver1=PETScKrylovSolver('gmres','hypre_euclid')
90
91 #Number of arc-length parameter steps
92 num_steps=10
93
```

```
94 ####  Block elimination algorithm ########
95
96 for n in range(num_steps+1):
97
98     # derivatives of N with respect to u or lamda
99     unorm=norm((u1.vector()-u0.vector()),'l2')
100    ds=sqrt(unorm**2+(l1-l0)**2)
101    Nu=PETScVector()
102    Nl=(l1-l0)/ds
103    Nu=(u1.vector()-u0.vector()) /ds
104   #update initial solution
105    u0.assign(u1)
106    l0=l1
107
108    #Newton's method parameters
109    eps = 1.0
110    tol = 1.0E-8  #tolerance
111    k = 0         #iteration count
112    maxiter = 25  #maximum Newton iterations
113
114    #Newton's method loop
115    while eps > tol and k < maxiter:
116        k += 1
117
118       #Nonlinear Variational problem
119        lcon=Constant(l1)
120        v = TestFunction(V)
121        u = TrialFunction(V)
122        F = -dot(grad(u1),grad(v))*dx +lcon*exp(u1)*v*dx
123        J = derivative(F, u1, u)
124        L=rhs(F)
```

Ο υπολογισμός της παραγώγου του υπολοίπου ως προς τη παράμετρο λ (γραμμές 127-130) είναι σχετικά απλός καθώς γίνεται με βάση τη γνωστή λύση u1. Η διακριτοποίηση γίνεται αυτόματα από το FEniCS. Αντίστοιχα η εξίσωση ορισμού του μήκους τόξου περιλαμβάνει απλά ένα εσωτερικό γινόμενο διανυσμάτων το οποίο υλοποιείται με τη συνάρτηση inner. Δεν απαιτείται κάποιος υποβιβασμός των vector objects σε python arrays. Μεταξύ των γραμμών 135-147 εφαρμόζεται ο αλγόριθμος BE για τον υπολογισμό της διόρθωσης της λύσης και της παραμέτρου στη τρέχουσα επανάληψη Newton. Στη συνέχεια ανανεώνονται οι παράγωγοι με βάση τη τρέχουσα λύση u1.

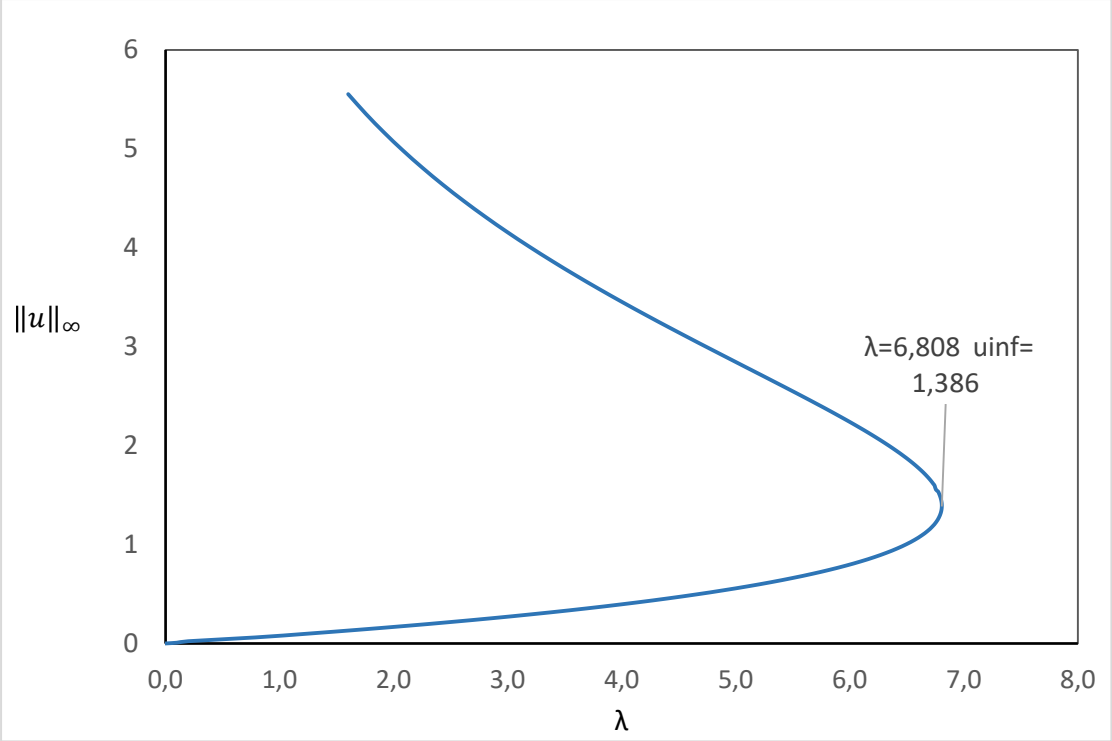
Python Code

```
125         Jacob,R=assemble_system(J,L,bc)
126 
127        # Residual derivative with respect to lamda
128         Fl=fun2(u1)*v*dx
129         Rl=assemble(Fl,tensor=PETScVector())
130 
131         #arc length equation
132         N=Nu.inner(u1.vector()-u0.vector())+Nl*(l1-l0)-10.
133 
134         #solve linear systems
135         z = Function(V)
136         solver1.solve(Jacob, z.vector(), R)
137         y = Function(V)
138         solver1.solve(Jacob, y.vector(), -Rl)
139 
140         #parameter correction
141         dl=(-(N+Nu.inner(z.vector())))/(Nl+Nu.inner(y.vector()))
142         #update solution
143         u1.vector()[:] = u1.vector()[:] + (z.vector()[:] + dl*y.vector()[:])
144 
145         #update lamda
146         l1 += dl
147 
```

```
148         #update derivatives
149         unorm=norm((u1.vector()-u0.vector()),'l2')
150         ds=sqrt(unorm**2+(l1-l0)**2)
151         Nl=(l1-l0)/ds
152         Nu=(u1.vector()-u0.vector()) /ds
153 
154         eps = norm((z.vector() + dl*y.vector()),'l2')
155     lamda = l1
156     print 'eps',eps
157     print  'lamda', l1
158     print 'Inf norm',norm(u1.vector(),'linf')
159 vtkfile = File("Bratu_arclengthBE.pvd")
160 vtkfile << u1
```

Ο παραπάνω κώδικας είναι συμβατός λειτουργικά με το FEniCS και μπορεί να εκτελεστεί και παράλληλα. Θα πρέπει ωστόσο να συγκριθούν τα αποτελέσματα από τη χρήση πεπερασμένων διαφορών για τον υπολογισμό των παραγώγων (που είναι πιο εύκολα υλοποιήσιμες) σε σχέση με την εφαρμογή των εξισώσεων (7.7)-(7.10).

Στο διάγραμμα 5.1 παρουσιάζεται η μεταβολή της νόρμας της λύσης συναρτήσει της παραμέτρου λ. Η διακριτοποίηση έγινε με 30 πεπερασμένα στοιχεία (τετράπλευρα) κατά τη x,y και z διεύθυνση. Τα αποτελέσματα δε μεταβλήθηκαν σημαντικά με περαιτέρω πύκνωση του πλέγματος. Παρουσιάζεται ένα σημείο στροφής η θέση του οποίου συμφωνεί με τις τιμές που αναφέρονται στη βιβλιογραφία.



Διάγραμμα 7.1 Μεταβολή της απειροστής νόρμας της λύσης συναρτήσει της παραμέτρου λ όπως προέκυψε από τη παραμετροποίηση μήκους τόξου και τον αλγόριθμο block-elimination.

7.4 Η μέθοδος αποπληθωρισμού (deflation)

Η μέθοδο παραμετροποίησης με το μήκος τόξου επιτρέπει τον εντοπισμό λύσεων και πέρα από το σημείο στροφής. Ωστόσο ο εντοπισμός του άνω κλάδου της λύσης δε μπορεί να γίνει άμεσα αλλά θα πρέπει πρώτα να προσεγγιστεί μια κοντινή περιοχή του σημείου στροφής. Μια από τις πιο πρόσφατα αναπτυγμένες μεθόδους για τον εντοπισμό διακριτών λύσεων μη γραμμικών μερικών διαφορικών εξισώσεων είναι ο αποπληθωρισμός [13] [14] [15] [16]. Σύμφωνα με αυτή τη μέθοδο τροποποιείται κατάλληλα το υπόλοιπο του προβλήματος προκειμένου να αποκλειστούν γνωστές λύσεις και η Newton να συγκλίνει σε νέες λύσεις ακόμα και όταν τροφοδοτείται με αρχική εκτίμηση που χρησιμοποιήθηκε για το προσδιορισμό γνωστών λύσεων. Η Ιακωβιανή του νέου αποπληθωρισμένου (deflated) προβλήματος είναι αρκετά πυκνή αλλά αποδεικνύεται ότι μπορεί να χρησιμοποιηθεί αποτελεσματικά ο προσταθεροποιητής της αρχικής Ιακωβιανής και οι επαναλήψεις του επιλύτη Krylov να μην αυξηθούν σημαντικά. Στη συνέχεια αναπτύσσεται η μέθοδος αυτή σε μη γραμμικά μεταβολικά προβλήματα ώστε να εφαρμοστεί στο πρόβλημα Bratu και να επιλυθεί με το FEniCS.

Έστω $f(u)$ η μη γραμμική εξίσωση της οποίας αναζητείται η λύση. Σε όρους μεταβολικής διατύπωσης αναζητείται λύση στο παρακάτω πρόβλημα:

$$\langle f(u), v\rangle = 0 \; \forall \; v \in U \; , (U \; συναρτησιακός \; χώρος)$$

$$\Rightarrow \int_{\Omega} f(u) v dx = 0 \; \forall \; v \in U \tag{7.14}$$

Ας υποθέσουμε ότι είναι ήδη γνωστή μια λύση του παραπάνω προβλήματος έστω r.Ορίζουμε ένα τελεστή αποπληθωρισμού (deflation operator):

$$M_{p,a}(u;r) = \left(\frac{1}{\|u-r\|_U{}^p} + a\right) I \ , p \geq 1, a \geq 0 \tag{7.15}$$

όπου I ο ταυτοτικός τελεστής.

Το α λαμβάνει συνήθως μη μηδενικές τιμές έτσι ώστε το αποπληθωρισμένο υπόλοιπο να μη τείνει στο 0 όταν το $\|u-r\|_U{}^p$ γίνεται υπερβολικά μεγάλο.Ο εκθέτης της νόρμας p συνήθως λαμβάνει τις τιμές 1,2. Ωστόσο η επιλογή του μπορεί να εξαρτάται από το ίδιο το πρόβλημα και αν η αλγοριθμική διαδικασία επίλυσης συγκλίνει. Το αποπληθωρισμένο υπόλοιπο (deflation residual) κατασκευάζεται με εφαρμογή του τελεστή αποπληθωρισμού στο $f(u)$:

$$g(u) = M_{p,a}(u;r) f(u) \tag{7.16}$$

Με βάση τη τροποποιημένη εξίσωση αναζητείται λύση στο παρακάτω μεταβολικό πρόβλημα:

$$\langle g(u), v\rangle = 0 \ \forall \ v \in U$$

$$\Rightarrow \langle \left(\frac{1}{\|u-r\|_U{}^p} + a\right) f(u), v\rangle = 0 \ \forall \ v \in U \tag{7.17}$$

Η τροποποιημένη εξίσωση όπως και η αρχική είναι μη γραμμική και πρέπει να επιλυθεί με επαναληπτική μέθοδο Newton. Για λόγους απλούστευσης συμβολίζουμε :

$$n(u) = \left(\frac{1}{\|u-r\|_U{}^p} + a\right) \tag{7.18}$$

Γραμμικοποίηση γύρω από μια προσέγγιση της λύσης u_k:

$$g(u) \cong g(u_k) + g'(u_k)\delta u \tag{7.19}$$

Η παράγωγος Frechet του $g(u_k)$, $g'(u_k)$ προκύπτει:

$$g'(u_k) = n(u_k) f'(u_k) + f(u_k) n'\,(u_k) \tag{7.20}$$

Η παράγωγος Frechet $n'\,(u_k)$ δίνεται από την εξίσωση (7.21)

$$n'\left(u^k\right) = -\frac{p u_k}{\|u_k - r\|^{p+2}} \tag{7.21}$$

Επομένως σε κάθε επανάληψη Newton επιλύεται το παρακάτω γραμμικό πρόβλημα ως προς τη διόρθωση της λύσης δu :

$$\langle g(u_k) + g'(u_k)\delta u, v\rangle = 0 \ \forall \ v \in U$$

$$\Rightarrow \int_{\Omega} (g(u_k) + g'(u_k)\delta u)v dx = 0 \; \forall \; v \in U$$

$$\Rightarrow \int_{\Omega} (n(u_k)f'(u_k) + f(u_k)n'(u_k))\, \delta u \; v \; dx + \int_{\Omega} g(u_k)v dx = 0 \; \forall \; v \in U$$

$$u_{k+1} = u_k + \omega \delta u \tag{7.22}$$

Στο παραπάνω μεταβολικό πρόβλημα εμφανίζεται η παράγωγος Frechet $f'(u_k)$ της αρχικής εξίσωσης. Αυτός ο υπολογισμός δεν είναι βολικός καθώς θα πρέπει η διατύπωση να προσαρμόζεται κάθε φορά στο αντίστοιχο πρόβλημα. Εναλλακτικά από τη διακριτοποίηση της εξίσωσης (5.20) προκύπτει:

$$J_G(u_k) = n(u_k)J_F(u_k) + F(u_k)d^T(u_k) \tag{7.23}$$

όπου $J_G(u_k), J_F(u_k)$ οι αντίστοιχες αποπληθωρισμένες και μη αποπληθωρισμένες Ιακωβιανές , $F(u_k)$ το διάνυσμα του διακριτοποιημένου υπολοίπου και $d^T(u_k)$ το διάνυσμα της διακριτοποιημένης παραγώγου Frechet του αποπληθωρισμένου τελεστή. Στη συνέχεια η εξάρτηση από τη λύση u_k μπορεί να παραληφθεί.

Με τη μορφή που ορίζεται στην εξίσωση (7.23) η J_G είναι πυκνή ακόμα και αν η δομή της J_F είναι αραιή καθώς προκύπτει με $1^{ης}$ τάξης διόρθωση της J_F. Εφαρμόζοντας την προσέγγιση Sherman-Morisson [17] στην εξίσωση (7.23) προκύπτει:

$$J_G^{-1} = (nJ_F + Fd^T)^{-1} = \frac{J_F^{-1}}{n} - \frac{\frac{1}{n^2}J_F^{-1}Fd^TJ_F^{-1}}{1 + \frac{1}{n}d^TJ_F^{-1}F} \tag{7.24}$$

Επομένως ο υπολογισμός του διανύσματος της διόρθωσης δU_k στη τρέχουσα επανάληψη Newton που αντιστοιχεί στις εξισώσεις (7.22) μπορεί να υπολογιστεί ως εξής:

$$\delta U_k = -J_G^{-1}G = -\frac{J_F^{-1}G}{n} + \frac{\frac{1}{n^2}J_F^{-1}Fd^TJ_F^{-1}G}{1 + \frac{1}{n}d^TJ_F^{-1}F} = -J_F^{-1}F + \frac{\frac{1}{n}J_F^{-1}Fd^TJ_F^{-1}F}{1 + \frac{1}{n}d^TJ_F^{-1}F}$$

$$\Rightarrow \delta U_k = -\left(1 - \frac{\frac{1}{n}d^TJ_F^{-1}F}{1 + \frac{1}{n}d^TJ_F^{-1}F}\right)J_F^{-1}F \tag{7.25}$$

Ο όρος $J_F^{-1}F$ αντιστοιχεί στη διακριτοποίηση και επίλυση του αρχικού προβλήματος ενώ ο όρος $d^TJ_F^{-1}F$ είναι ένα απλό εσωτερικό γινόμενο διανυσμάτων. Οι υπόλοιπες πράξεις είναι βαθμωτές. Όπως και στη περίπτωση του αλγορίθμου BE στην ενότητα 7.3 η παραπάνω σχέση μπορεί πολύ εύκολα να εφαρμοστεί στο FEniCS καθώς δε περιλαμβάνει καθόλου πράξεις μεταξύ πινάκων εκτός από την επίλυση του γραμμικού προβλήματος σε κάθε επανάληψη Newton.

7.4.1 Εφαρμογή στο πρόβλημα Bratu

Στις γραμμές 24-74 επιλύεται το μη γραμμικό πρόβλημα με τη μέθοδο Newton για μια κοντινή τιμή της ζητούμενης παραμέτρου προκειμένου η λύση να τροφοδοτηθεί ως αρχική εκτίμηση στην αποπληθωρισμένη Newton. Η χρήση της λύσης για τη ζητούμενη παράμετρο ως αρχική εκτίμηση δε μπορεί να οδηγήσει στην εύρεση κάποιας νέας λύσης καθώς ο τελεστής αποπληθωρισμού δεν ορίζεται (βλ εξίσωση (7.15) για u=r). Στη συνέχεια επιλύεται με τον ίδιο τρόπο το μη αποπληθωρισμένο πρόβλημα και οι λύσεις αποθηκεύονται σε διαφορετικά FEniCS functions (γραμμές 77-114).

Python Code

```
 7 from dolfin import*
 8 #Domain omega [0,1]x[0,1]x[0,0.5]
 9 mesh=BoxMesh(Point(0,0,0),Point(1,1,0.5),10,10,10)
10 V=FunctionSpace(mesh,"Lagrange",2)
11
12 #Dirichlet boundary conditions
13 u_D0=Constant(0)
14 tol = 1e-14
15
16 def boundary_D0(x,on_boundary):
17     return on_boundary and (near(x[0],0,tol) or near(x[0],1,tol) \
18     or near(x[1],0,tol) or near(x[1],1,tol))
19 bc0=DirichletBC(V,u_D0,boundary_D0)
20
```

```
21 ########################
22 #apply param step to provide an initial guess to deflated newton
23
24 # parameter
25 lamda=6.
26
27 parameters['linear_algebra_backend'] = "PETSc"
28
29 # define nonlinear term
30 def fun(u,lamda):
31     return lamda*exp(u)
32
33 v = TestFunction(V)
34
35
36 # most recently computed solution
37 u_k=Function(V)
38 du=TrialFunction(V)
39
40 #jacobian matrix
41 J=-inner(grad(du),grad(v))*dx+fun(u_k,lamda)*du*v*dx
42 #residual
43 F=inner(grad(u_k),grad(v))*dx-fun(u_k,lamda)*v*dx
44 du=Function(V)
45 u=Function(V)
46 dU=du.vector()
47
48 solver1=PETScKrylovSolver('gmres','hypre_euclid')
49 #Newton's Method parameters
50 omega = 1.0       # relaxation parameter
51 eps = 1.0
52 tol = 1.0E-5
53 k = 0
54 maxiter = 20
55
```

```
64
65            #calculate L-2 norm
66            eps = norm(du.vector(),'l2')
67
68            #update solution
69            u.vector()[:] = u_k.vector() + omega*du.vector()
70            u_k.assign(u)
71
72 u_kdef=Function(V)
73 u_kdef.assign(u)
74 print 'Inf norm',norm(u_kdef.vector(),'linf')
75
76 ########################
77 ##undeflated problem
78
79 lamda=5.
80 # most recently computed solution
81
82 du=TrialFunction(V)
83
84 #jacobian matrix
85 J=-inner(grad(du),grad(v))*dx+fun(u_k,lamda)*du*v*dx
86 #residual
87 F=inner(grad(u_k),grad(v))*dx-fun(u_k,lamda)*v*dx
88 du=Function(V)
```

```
 89 u=Function(V)
 90 dU=du.vector()
 91
 92 #Newton's Method parameters
 93 omega = 1.0       # relaxation parameter
 94 eps = 1.0
 95 tol = 1.0E-5
 96 k = 0
 97 maxiter = 20
 98
 99   #Newton iterations
100 while eps > tol and k < maxiter:
101
102            k += 1
103            Jac,R=assemble_system(J,F,bc0)
104
105            solver1.solve(Jac,dU,R)
106
107            #calculate L-2 norm
108            eps = norm(du.vector(),'l2')
109
110            #update solution
111            u.vector()[:] = u_k.vector() + omega*du.vector()
112            u_k.assign(u)
113
114 print 'Inf norm',norm(u.vector(),'linf')
```

Ορίζονται οι παράμετροι του αποπληθωρισμένου τελεστή (117-122) και οι συναρτήσεις που θα αποθηκεύουν τη λύση του αποπληθωρισμένου προβλήματος (132-134). Ξεκινά ο βρόγχος επαναλήψεων Newton (γραμμή 142). Ορίζεται και επιλύεται εκ νέου το μη γραμμικό μεταβολικό πρόβλημα για τη τρέχουσα προσέγγιση της λύσης u_k (γραμμές 146-153) όπως απαιτείται από την εξίσωση (7.25).

Python Code

```
115
116 ############
117 #deflated problem
118 #define deflation parameters
119 #power
120 p=1.0
121 #shift
122 a=0.
123 # deflated Newton's Method parameters
124 omega =1.    # relaxation parameter
125 eps = 1.0
126 tol = 1.0E-5
127 k = 0
128 maxiter =100
129
130 rerr=1.0E-14
131
132 u_def=Function(V)
133 du_def=Function(V)
134 DU_def=du_def.vector()
135
136 #solution of undeflated problem for sherman-morrison formula (JF)^-1 *F
137 du_undef=Function(V)
138 DU_undef=du_undef.vector()
139
140
```

```
141   # deflated Newton iterations
142 while eps > tol and k < maxiter:
143
144     k += 1
145     print k
146     du=TrialFunction(V)
147     #jacobian matrix non-deflated
148     J=-inner(grad(du),grad(v))*dx+fun(u_kdef,lamda)*du*v*dx
149     #residual non-deflated
150     F=inner(grad(u_kdef),grad(v))*dx-fun(u_kdef,lamda)*v*dx
151     Jac,R=assemble_system(J,F,bc0)
152
153     solver1.solve(Jac,DU_undef,R)
```

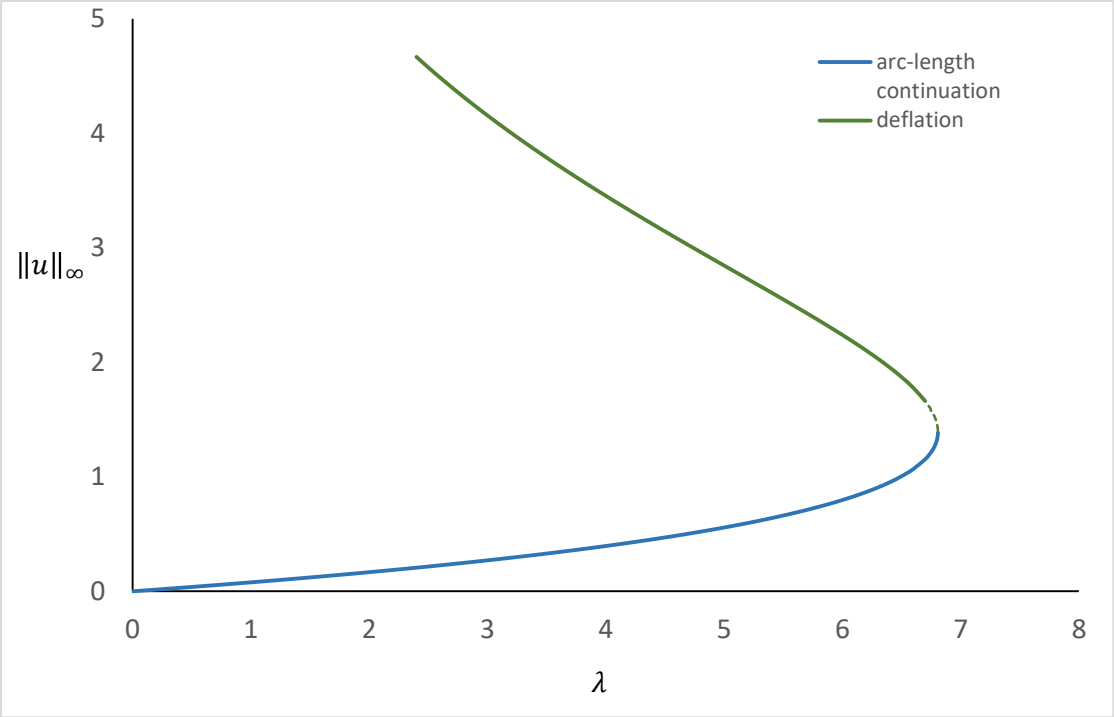
Στις γραμμές 155-170 ορίζεται ο αποπληθωρισμένος τελεστής και η παράγωγός του. Με τη συνθήκη if/else εξασφαλίζεται ότι (για τιμή $\alpha \neq 0$) στη περίπτωση που χρησιμοποιηθεί ως αρχική εκτίμηση η λύση του μη αποπληθωρισμένου προβλήματος δε θα υπάρξει διαίρεση με το 0. Φυσικά αυτή η επιλογή αρχικής εκτίμησης δε θα οδηγήσει σε νέα λύση καθώς το υπόλοιπο θα είναι απλά ανάλογο του αρχικού. Στη συνέχεια υπολογίζεται η τρέχουσα διόρθωση της λύσης με βάση την εξίσωση (7.25).

Python Code

```
154
155     if near(norm((u_kdef.vector()-u.vector()),'l2'),0.,rerr):
156        n_uk=a
157        ndot_uk=0.
158        dTdu_undef=0.
159     else:
160
161         # deflation operator
162
163        n_uk=  ( 1/(norm((u_kdef.vector()-u.vector()),'l2')**p)) +a
164
165        ndot_uk=PETScVector()
166        expr=norm((u_kdef.vector()-u.vector()),'l2')**(p+2)
167         #frechet derivative of deflation operator
168        ndot_uk=-p*(u_kdef.vector()/expr)
169       #calculate dot product of transpose(n'),(JF)^-1*F
170        dTdu_undef=ndot_uk.inner(-DU_undef)
171
172
173     # sherman-morrison coefficient
174     m=1- ( (  (1/n_uk)*dTdu_undef )/(1+ (1/n_uk)*dTdu_undef) )
175
176     du_def.vector()[:]=m*DU_undef
177
178 #    #calculate L-2 norm
179     eps = norm(du_def.vector(),'l2')
180
181 #    #update solution
182     u_def.vector()[:] = u_kdef.vector() + omega*du_def.vector()
183     u_kdef.assign(u_def)
184
185 print 'Inf norm',norm(u_def.vector(),'linf')
186
187 vtkfile = File("Bratu_undeflated.pvd")
188 vtkfile << u
189 vtkfile = File("Bratu_deflated.pvd")
190 vtkfile << u_def
```

Στο συγκεκριμένο πρόβλημα η νόρμα $\|u - r\|_U{}^p$ δεν έγινε πολύ μεγάλη κατά τη διάρκεια των επαναλήψεων. Μάλιστα παρατηρήθηκε ότι για επιλογή α=0 η σύγκλιση σε νέα λύση (που αντιστοιχεί στον άνω κλάδο) ήταν γρηγορότερη. Επιπλέον η καλύτερη επιλογή για τον εκθέτη ήταν η τιμή p=1. Ένα πρόβλημα το οποίο αναφέρεται και στη σχετική βιβλιογραφία [16] είναι η διαδικασία επιλογής αρχικής εκτίμησης ειδικά στη περίπτωση αρκετά πυκνών πλεγμάτων. Έστω λ_s η τιμή της παραμέτρου για την οποία αναζητείται μια νέα λύση που δεν έχει ανακαλυφθεί (στον παραπάνω κώδικα αντιστοιχεί σε λ=5). Παρατηρείται ότι όταν ως αρχική εκτίμηση χρησιμοποιείται μια γνωστή λύση για $\lambda > \lambda_s$ η αποπληθωρισμένη Newton συγκλίνει πράγματι σε νέα λύση που αντιστοιχεί στον άνω κλάδο. Ωστόσο αν χρησιμοποιηθεί ως αρχική εκτίμηση λύση με $\lambda < \lambda_s$ τότε η αποπληθωρισμένη Newton συγκλίνει στην αρχική γνωστή λύση ή αποκλίνει. Αυτή η συμπεριφορά δυσκολεύει την επιλογή αρχικής εκτίμησης όσο η τιμή της παραμέτρου πλησιάζει την οριακή τιμή στο σημείο στροφής. Επιπλέον σε προβλήματα που δεν υπάρχει κάποια διαθέσιμη πληροφορία για τη μορφή του χώρου των λύσεων θα πρέπει να γίνουν πολλές δοκιμές στην αρχική εκτίμηση. Αυτό που προτείνεται [13] είναι αρχικά όλοι οι σύνηθεις κλάδοι της λύσης να προσδιοριστούν με κάποια μέθοδο

παραμετρικού βηματισμού (όπως η παραμετροποίηση μήκους τόξου). Στη συνέχεια εξάγεται μια αρχική εκτίμηση ως μέσος όρος όλων των λύσεων στους γνωστούς κλάδους. Πιο προχωρημένες μέθοδοι προτείνουν τη χρήση των ειδώλων των γνωστών συμμετρικών λύσεων όταν αναμένεται η ύπαρξη μη συμμετρικών λύσεων (όπως σε προβλήματα ροής). Σε κάθε περίπτωση η χρήση αρχικής εκτίμησης πολύ κοντά σε κάποια γνωστή λύση έχει ως αποτέλεσμα μικρά διαστήματα εμπιστοσύνης και η σύγκλιση της Newton μπορεί να είναι πολύ αργή.



Διάγραμμα 7.2 . Μεταβολή της νόρμας της λύσης συναρτήσει της παραμέτρου λ. Ο κάτω κλάδος της λύσης μέχρι το σημείο στροφής προέκυψε με παραμετροποίηση μήκους τόξου ενώ ο πάνω με αποπληθωρισμό. Σε μια περιοχή κοντά στο σημείο στροφής η μέθοδος αποπληθωρισμού αδυνατεί να δώσει λύση.

7.4.2 Εφαρμογή σε πρόβλημα ροής ρευστού σε κανάλι με απότομη διαστολή

Ο ορισμός του τελεστή αποπληθωρισμού μπορεί να γενικευτεί περαιτέρω όταν είναι γνωστή μια οικογένεια λύσεων έστω $r_1, r_2, \dots, r_M$ και αναζητείται μια νέα διακριτή λύση u . Για πολλαπλές γνωστές λύσεις ο τελεστής ορίζεται ως το γινόμενο των επιμέρους τελεστών:

$$M_{p,a}(u; r_1, r_2, .., r_M) = \prod_{i=1}^{M} M_{p,a}(u; r_i) \tag{7.26}$$

Η δράση του τελεστή στην αρχική μη γραμμική εξίσωση είναι:

$$G(u; v) = \langle \prod_{i=1}^{M} \left(\frac{1}{\|u - r_i\|_U{}^p} + a \right) f(u), v \rangle \tag{7.27}$$

Ο υπολογισμός της αποπληθωρισμένης Ιακωβιανής γίνεται με τον ίδιο τρόπο που αναφέρθηκε στην ενότητα 7.4 . Ωστόσο ο υπολογισμός της παραγώγου του τελεστή αποπληθωρισμού είναι πιο περίπλοκος. Προκειμένου να είναι αυτοματοποιήσιμη η παραγώγιση και να μπορούν να χρησιμοποιηθούν νόρμες στο χώρο Hilbert είναι προτιμότερο να οριστούν οι νόρμες ως ολοκληρωτικοί όροι και όχι ως νόρμες διανυσμάτων. Για παράδειγμα για νόρμα τύπου H1 ορίζεται:

$$\|u - r_i\|_U = \left(\int_\Omega |u - r_i|^2 dx \right)^{\frac{1}{2}} \tag{7.28}$$

Στη συνέχεια μπορεί να γίνει αυτοματοποιημένη αριθμητική παραγώγιση του ολοκληρωτικού όρου $\int_\Omega |u - r_i|^2 dx$ από το FEniCS.

Ορίζουμε:

$$n(u) = M_{p,a}(u;, r_2, .., r_M) = \prod_{i=1}^{M} M_{p,a}(u; r_i) = \prod_{i=1}^{M} n_i(u) \tag{7.29}$$

$$nd_i(\mathrm{u}) = \frac{d(\int_\Omega |u - r_i|^2 dx)}{du} \tag{7.30}$$

$$d_i(u) = \frac{d\left(\frac{1}{\|u - r_i\|_U{}^p} + a\right)}{du} = -\frac{p}{2} \frac{1}{\|u - r_i\|_U^{\frac{p}{2}+1}} \cdot nd_i(u) \tag{7.31}$$

Από την εξίσωση (7.31) μετά τη διακριτοποίηση προκύπτει ένα διάνυσμα όπου:

$$\dim d_i = \dim u \tag{7.32}$$

Η παράγωγος του τελεστή αποπληθωρισμού μπορεί να προκύψει επομένως από το κανόνα παραγώγισης γινομένου γνωρίζοντας τις παραγώγους των επιμέρους όρων:

$$d(u) = \frac{d(n(u))}{du} = \sum_{i=1}^{M} \frac{n(u)}{n_i(u)} d_i(u) \tag{7.33}$$

Εκτός από τη τροποποίηση του τελεστή η διαδικασία που ακολουθείται για την εύρεση νέων διακριτών λύσεων παραμένει ίδια. Η διόρθωση της νέας λύσης στη τρέχουσα επανάληψη Newton προκύπτει ως παραπροϊόν της διόρθωσης της λύσης για το αρχικό πρόβλημα.

Για να εξεταστεί η εφαρμογή του πολλαπλού αποπληθωρισμού στο FEniCS επιλύεται ένα δι-διάστατο πρόβλημα απότομης διαστολής ρευστού σε κανάλι το οποίο αναφέρεται και στη σχετική βιβλιογραφία [13].

Το πρόβλημα περιγράφεται από τις αδιαστατοποιημένες εξισώσεις Navier-Stokes για Νευτωνικό, ασυμπίεστο ρευστό σε μόνιμη κατάσταση:

$$-\frac{1}{Re}\nabla^2 u + u \cdot \nabla u + \nabla p = 0$$

$$\nabla \cdot u = 0 \tag{7.33}$$

όπου u η διανυσματική συνάρτηση της ταχύτητας, p η βαθμωτή συνάρτηση της πίεσης και Re ο αριθμός Reynolds.

Η γεωμετρία προκύπτει ως η ένωση 2 ορθογωνίων παραλληλογράμμων όπως φαίνεται και στην εικόνα 7.2:

$$\Omega = [0,2.5]x[-1,1] \cup [2.5,150]x[-6,6]$$

Ως συνοριακές συνθήκες λαμβάνονται:

Στην είσοδο του καναλιού ροή τύπου Poiseuille:

$$u_x = 1 - y^2, u_y = 0, \;\; x = 0$$

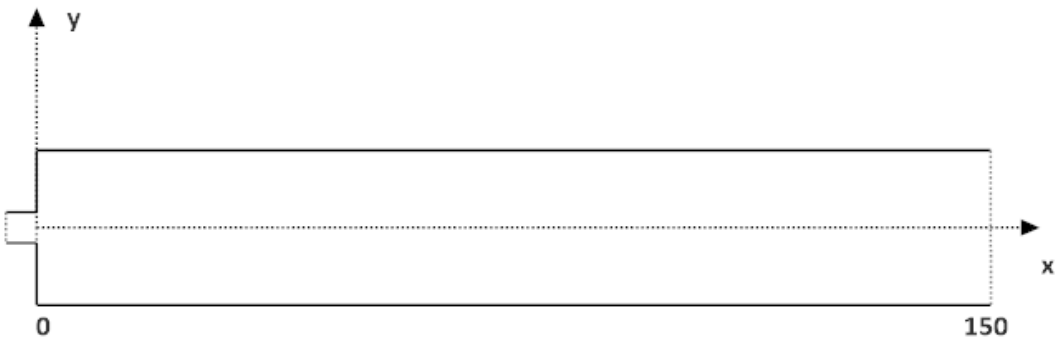
Στα τοιχώματα:

$$u = 0$$

Στην έξοδο του καναλιού ελεύθερη ροή του ρευστού:

$$\nabla u \cdot n = pn, x = 150$$

όπου n το μοναδιαίο κάθετο διάνυσμα στην επιφάνεια.



Εικόνα 7.2 Δισδιάστατο κανάλη ροής με απότομη διαστολή στην είσοδο.

Το παραπάνω πρόβλημα αποτελείται από ένα σύστημα μερικών διαφορικών εξισώσεων που πρέπει να τροποποιηθεί κατάλληλα για να επιλυθεί . Το FEniCS έχει τη δυνατότητα επίλυσης τέτοιων συστημάτων ως ένα άθροισμα μεταβολικών μορφών που σε κάθε εξίσωση αντιστοιχεί διαφορετικό test function.

Έστω οι συναρτήσεις βάσης:

$$v, q \in U \text{ , } (U \text{ συναρτησιακός χώρος})$$

Η μεταβολική μορφή των εξισώσεων (7.33) είναι:

$$-\int_{\Omega}\frac{1}{Re}(\nabla^2 u + u\cdot\nabla u + \nabla p)v dx = 0 \quad v \in U \tag{7.34}$$

$$\int_{\Omega}(\nabla\cdot u)q dx = 0 \quad v \in U \tag{7.35}$$

Αθροίζοντας κατά μέλη τις εξισώσεις (7.34),(7.35) προκύπτει:

$$\int_{\Omega}(-\frac{1}{Re}\nabla^2 u + u\cdot\nabla u + \nabla p)v dx + \int_{\Omega}(\nabla\cdot u)q dx = 0 \quad \Rightarrow$$

$$-\frac{1}{Re}\int_{\Omega}\nabla^2 u v dx + \int_{\Omega} u\cdot\nabla u v dx + \int_{\Omega}\nabla p v dx + \int_{\Omega}\nabla\cdot u q dx = 0$$

$$v, q \in U \tag{7.36}$$

Επομένως με ολοκλήρωση κατά μέλη του 1ο και του 4ο όρου της εξίσωσης (7.36) :

$$\frac{1}{Re}\int_{\Omega}\nabla u\nabla v dx + \int_{\Omega} u\cdot\nabla u v dx - \int_{\Omega}\nabla\cdot v p dx + \int_{\Omega}(\nabla\cdot u)q dx \; -\frac{1}{Re}\int_{\Gamma_{out}} p n v ds = 0 \tag{7.37}$$

Ο υπολογισμός της Ιακωμβιανής επιλέγεται και σε αυτή τη περίπτωση να γίνει αυτόματα από το FEniCS.

Το παραπάνω πρόβλημα είναι συμμετρικό γύρω από τον άξονα x (εικόνα 7.2) και για χαμηλούς αριθμούς Re (<18) έχει συμμετρική λύση που είναι ευσταθής. Για ένα κρίσιμο αριθμό Re και πάνω παρουσιάζεται διακλάδωση (birfucation) της λύσης . Προκύπτει ένας ασταθής συμμετρικός κλάδος και (πιθανώς περισσότερα από ένα) ζευγάρια μη συμμετρικών κλάδων [18].

Η κατασκευή της γεωμετρίας και του πλέγματος έγινε στο Gmsh. Το πλέγμα που κατασκευάστηκε είναι μη δομημένο αποτελούμενο από 10^5 τριγωνικά στοιχεία και $6\cdot 10^4$ κόμβους. Η διακριτοποίηση στο FEniCS έγινε με πεπερασμένα στοιχεία Taylor-Hood [19] δηλαδή χρησιμοποιήθηκαν διωνυμικές συναρτήσεις βάσης για τη ταχύτητα και γραμμικές συναρτήσεις βάσης για τη πίεση. Τα γραμμικά αλγεβρικά συστήματα σε κάθε επανάληψη Newton επιλύθηκαν με τον άμεσο παράλληλο επιλύτη MUMPS του PETSc.

Προκειμένου να είναι δυνατή η παράλληλη επίλυση του προβλήματος με το FEniCS πρέπει να γίνει μια προεπεξεργασία του πλέγματος που δημιουργήθηκε στο Gmsh. Αρχικά το αρχείο τύπου .msh μετατρέπεται μέσω του dolphin σε αρχείο .xml . Ωστόσο τα αρχεία τύπου .xml δε μπορούν να προσπελαστούν στο FEniCS κατά τη παράλληλη επίλυση. Γιαυτό το λόγο τα αρχεία .xml μετατρέπονται εκ νέου σειριακά σε μορφή hdf5 από το DOLFIN όπως φαίνεται στο παρακάτω απόσπασμα:

Python Code

```
from dolfin import *
mesh = Mesh("pipe.xml")
subdomains = MeshFunction("size_t", mesh, "pipe_physical_region.xml")
boundaries = MeshFunction("size_t", mesh, "pipe_facet_region.xml")
hdf = HDF5File(mesh.mpi_comm(), "file.h5", "w")
hdf.write(mesh, "/mesh")
hdf.write(subdomains, "/subdomains")
hdf.write(boundaries, "/boundaries")
```

Το αρχείο hdf5 που περιέχει τις πληροφορίες του πλέγματος μπορεί να εισαχθεί σε οποιοδήποτε script. Επίσης μπορεί να γίνει ανάκτηση των υποχωρίων και των συνόρων (subdomains/boundaries):

Python Code

```
from dolfin import *
#import mesh from hdf5 file for parallel processing
mesh = Mesh()
hdf = HDF5File(mesh.mpi_comm(), "file.h5", "r")
hdf.read(mesh, "/mesh", False)
#import subdomains and boundaries
subdomains = MeshFunction("size_t", mesh, mesh.topology().dim())
hdf.read(subdomains, "/subdomains")
boundaries = MeshFunction("size_t", mesh, mesh.topology().dim()-1)
hdf.read(boundaries, "/boundaries")
```

Η αλγοριθμική διαδικασία που ακολουθήθηκε για την ανακάλυψη των μη συμμετρικών λύσεων με τον αποπληθωρισμό είναι η εξής:

1.Ανακαλύπτεται ο συμμετρικός κλάδος ξεκινώντας από $Re = 10$, χρησιμοποιώντας απλό παραμετρικό βηματισμό και βήμα $dRe = 0.75$. Ως αρχική εκτίμηση για την έναρξη του continuation χρησιμοποιείται η μηδενική λύση ενώ στη συνέχεια χρησιμοποιείται η λύση για την αμέσως προηγούμενη τιμή της παραμέτρου. Ο παραμετρικός βηματισμός τερματίζεται όταν $Re = 85$.

2.Εφόσον είναι γνωστή η συμμετρική λύση εφαρμόζεται αποπληθωρισμός και αναζητείται μια νέα λύση καθώς αυξάνεται η τιμή του αριθμού Reynolds με βήμα $dRe = 0.75$. Ουσιαστικά η μέθοδος αποπληθωρισμού ‘’ακολουθεί’’ το συμμετρικό κλάδο αναζητώντας νέες λύσεις. Ως αρχική εκτίμηση χρησιμοποιείται μια συμμετρική λύση για κοντινή τιμή της παραμέτρου.

3.Όταν βρεθεί μια νέα (μη συμμετρική) λύση με τον αποπληθωρισμό τότε χρησιμοποιείται απλός παραμετρικό βηματισμός για την εύρεση του υπόλοιπου κλάδου.

4.Τα βήματα 2-3 επαναλαμβάνονται εφαρμόζοντας πολλαπλό αποπληθωρισμό αποκλείοντας όλες τις γνωστές λύσεις έως ότου δε μπορούν να ανακαλυφθούν νέες λύσεις για τιμή της παραμέτρου στο διάστημα $[10,85]$.

Για την εφαρμογή της διαδικασίας στο FEniCS χρησιμοποιούνται διαφορετικά αρχεία script για κάθε παραμετρικό βηματισμό και αποπληθωρισμό που εκτελείται. Η επικοινωνία μεταξύ τους γίνεται με αρχεία τύπου hdf που περιέχουν τις λύσεις.

Στη συνέχεια παρουσιάζονται αποσπάσματα από την εφαρμογή της παραπάνω διαδικασίας στο FEniCS.

Ορισμός πεπερασμένων στοιχείων Taylor-Hood και μεικτού συναρτησιακού χώρου:

Python Code

```
#define Taylor-Hood Finite Elements
Ve = VectorElement("CG", triangle, 2)
Qe = FiniteElement("CG", triangle, 1)
Ze = MixedElement([Ve, Qe])
#define mixed Function Space
Z  = FunctionSpace(mesh, Ze)
```

Ορισμός συνοριακών συνθηκών τύπου Dirichlet:

Python Code

```
# Inlet BC
poiseuille = Expression(("-(x[1] + 1) * (x[1] - 1)", "0.0")
, degree=1, mpi_comm=Z.mesh().mpi_comm())
def inflow(x, on_boundary):
    return on_boundary and near(x[0], 0.0)
bc_inflow = DirichletBC(Z.sub(0), poiseuille, inflow)

 # Wall BC
def wall(x, on_boundary):
    return on_boundary and not near(x[0], 0.0) and not near(x[0], 150.0)
bc_wall = DirichletBC(Z.sub(0),(0,0), wall)

#Dirichlet boundary conditions for usol
bcs = [bc_inflow, bc_wall]
```

Μαρκάρισμα του συνόρου που αντιστοιχεί στην έξοδο του ρευστού από το κανάλι για την εφαρμογή της συνοριακής συνθήκης ελεύθερης ροής τύπου Neumann:

Python Code

```
boundaries.set_all(0)
class Outflow(SubDomain):
      def inside(self, x, on_boundary):
                return on_boundary and near(x[0], 150.0)


Outflow().mark(boundaries, 1)
ds_outflow = ds(subdomain_data=boundaries)(1)
```

Ορισμός μοναδιαίου κάθετου διανύσματος :

Python Code

```
n = FacetNormal(mesh)
```

Ορισμός συναρτήσεων βάσης και της λύσης. Ορισμός υπολοίπου και αυτόματη παραγώγιση του. Εφαρμογή της μεθόδου Newton υπό τη μορφή αυτόματου μη γραμμικού επιλύτη:

Python Code

```
(v,q)=TestFunctions(Z)

#solution contains vector-function u and scalar function p
usol=Function(Z)
#split solution to its components in ufl format
(u,p)=split(usol)

du=TrialFunction(Z)

#Reynolds number
Re=10.

#Residual
F =( (1.0/Constant(Re)) * inner(grad(u), grad(v))*dx+ inner(grad(u)*u, v)*dx
- div(v)*p*dx+ q*div(u)*dx- Constant( 1.0/Re) * inner(v, p*n)*ds_outflow)
#Jacobian
J=derivative(F,usol,du)

#solve nolinear problem
problem = NonlinearVariationalProblem(F, usol, bcs, J)
solver  = NonlinearVariationalSolver(problem)

#solver parameters
prm = solver.parameters
prm['newton_solver']['absolute_tolerance'] = 1E-6
prm['newton_solver']['relative_tolerance'] = 1E-5
prm['newton_solver']['maximum_iterations'] = 40
prm['newton_solver']['relaxation_parameter'] = 1.0
prm['newton_solver']["convergence_criterion"]='incremental'
solver.parameters["newton_solver"]["linear_solver"] = "mumps"
solver.parameters["newton_solver"]["preconditioner"] ="none"

solver.solve()
```

Εφαρμογή πολλαπλού αποπληθωρισμού (εδώ για 4 γνωστές λύσεις). Αρχικά ορίζονται οι παράμετροι του αποπληθωρισμού και της Newton καθώς και οι απαραίτητες συναρτήσεις που θα αποθηκεύουν τη τρέχουσα λύση, τη τρέχουσα διόρθωση της αποπληθωρισμένης και της μη αποπληθωρισμένης λύσης. Στη συνέχεια εντός του βρόχου while ορίζεται το μη γραμμικό μεταβολικό πρόβλημα και επιλύεται το γραμμικό σύστημα μετά τη διακριτοποίηση:

Python Code

```
############
#deflated problem
#define deflation parameters
#power
power=1.0
#shift
shift=1.0
# deflated Newton's Method parameters
omega =0.75   # relaxation parameter
eps = 1.0
tol = 1.0E-5
k = 0
maxiter =40

#solution for the third deflation
usol_def4=Function(Z)
```

```
usol_def4.assign(uinit)
(u_def4,p_def4)=split(usol_def4)
dusol_def4=Function(Z)

Re=43.
#solution of undeflated problem for sherman-morrison formula (JF)^-1 *F
dusol_undef=Function(Z)

#  # deflated Newton iterations
while eps > tol and k < maxiter:
    k += 1
    du=TrialFunction(Z)
    #Residual
    F =( (1.0/Constant(Re)) * inner(grad(u_def4), grad(v))*dx
    + inner(grad(u_def4)*u_def4, v)*dx- div(v)*p_def4*dx+ q*div(u_def4)*dx)


    #Jacobian
    J=derivative(F,usol_def4,du)
    #assemble linear system
    Jac,R=assemble_system(J,F,bcsdu)
    #solve
    solve(Jac,dusol_undef.vector(),-R,"mumps","none")
```

Στη συνέχεια ορίζονται οι νόρμες H1 (Hilbert) και υπολογίζονται αυτόματα από το FEniCS. Ο τελεστής αποπληθωρισμού προκύπτει ως το γινόμενο των επιμέρους όρων που περιλαμβάνουν τις γνωστές λύσεις. Η παράγωγος του τελεστή προκύπτει από το κανόνα γινομένου ενώ οι νόρμες παραγωγίζονται αυτόματα ως προς την άγνωστη λύση από το FEniCS. Η διόρθωση της άγνωστης αποπληθωρισμένης λύσης στη τρέχουσα επανάληψη γίνεται με τη προσέγγιση Sherman-Morisson ως παραπροϊόν της διόρθωσης της λύσης του αρχικού προβλήματος.

Python Code

```
    #squared norms
    normd1=assemble(inner((usol_def4-usol_undef),(usol_def4-usol_undef))*dx)
    normd2=assemble(inner((usol_def4-usol_def1),(usol_def4-usol_def1))*dx)
    normd3=assemble(inner((usol_def4-usol_def2),(usol_def4-usol_def2))*dx)
    normd4=assemble(inner((usol_def4-usol_def3),(usol_def4-usol_def3))*dx)


    n1=(normd1**(-power/2))+shift
    n2=(normd2**(-power/2))+shift
    n3=(normd3**(-power/2))+shift
    n4=(normd4**(-power/2))+shift

    #deflation operator
    n_uk=n1*n2*n3*n4

    #norm derivatives
    normderiv1=assemble(derivative((inner((usol_def4-usol_undef)
    ,(usol_def4-usol_undef))*dx),usol_def4))

    normderiv2=assemble(derivative((inner((usol_def4-usol_def1)
    ,(usol_def4-usol_def1))*dx),usol_def4))

    normderiv3=assemble(derivative((inner((usol_def4-usol_def2)
    ,(usol_def4-usol_def2))*dx),usol_def4))

    normderiv4=assemble(derivative((inner((usol_def4-usol_def3)
    ,(usol_def4-usol_def3))*dx),usol_def4))
```

```
    dfactor1=(-power/2)*(normd1**((-power/2)-1.0))
    dfactor2=(-power/2)*(normd2**((-power/2)-1.0))
    dfactor3=(-power/2)*(normd3**((-power/2)-1.0))
    dfactor4=(-power/2)*(normd4**((-power/2)-1.0))
```

```
    ndot_uk=PETScVector()

    #deflation operator derivative
    ndot_uk=(dfactor1*normderiv1*n2*n3*n4+dfactor2*normderiv2*n1*n3*n4
    +dfactor3*normderiv3*n2*n1*n4
    +n1*n2*n3*dfactor4*normderiv4    )


    #calculate dot product of transpose(n'),(JF)^-1*F
    dTdu_undef=ndot_uk.inner(-dusol_undef.vector())

    # sherman-morrison coefficient
    m=1- ( (  (1/n_uk)*dTdu_undef )/(1+ (1/n_uk)*dTdu_undef) )

    dusol_def4.vector()[:]=m*dusol_undef.vector()

#    #calculate L-2 norm
    eps = norm(dusol_def4.vector(),'l2')

#    #update solution
    usol_def4.vector()[:]  += omega*dusol_def4.vector()

    if MPI.rank(mpi_comm_world()) == 0:
       print 'iteration:',k
       print 'relative error:',eps
```

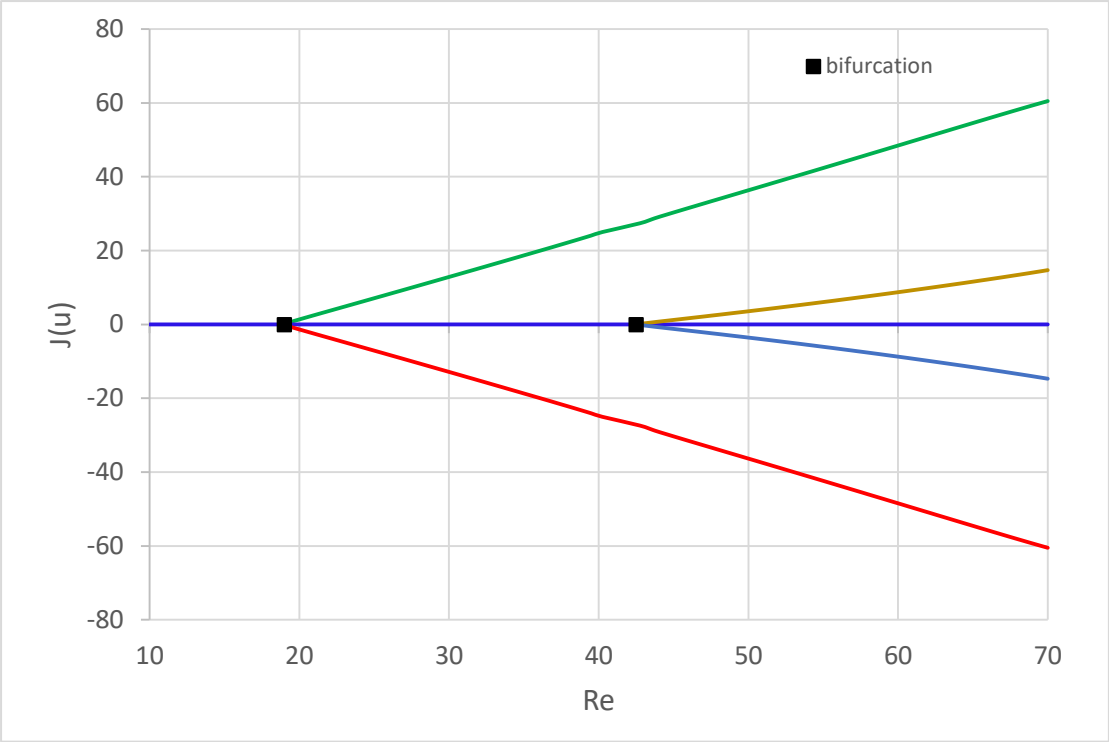
Τα αποτελέσματα του αποπληθωρισμού για αριθμούς Reynolds $10 \leq Re \leq 70$ παρουσιάζονται στο διάγραμμα 7.3. Ως μέτρο της ασυμμετρίας της λύσης χρησιμοποιήθηκε η παρακάτω έκφραση:

$$J(u) = \pm \int_{\Omega} |u - u_{-}|^2 \, dx$$

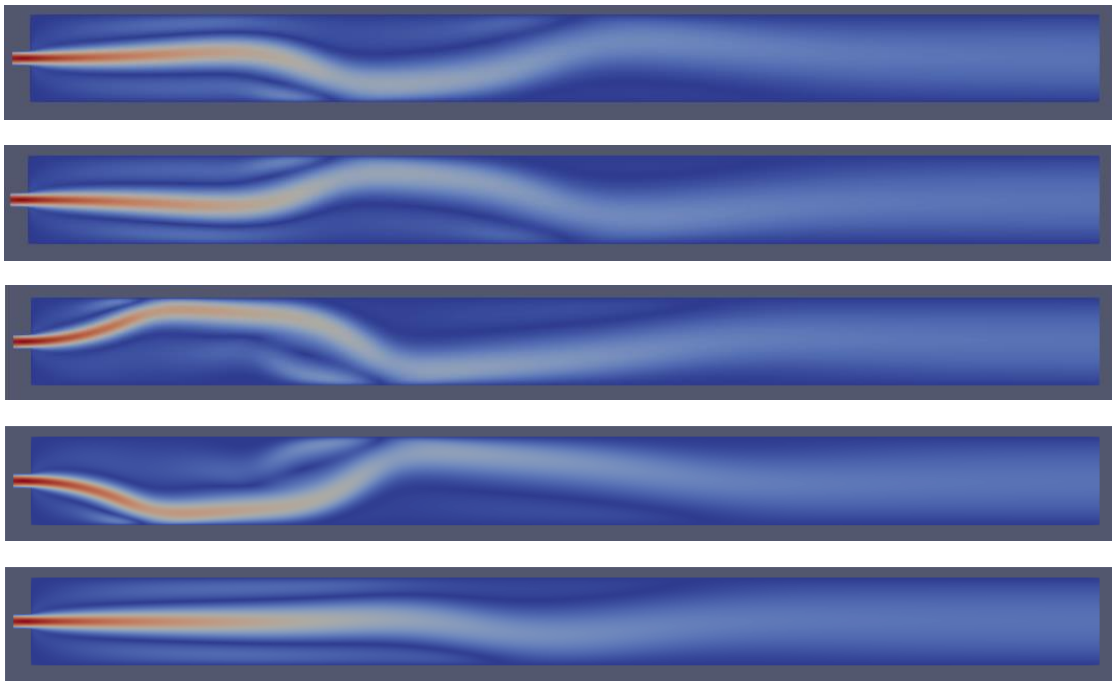
όπου u, η συμμετρική και u_{-} μια μη συμμετρική λύση της διανυσματικής ταχύτητας του ρευστού για τη τρέχουσα τιμή του αριθμού Re κατά μήκος ολόκληρου του αγωγού. Το πρόσημο επιλέγεται ανάλογα με το αν η ασυμμετρία είναι θετική ή αρνητική σε σχέση με τον y=0 (+ για θετική και – για αρνητική ασυμμετρία).

Αν και αναμενόταν άλλο ένα σημείο διακλάδωσης για Re>70 δεν κατέστη δυνατή η εύρεση της ασταθούς συμμετρικής λύσης από την μη αποπληθωρισμένη Newton. Αντιθέτως σύγκλινε σε μια 5η μη συμμετρική λύση για Re>70. Στη συνέχεια η 5η εφαρμογή του αποπληθωρισμού σύγκλινε και αυτή στην ίδια μη συμμετρική λύση. Πιθανώς θα έπρεπε να μειωθεί το βήμα στο παραμετρικό βηματισμό του συμμετρικού κλάδου, να γίνει χαλάρωση της μεθόδου Newton και να χρησιμοποιηθεί κάποια πιο αποτελεσματική μέθοδος για τη τροφοδότηση αρχικής εκτίμησης στον αποπληθωρισμό όπως η μέθοδος reflection (χρησιμοποιείται το είδωλο της προσδιορισμένης μη συμμετρικής λύσης ως προς τον άξονα y=0 ως αρχική εκτίμηση).

Ωστόσο η εφαρμογή του αποπληθωρισμού κρίνεται αποτελεσματική εφόσον προσδιορίστηκαν 5 από τις 6 μη συμμετρικές λύσεις (εικόνα 7.3) χρησιμοποιώντας μια πολύ απλή μέθοδο για την επιλογή αρχικής εκτίμησης. Δηλαδή η συμπεριφορά της μεθόδου είναι καλή ακόμα και όταν δε συνδυάζεται με κάποια άλλη προχωρημένη μέθοδο παραμετρικού βηματισμού/εύρεσης αρχικής εκτίμησης.



Διάγραμμα 7.3 Ασυμμετρία της λύσης για τη ταχύτητα σε απότομη διαστολή ρευστού σε κανάλι. Η συμμετρική λύση είναι ευσταθής μέχρι ένα συγκεκριμένο αριθμό Re. Στη συνέχεια παρουσιάζεται διακλάδωση σε μη συμμετρικές λύσεις ενώ η συμμετρική λύση είναι ασταθής. Η κλίμακα χρώματος χρησιμοποιείται για καλύτερη διάκριση των κλάδων.



Εικόνα 7.3 Μη συμμετρικές λύσεις της ταχύτητας του ρευστού για Re=82.5. Η 6η μη συμμετρική λύση (το είδωλο της τελευταίας λύσης) δε κατέστη δυνατό να βρεθεί αποκλειστικά με τη μέθοδο αποπληθωρισμού.

Συμπεράσματα

Τα συμπεράσματα που προκύπτουν με βάση την εμπερία χρήσης του FEniCS για την επίλυση απαιτητικών προβλημάτων συνοψίζονται στα εξής:

- Το FEniCS απαιτεί μια στοιχειώδη κατανόηση και εμπειρία χρήσης της μεθόδου Πεπερασμένων Στοιχείων καθώς και βασικές γνώσεις προγραμματισμού σε γλώσσα C++/python. Επιπλέον ο χρήστης θα πρέπει να γνωρίζει καλά το φυσικό και μαθηματικό υπόβαθρο του προβλήματος προκειμένου να εκφράσει σωστά τις μεταβολικές μορφές και τις συνοριακές συνθήκες. Το λογισμικό δε παρέχει έτοιμα πακέτα (cases ή modules) για διαφορετικού τύπου προβλήματα όπως πιθανώς κάνουν άλλα εμπορικά λογισμικά. Επομένως ο χρήστης πρέπει να επενδύσει κάποιο χρόνο για την εξοικείωσή του με το λογισμικό. Ωστόσο το FEniCS παρέχει μια υψηλού επιπέδου διεπαφή με το χρήστη και διατηρεί το μαθηματικό συμβολισμό του προβλήματος σε αντίθεση με πηγαίους κώδικες πεπερασμένων στοιχείων.
- Το πλεονέκτημα του FEniCS έναντι άλλων λογισμικών είναι ότι δεν υπάρχει περιορισμός στο είδος του προβλήματος, στην πολυπλοκότητα και το πλήθος των διαφορικών εξισώσεων που θα επιλυθούν καθώς όλες εκφράζονται με τη γενική μεταβολική μορφή που αναγνωρίζεται από τη γλώσσα UFL.
- Το FEniCS μπορεί να αντιμετωπίσει αποτελεσματικά σχεδόν οποιαδήποτε μορφή συνοριακών συνθηκών. Ο χρήστης μπορεί να ορίσει εσωτερικά ή εξωτερικά υποσύνορα κατά τη διαδικασία κατασκευής του πλέγματος και σύνθετες εκφράσεις για συνοριακές συνθήκες τύπου Dirichlet.
- Είναι προτιμότερο για σύνθετες γεωμετρίες και πλέγματα να χρησιμοποιείται το εξωτερικό λογισμικό Gmsh. Η διαδικασία διαχείρισης αρχείων msh από το DOLFIN είναι βελτιστοποιημένη τόσο για σειριακή όσο και για παράλληλη επεξεργασία.
- Η αντιμετώπιση διαφορετικής διάστασης (1D, 2D, 3D) γραμμικών ή μη γραμμικών προβλημάτων δεν παρουσιάζει σημαντικές διαφοροποιήσεις.
- Η παράλληλη επεξεργασία προβλημάτων σε πολλαπλούς πυρήνες είναι ιδιαίτερα αποδοτική αν αναλογιστεί κανείς ότι ο χρήστης δεν απαιτείται να κάνει οποιαδήποτε αλλαγή στο κώδικα που κατασκεύασε για σειριακή επεξεργασία. Η επιτάχυνση των υπολογισμών σε 8 πυρήνες αγγίζει το 75% σε ένα τρι-διάστατο μη γραμμικό πρόβλημα με 4 εκατομμύρια βαθμούς ελευθερίας. Η δέσμευση μνήμης σε ένα τόσο μεγάλο πρόβλημα είναι διαχειρίσιμη ακόμα και από ένα σύγχρονο προσωπικό υπολογιστικό σύστημα καθώς δε ξεπερνά τα 22 GB.
- Το FEniCS μπορεί να συνδυαστεί εύκολα με μεθόδους ανάλυσης του χώρου των λύσεων μη γραμμικών προβλημάτων διατηρώντας τις δυνατότητες για αποδοτική παράλληλη επεξεργασία αρκεί οι μέθοδοι αυτοί να μη περιλαμβάνουν πολλαπλασιασμό ή δημιουργία επαυξημένων πινάκων. Οι πράξεις αυτές δεν υποστηρίζονται από το λογισμικό. Επομένως θα πρέπει να υποβιβαστούν τα matrix objects σε python arrays,θυσιάζοντας την αυτοματοποιημένη διαδικασία κλήσης των συναρτήσεων επικοινωνίας του MPI. Πιθανώς να υπάρχει ένας πιο ευέλικτος τρόπος χρησιμοποιώντας απευθείας τα εργαλεία των πακέτων γραμμικής άλγεβρας. Ωστόσο κάτι τέτοιο ξεφεύγει από τους στόχους της παρούσας εργασίας.

Προτάσεις για περαιτέρω έρευνα

Στη παρούσα εργασία εξετάστηκε αναλυτικά η εφαρμογή μεθόδων ανάλυσης του χώρου των λύσεων μη γραμμικών προβλημάτων στο FEniCS. Προκύπτει ιδιαίτερο ενδιαφέρον για την εφαρμογή της παραμετροποίησης μήκους τόξου υπό τη μορφή επαυξημένου συστήματος. Κάτι τέτοιο θα μπορούσε να γίνει εναλλακτικά ορίζοντας ένα νέο μεταβολικό πρόβλημα όπου η παράμετρος θα είναι συνάρτηση των συντεταγμένων του χωρίου σε συνδυασμό με κάποιο περιορισμό (για παράδειγμα να έχει την ίδια τιμή σε όλο το χωρίο) προκειμένου να κατασκευάσει το FEniCS το επαυξημένο σύστημα εξισώσεων. Θα μπορούσαν να συνδυαστούν επίσης οι μέθοδοι παραμετρικού βηματισμού με ανάλυση ευστάθειας αξιοποιώντας τον eigensolver SLEPc του PETSc. Η δημιουργία ενός αποδοτικού αλγορίθμου παραμετροποίησης μήκους τόξου/ αποπληθωρισμού και άλλων μεθόδων παραμετρικού βηματισμού με ανάλυση ευστάθειας στο FEniCS θα ήταν ένα πολύ ισχυρό εργαλείο για την παραμετρική ανάλυση ρεαλιστικών μη γραμμικών μοντέλων.

Βιβλιογραφία

[1] M. G. Larson and F. Bengzon, The Finite Element Method: Theory, Implementation, and Applications.Texts in Computational Science and Engineering. Springer, 2013.

[2] Α. Γ. Μπουντουβής, Υπολογιστική Ανάλυση με τη μέθοδο των Πεπερασμένων Στοιχείων,Εισαγωγικές Σημειώσεις, ΕΜΠ, Αθήνα, 1992.

[3] A. Logg, K. A. Mardal and G. N. Wells, Automated Solution of Differential Equations by the Finite Element Method-The FeniCS Book, Springer, 2011.

[4] Hoffman, Jansson, Logg and Wells, Dolfin User Manual, 2006.

[5] U. Ayachit, The Paraview Guide:Community Edition, Kitware, 2017.

[6] S. L. Mouradian and A. Avdis, A Gmsh Tutorial, London, 2012.

[7] A. G. Boudouvis, *Mechanisms of surface instabilities and pattern formation in ferromagnetic liquids.PhD Thesis, University of* Minnesota,USA, 1987.

[8] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing,* vol. 3, pp. 856-869, 1986.

[9] Α. Σπυρόπουλος, *Υπολογισμοί μεγάλης κλίμακας με μεθόδους παράλληλης επεξεργασίας σε μη γραμμικά προβλήματα διεπιφανειακής μαγνητο-ρευστομηχανικής,Διδακτορική Διατριβή, ΕΜΠ,* Αθήνα, 2003.

[10] T. F. C. Chan and H. B. Keller, "Arc length continuation and multi-grid techniques for nonlinear elliptic eigenvalue problems," *SIAM Journal on Scientific and Statistical Computing,* vol. 3, no. 2, p. 173–194, 1982.

[11] R. Glowinski, H. B. Keller and L. Reinhart, "Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems," *SIAM Journal on Scientific and Statistical Computing,* vol. 6, no. 4, p. 793–832, 1985.

[12] Ν. Αμαρτωλός, *Επίλυση προβλημάτων Πεπερασμένων Στοιχείων με χρήση της πλατφόρμας FeniCS,Μεταπτυχιακή Διατριβή, ΕΜΠ,* Αθήνα, 2018.

[13] P. E. Farell, A. Birkisson and S. W. Funke, "Deflation techniques for finding distinct solutions of nonlinear partial differential equations," *SIAM Methods and Algorithms for Scientific Computing,* vol. 37, no. 4, p. A2026–A2045, 2015.

[14] P. E. Farell, . C. H. L. Beentjes and A. Birkisson, "The computation of disconnected bifurcation diagrams," 2016. arXiv:1603.00809 [math.NA].

[15] C. Beentjes, *Computing bifurcation diagrams with deflation,MSc Thesis, University of Oxford,* 2015.

[16] J. H. Adler, D. B. Emerson, P. E. Farell and S. P. Maclachlan, "Combining deflation and nested iteration for computing multiple solutions of nonlinear variational problems," *SIAM Journal on Scientific and Statistical Computing,* vol. 39, p. B29–B52, 2017.

[17] W. W. Hager, "Updating the Inverse of a Matrix," *SIAM Journal on Scientific and Statistical Computing,* vol. 31, pp. 221-239, 1989.

[18] S. Strogatz, Non-linear Dynamics and Chaos: With applications to Physics, Biology, Chemistry and Engineering, Perseus Books, 2015.

[19] C. Taylor and P. Hood, "A numerical solution of Navier Stokes Equations using Finite Element Technique," *Computers & Fluids,* vol. 1, pp. 73-100, 1973.

Ηλεκτρονικοί σύνδεσμοι

1. *https://www.comsol.com/multiphysics/finite-element-method*
2. *https://cdn.comsol.com/cyclopedia/finite-element-method/plot-using-linear-combinations.png*
3. *https://FEniCSproject.org/*
4. *https://www.uchicago.edu/*
5. *https://www.chalmers.se/en/Pages/default.aspx*
6. *https://www.mcs.anl.gov/petsc/*
7. *https://trilinos.github.io/*
8. *http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview*
9. *http://www.labri.fr/perso/pelegrin/scotch/*
10. *https://www.paraview.org/*
11. *http://gmsh.info/*
12. *https://www.docker.com/*
13. https://www.linux.org/
14. https://upload.wikimedia.org/wikiversity/en/5/5c/Multiprocessing.wiki.20150330.pdf
15. http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf
16. *https://www.amd.com/en/products/cpu/6380*
17. https://www.ssh.com/
18. https://openvpn.net/
19. https://www.openmp.org/
20. https://www.mpi-forum.org/
21. *https://www.cac.cornell.edu/Education/training/StampedeJan2015/Scalability.pdf*
22. http://calcul.math.cnrs.fr/IMG/pdf/matrix_lyon.pdf
23. *https://www.sharcnet.ca/Software/Ansys/16.2.3/en-us/help/cfx_mod/i1328402.html*
24. http://cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Παράρτημα Α

Διαθέσιμοι επιλύτες/προσταθεροποιητές

Στα κεφάλαια 3,4 χρησιμοποιήθηκε ο προκαθορισμένος επιλύτης του PETSc ενώ στα κεφάλαια 5,6 ο επαναληπτικός επιλύτης GMRES σε συνδυασμό με ένα προσταθεροποιητή μερικής αποικοδόμησης (incomplete lu decomposition). Γενικά το FEniCS διαθέτει ένα μεγάλο πλήθος άμεσων ή επαναληπτικών επιλυτών και προσταθεροποιητών το οποίο επεκτείνεται διαρκώς σε σχέση με τις αρχικές εκδόσεις. Με τις παρακάτω 2 εντολές είναι δυνατή η εκτύπωση των διαθέσιμων μεθόδων. Το output παρουσιάζεται στην εικόνα Α.1.

Python Code

```
list_linear_solver_methods()
list_krylov_solver_preconditioners()
```

```
Solver method   |  Description
--------------------------------------------------------------------------------
bicgstab        |  Biconjugate gradient stabilized method
cg              |  Conjugate gradient method
default         |  default linear solver
gmres           |  Generalized minimal residual method
minres          |  Minimal residual method
mumps           |  MUMPS (MUltifrontal Massively Parallel Sparse direct Solver)
petsc           |  PETSc built in LU solver
richardson      |  Richardson method
superlu         |  SuperLU
tfqmr           |  Transpose-free quasi-minimal residual method
umfpack         |  UMFPACK (Unsymmetric MultiFrontal sparse LU factorization)
Preconditioner    |  Description
---------------------------------------------------------------
amg               |  Algebraic multigrid
default           |  default preconditioner
hypre_amg         |  Hypre algebraic multigrid (BoomerAMG)
hypre_euclid      |  Hypre parallel incomplete LU factorization
hypre_parasails   |  Hypre parallel sparse approximate inverse
icc               |  Incomplete Cholesky factorization
ilu               |  Incomplete LU factorization
jacobi            |  Jacobi iteration
none              |  No preconditioner
petsc_amg         |  PETSc algebraic multigrid
sor               |  Successive over-relaxation
```

Εικόνα Α.1 Διαθέσιμοι solvers και preconditioners στην έκδοση 2017.2.0

Η εφαρμογή των παρακάτω μεθόδων εξαρτάται από το πακέτο γραμμικής άλγεβρας που χρησιμοποιείται (PETSc, uBLAS, Epetra, MTL4), η επιλογή του οποίου μπορεί να γίνει ως εξής:

Python Code

```
parameters['linear_algebra_backend']=backendname
```

Παράρτημα Β

Άλλες βιβλιοθήκες του FEniCS

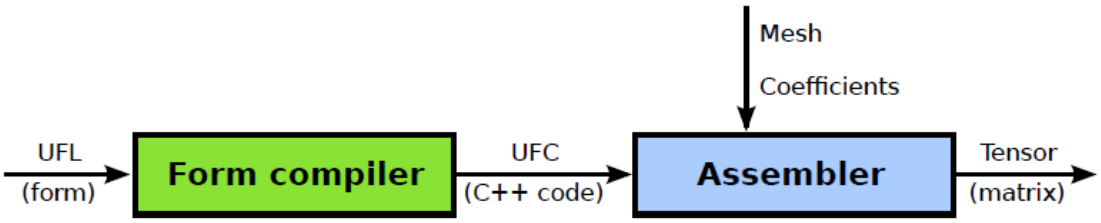
B.1 FIAT

Το FIAT (Finite Element Automatic Tabulator) είναι ένα προγραμματιστικό πακέτο Python το οποίο εξυπηρετεί την αυτόματη κατασκευή συναρτήσεων βάσης της μεθόδου πεπερασμένων στοιχείων. Το FIAT έχει τη δυνατότητα κατασκευής συναρτήσεων βάσης για ένα μεγάλο εύρος από κατηγορίες πεπερασμένων στοιχείων (γραμμές, τρίγωνα και τετράεδρα) όπως τα στοιχεία Lagrange, Raviart-Thomas, Brezzi-Douglas-Marini και Nedelec. Είναι επίσης ικανό να παράγει στοιχεία τανυστικών γινομένων (tensor-product elements) και έναν αριθμό από άλλα εξωτικά στοιχεία όπως τα Argyris, Hermite και Morley.

B.2 UFC

Ένα κεντρικό κομμάτι του FEniCS είναι η βιβλιοθήκη UFC (Unified Form-assembly Code). Το UFC είναι μια διεπαφή μεταξύ του συγκεκριμένου προβλήματος που επιλύεται και του γενικού περιεχομένου διαδικασιών που εφαρμόζονται σε όλα τα προγράμματα πεπερασμένων στοιχείων. Ειδικότερα ορίζει τη δομή και τη ταυτότητα του κώδικα που παράγεται από τους μεταγλωττιστές μορφών (form compilers) FFC και SFC για το DOLFIN. Η διεπαφή UFC εφαρμόζεται σε ένα μεγάλο εύρος προβλημάτων πεπερασμένων στοιχείων (περιλαμβάνοντας μεικτά πεπερασμένα στοιχεία ή μεθόδους discontinuous Galerkin) και μπορεί να χρησιμοποιείται σε συνδυασμό με βιβλιοθήκες όπου διαφέρουν κατά πολύ στο σχεδιασμό τους. Για αυτό το λόγο η διεπαφή δεν εξαρτάται από άλλα τμήματα ή βιβλιοθήκες του FEniCS και αποτελείται μόνο από μια συλλογή κλάσεων C++ .

Από το Φεβρουάριο του 2014 το UFC ενσωματώθηκε στο FFC.



Εικόνα Β.1 Διάγραμμα ροής της διαδικασίας εξαγωγής του αλγεβρικού συστήματος μετά τη διακριτοποίηση

B.3 INSTANT

Το instant είναι ένα μικρό python module για άμεση μεταγλώττιση (JIT-just in time compilation) κώδικα C++/C και μπορεί να συνδυαστεί αποδοτικά με τα εργαλεία παραγωγής κώδικα των DOLFIN, FFC και SCF.

Το Instant δεν είναι πλέον απαραίτητο μετά την έκδοση 2017.2.0 του FEniCS.

Παράρτημα Γ

Οδηγίες εγκατάστασης του FEniCS μέσω του Docker

Μια από τις πιο σύγχρονες λύσεις για την εγκατάσταση λογισμικού σε διαφορετικού τύπου πλατφόρμες και λειτουργικά συστήματα είναι τα αποκαλούμενα Docker containers. Το FEniCS Project παρέχει containers υψηλής απόδοσης τα οποία λειτουργούν καλά σε όλα τα λειτουργικά συστήματα όπως τα Linux, Mac και Windows. Για να μπορούν να χρησιμοποιηθούν τα FEniCS containers θα πρέπει να εγκατασταθεί πρώτα το Docker. Η εγκατάσταση του Docker είναι απλή και οδηγίες περιλαμβάνονται στη διαδικτυακή σελίδα του Docker. Μετά την εγκατάσταση του Docker εκτελείται η παρακάτω εντολή στο τερματικό παράθυρο του Docker για την εγκατάσταση του FEniCS:

Terminal

```
Terminal> curl -s https://get.fenicsproject.org | bash
```

Για την εκτέλεση του FEniCS χρησιμοποιείται η εντολή:

Terminal

```
Terminal> fenicsproject run
```

Άλλες δυνατότητες και περιεχόμενα του FEniCS μπορούν να βρεθούν εκτελώντας την εντολή *FEniCSproject help.*