Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Εφαρμοσμενων Μαθηματικων και Φυσικων Επιστημων
Τομεας Μαθηματικων
Δ.Π.Μ.Σ. Εφαρμοσμενες Μαθηματικες Επιστημες

Η μέθοδος πεπερασμένων στοιχείων σε παράλληλες αρχιτεκτονικές υπολογιστών

Μεταπτυχιακή Διπλωματική Εργασία

του

Θωμά Κάππα

Επιβλέπων:
Εμμανουήλ Γεωργούλης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2017

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Εφαρμοσμενων Μαθηματικων και Φυσικων Επιστημων
Τομεας Μαθηματικων
Δ.Π.Μ.Σ. Εφαρμοσμενες Μαθηματικες Επιστημες

Η μέθοδος πεπερασμένων στοιχείων σε παράλληλες αρχιτεκτονικές υπολογιστών

Μεταπτυχιακή Διπλωματική Εργασία

του

Θωμά Κάππα

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις ___ /___ /2017.

Εμμανουήλ Γεωργούλης	Βασίλειος Κοκκίνης	Κωνσταντίνος Χρυσαφίνος
Αναπληρωτής Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.
.....	.....	.....
(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*

Αθήνα, Απρίλιος 2017

Περιεχόμενα

Πρόλογος

Οι μερικές διαφορικές εξισώσεις, δηλαδή εξισώσεις που περιλαμβάνουν μια άγνωστη συνάρτηση και μερικές παραγώγους αυτής, χρησιμοποιούνται για να περιγράψουν μια ευρεία ποικιλία φαινομένων, από τη φυσική έως τη βιολογία και από τη μηχανική έως τα οικονομικά.

Ενώ όμως χρησιμοποιούνται εδώ και πολλές δεκαετίες σε κάθε τομέα της επιστήμης, η εύρεση αναλυτικών λύσεων αποτελεί ένα εγγενώς δύσκολο πρόβλημα, εκτός από κάποια απλουστευμένα μοντέλα. Αυτός ίσως είναι και ο λόγος που η αριθμητική επίλυση μερικών διαφορικών εξισώσεων, δηλαδή η χρήση μεθόδων της αριθμητικής ανάλυσης για την εύρεση προσεγγιστικών λύσεων γνωρίζει ιδιαίτερη άνθηση, με νέες μεθόδους που επιτυγχάνουν γρηγορότερη σύγκλιση και μπορούν να εφαρμοστούν σε μεγαλύτερη γκάμα προβλημάτων να προτείνονται συνεχώς.

Ένας άλλος λόγος είναι η μεγάλη πρόοδος που έχει γίνει στον τομέα των υπολογιστών. Η αριθμητική ανάλυση είναι συνυφασμένη με τη χρήση υπολογιστών αφού η επίλυση με αριθμητικές μεθόδους ακόμα και της απλούστερης εξίσωσης περιλαμβάνει την εκτέλεση τόσων πολλών πράξεων που η επίλυση στο χαρτί είναι πρακτικά αδύνατη. Ειδικά τα τελευταία χρόνια έχει αρχίσει η κατασκευή κυκλωμάτων που συνδυάζουν περισσότερους του ενός επεξεργαστές μέσα στο ίδιο κύκλωμα. Έτσι, ενώ παλαιότερα η χρήση παράλληλης επεξεργασίας αφορούσε μόνο πολύ μεγάλους οργανισμούς, που είχαν τη δυνατότητα να χρησιμοποιήσουν δεκάδες ή εκατοντάδες συστήματα συνδεδεμένα μεταξύ τους μέσω ενός γρήγορου δικτύου, πλέον ένας σύγχρονος κεντρικός επεξεργαστής διαθέτει μερικές δεκάδες πυρήνες ενώ μια κάρτα γραφικών μερικές χιλιάδες πάνω στο ίδιο ολοκληρωμένο κύκλωμα.

Στην παρούσα διπλωματική εργασία για το Δ.Π.Μ.Σ. “Εφαρμοσμένες Μαθηματικές Επιστήμες” του Ε.Μ.Π. θα μελετήσουμε τη χρήση σύγχρονων παράλληλων μεθόδων, και συγκεκριμένα την πλατφόρμα CUDA για κάρτες γραφικών της Nvidia, για την επίλυση του προβλήματος Poisson στις δύο διαστάσεις με ομογενείς συνοριακές συνθήκες Dirichlet. Σκοπός της εργασίας είναι να ερευνήσουμε αν η χρήση καρτών γραφικών μπορεί και σε ποιο βαθμό να επιταγχύνει την κατασκευή του γραμμικού συστήματος της μεθόδου πεπερασμένων στοιχείων και να συγκρίνουμε μεθόδους χαμηλής και υψηλής τάξης σύγκλισης.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ορισμένους ανθρώπους που με τον ένα ή τον άλλο τρόπο συνέβαλαν στην ολοκλήρωση της παρούσας διπλωματικής εργασίας. Πρώτο από όλους θα ήθελα να ευχαριστήσω τον επιβλέποντα της εργασίας, Καθηγητή Μανώλη Γεωργούλη που αμέσως αντιλήφθηκε τις ερευνητικές μου ανησυχίες και μου ανέθεσε ένα τόσο ενδιαφέρον θέμα, καθώς και για την αμέριστη στήριξή του. Δεν θα μπορούσα να μην ευχαριστήσω τον Dr. Zhaonan (Peter) Dong από το University of Leicester για την πολύτιμη βοήθεια που μου παρείχε στον ιδιαίτερα περιορισμένο χρόνο του. Θερμές ευχαριστίες και στους καθηγητές Κωνσταντίνο Χρυσαφίνο και Βασίλειο Κοκκίνη που δέχθηκαν να είναι μέλη της τριμελούς επιτροπής αξιολόγησης της διπλωματικής εργασίας. Τέλος, θέλω να ευχαριστήσω την οικογένειά μου για την πολύτιμη ηθική και υλική συμπαράσταση όλα αυτά τα χρόνια.

Στην Κορίνα.

Κεφάλαιο 1

Η μέθοδος πεπερασμένων στοιχείων για ελλειπτικά προβλήματα

Υπάρχουν τρεις μεγάλες κατηγορίες μεθόδων για την αριθμητική επίλυση μερικών διαφορικών εξισώσεων. Οι μέθοδοι πεπερασμένων διαφορών (FDM), οι μέθοδοι πεπερασμένων στοιχείων (FEM) και οι μέθοδοι πεπερασμένων όγκων (FVM). Η κατασκευή της πρώτης μεθόδου είναι συνήθως τετριμμένη αλλά είναι δύσκολη η εφαρμογή της σε περίπλοκα χωρία. Στην εργασία αυτή θα χρησιμοποιήσουμε την δεύτερη μέθοδο, που εφαρμόζεται εύκολα σε χωρία οποιασδήποτε γεωμετρίας σε όλες τις διαστάσεις.

1.1 Βασικές έννοιες

Θα ξεκινήσουμε αναφέροντας ορισμένους ορισμούς που θα φανούν χρήσιμοι στη συνέχεια.

Ορισμός 1.1.1. Έστω $\Omega \subset \mathbb{R}^d$ ένα ανοικτό υποσύνολο του $\mathbb{R}^d$ για $d > 1$ (που καλείται διάσταση) και έστω $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ ένα διάνυσμα στο Ω. Έστω μια άγνωστη συνάρτηση $u : \Omega \to \mathbb{R}$ η οποία έχει μερικές παραγώγους έως και k τάξης (με k θετικό ακέραιο). Μια μερική διαφορική εξίσωση k τάξης στο Ω στις d διαστάσεις είναι μια εξίσωση της μορφής

$$F(\mathbf{x}, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots, \frac{\partial u}{\partial x_d}, \frac{\partial^2 u}{\partial x_1^2}, \ldots, \frac{\partial^k u}{\partial x_{d-1} \partial x_d^{k-1}}) = 0, \tag{1.1}$$

με την F να είναι δοθείσα συνάρτηση.

Ορισμός 1.1.2. Έστω μια συνάρτηση $u : \Omega \to \mathbb{R}$, όπου $\Omega \subset \mathbb{R}^d$ ένα ανοικτό σύνολο. Ορίζουμε ως *στήριγμα* της u να είναι η κλειστότητα του συνόλου $\{\mathbf{x} \in \Omega : u(\mathbf{x}) \neq 0\}$. Με άλλα λόγια, το στήριγμα μιας συνάρτησης είναι το μικρότερο κλειστό σύνολο που περιέχει τα στοιχεία του πεδίου ορισμού με μη-μηδενική εικόνα.

Ορισμός 1.1.3. Έστω $1 \leq p < \infty$ και $(\Omega, \mathcal{F}, \mu)$ ένας χώρος μέτρου. Ορίζουμε τον *χώρο Lebesgue* $L^p(\Omega)$ να απαρτίζεται από όλες τις μετρήσιμες συναρτήσεις $u : \Omega \to \mathbb{R}$ για τις οποίες

$$\int_\Omega |u|^p \, \mathrm{d}\mu < \infty. \tag{1.2}$$

Η L^p νόρμα της $u \in L^p(\Omega)$ είναι η ποσότητα

$$\|u\|_{L^p(\Omega)} := \left(\int_\Omega |u|^p \, \mathrm{d}\mu \right)^{\frac{1}{p}}. \tag{1.3}$$

Στη συνέχεια θα μας απασχολήσει μόνο ο χώρος L^2 που είναι ο μόνος χώρος Hilbert από όλους τους L^p. Σε αυτό τον χώρο ορίζουμε το εσωτερικό γινόμενο

$$\langle u, v \rangle_{L^2(\Omega)} := \int_\Omega uv \, \mathrm{d}\mu. \tag{1.4}$$

Ορισμός 1.1.4. Έστω $\Omega \subset \mathbb{R}^d$ ένα ανοικτό σύνολο. Θα συμβολίζουμε με $C_0^\infty(\Omega)$ το σύνολο των συναρτήσεων $\varphi : \Omega \to \mathbb{R}$ που είναι απείρως παραγωγίσιμες και έχουν συμπαγές (κλειστό και φραγμένο) στήριγμα.

Ο χώρος $C_0^\infty(\Omega)$ χρησιμοποιείται για τη γενίκευση της έννοιας της παραγώγου μιας συνάρτησης.

Ορισμός 1.1.5 (Ασθενής παράγωγος). Έστω $\Omega \subset \mathbb{R}^d$ ένα ανοικτό διάστημα και έστω u και v συναρτήσεις στον χώρο $L^1_{loc}(\Omega)$ (ο χώρος των τοπικά ολοκληρώσιμων συναρτήσεων, δηλαδή των συναρτήσεων που είναι ολοκληρώσιμες σε κάθε συμπαγές υποσύνολο του πεδίου ορισμού τους). Η συνάρτηση $v : \Omega \to \mathbb{R}$ θα καλείται *α-ασθενής παράγωγος* της συνάρτησης $u : \Omega \to \mathbb{R}$ αν

$$\int_\Omega u D^a \varphi = (-1)^{|\alpha|} \int_\Omega v\varphi \quad \forall \varphi \in C_0^\infty(\Omega), \tag{1.5}$$

όπου $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d)$ είναι ένας πολυδείκτης από μη-αρνητικούς ακέραιους, $|\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_d$ και

$$D^a \varphi = \frac{\partial^{|\alpha|} \varphi}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}}.$$

Τέλος, χρησιμοποιώντας τους χώρους Lebesgue θα ορίσουμε τους χώρους Sobolev.

Ορισμός 1.1.6. Έστω $\Omega \subset \mathbb{R}^d$ ένα ανοικτό σύνολο, $k \in \mathbb{N} \cup \{0\}$ και $1 \leq p < \infty$. Ορίζουμε τον *χώρο Sobolev* $W_p^k(\Omega)$ ως εξής:

$$W_p^k(\Omega) := \{ u \in L^p(\Omega) : D^\alpha u \in L^p(\Omega) \ \forall \, |\alpha| \leq k \}. \tag{1.6}$$

Ορίζουμε επίσης την αντίστοιχη νόρμα $\| \cdot \|_{W_p^k(\Omega)}$ και ημινόρμα $| \cdot |_{W_p^k(\Omega)}$ ως:

$$\|u\|_{W_p^k(\Omega)} := \left(\sum_{|\alpha| \leq k} \|D^\alpha u\|^p_{L^p(\Omega)} \right)^{\frac{1}{p}} \tag{1.7}$$

$$|u|_{W_p^k(\Omega)} := \left(\sum_{|\alpha| = k} \|D^\alpha u\|^p_{L^p(\Omega)} \right)^{\frac{1}{p}}. \tag{1.8}$$

Για $p = 2$ θα χρησιμοποιούμε τον συμβολισμό $W_p^k(\Omega) \equiv H^k(\Omega)$. Όπως και με τον χώρο L^2, οι χώροι H^k εφοδιασμένοι με το εσωτερικό γινόμενο

$$\langle u, v \rangle_{H^k(\Omega)} := \sum_{|\alpha| \leq k} \langle D^\alpha u, D^\alpha v \rangle_{L^2(\Omega)} \tag{1.9}$$

αποτελούν χώρους Hilbert. Για $k = 0, p = 2$ έχουμε $H^0(\Omega) \equiv L^2(\Omega)$. Τέλος θα συμβολίζουμε με $H_0^1(\Omega)$ τον χώρο

$$H_0^1(\Omega) := \left\{ u \in H^1(\Omega) : u = 0 \text{ στο } \partial\Omega \right\}. \tag{1.10}$$

1.2 Το πρόβλημα Poisson στις δύο διαστάσεις και η κατασκευή της μεθόδου πεπερασμένων στοιχείων

Έστω το πρόβλημα

$$\begin{aligned} -\Delta u &= f \text{ στο } \Omega \\ u &= 0 \text{ στο } \partial\Omega, \end{aligned} \tag{1.11}$$

όπου Δ είναι ο Λαπλασιανός τελεστής που ορίζεται ως $\Delta f = \nabla^2 f = \nabla \cdot \nabla f$ και $\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n} \right)$. Θα μετατρέψουμε το πρόβλημα στην αντίστοιχη ασθενή μορφή. Πολλαπλασιάζουμε την εξίσωση με $v \in \mathcal{H} = H_0^1(\Omega)$ και ολοκληρώνουμε στο Ω.

$$-\int_\Omega \Delta u v \, \mathrm{d}V = \int_\Omega f v \, \mathrm{d}V, \tag{1.12}$$

και χρησιμοποιώντας το θεώρημα απόκλισης έχουμε

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}V - \oint_{\partial\Omega} v \nabla u \cdot \vec{n} \, \mathrm{d}S = \int_\Omega f v \, \mathrm{d}V, \tag{1.13}$$

όπου $\vec{n}$ είναι το μοναδιαίο διάνυσμα, κάθετο στο σύνορο, με φορά έξω από το χωρίο. Όμως $v = 0$ στο $\partial\Omega$ και συνεπώς η *ασθενής μορφή* του προβλήματος (1.11) είναι

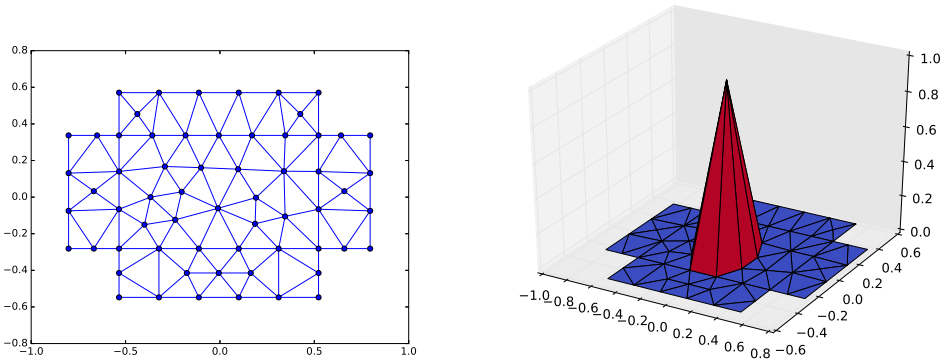
$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}V = \int_\Omega f v \, \mathrm{d}V \qquad \forall v \in \mathcal{H}. \tag{1.14}$$

Το δεύτερο στάδιο είναι να ορίσουμε έναν χώρο πεπερασμένης διάστασης, που θα μας επιτρέψει να λύσουμε τη διαφορική εξίσωση προσεγγιστικά. Χωρίζουμε το Ω σε τρίγωνα $\tau \in \mathcal{T}$, με το $\mathcal{T}$ να συμβολίζει το σύνολο των τριγώνων της διαμέρισης (πλέγμα), όπως φαίνεται στο σχήμα 1.1. Τις κορυφές των τριγώνων θα τις αποκαλούμε από εδώ και στο εξής κόμβους της τριγωνοποίησης. Επιπλέον, ορίζουμε την παράμετρο h ως:

$$h = \max_{\tau \in \mathcal{T}} \{\mathrm{diam}(\tau)\},$$

με την διάμετρο κάθε τριγώνου να είναι το μήκος της μεγαλύτερης πλευράς του.

Πλέον ορίζουμε μια μικρότερη οικογένεια χώρου συναρτήσεων $\mathcal{H}_h \subset \mathcal{H}$ που περιλαμβάνει τις συναρτήσεις που είναι συνεχείς στο Ω, πολυώνυμα $1^{\text{ου}}$ βαθμού σε κάθε τρίγωνο και παίρνουν την τιμή 0 στο $\partial\Omega$. Αριθμώντας αυθαίρετα τους



Σχήμα 1.1: Αριστερά μια τριγωνοποίηση ενός χωρίου στο επίπεδο και δεξιά η συνάρτηση βάσης (πυραμίδα) που αντιστοιχεί σε έναν εσωτερικό κόμβο.

κόμβους της τριγωνοποίησης που δεν ανήκουν στο σύνορο ως $1, \ldots, N$, κάθε συνάρτηση $w_h \in \mathcal{H}_h$ μπορεί να γραφτεί ως γραμμικός συνδυασμός των συναρτήσεων βάσης φ_i, με κάθε φ_i να παίρνει την τιμή 1 στον κόμβο i και την τιμή 0 στους υπόλοιπους κόμβους της τριγωνοποίησης:

$$w_h = \sum_{i=1}^{N} W_i \varphi_i$$

Πλέον έχουμε το ακόλουθο προσεγγιστικό πρόβλημα: να βρεθεί $u_h \in \mathcal{H}_h$ τέτοια ώστε

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, \mathrm{d}V = \int_\Omega f v_h \, \mathrm{d}V \qquad \forall v_h \in \mathcal{H}_h. \tag{1.15}$$

Όμως κάθε v_h, αφού ανήκει στον $\mathcal{H}_h$, μπορεί να γραφτεί ως γραμμικός συνδυασμός των συναρτήσεων βάσης φ_i, οπότε έχουμε το ακόλουθο πρόβλημα: να βρεθεί $u_h \in \mathcal{H}_h$ τέτοια ώστε

$$\int_\Omega \nabla u_h \cdot \nabla \varphi_i \, \mathrm{d}V = \int_\Omega f \varphi_i \, \mathrm{d}V \qquad \forall i = 1, \ldots, N. \tag{1.16}$$

Ακόμα $u_h \in \mathcal{H}_h$, οπότε μπορεί να γραφτεί ως $u_h = \sum_{j=1}^{N} U_j \varphi_j$ για κάποια άγνωστα $U_j \in \mathbb{R}$, συνεπώς πλέον αναζητούμε $U_j \in \mathbb{R}$, $j = 1, \ldots, N$ τέτοια ώστε

$$\sum_{j=1}^{N} U_j \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, \mathrm{d}V = \int_\Omega f \varphi_i \, \mathrm{d}V \qquad \forall i = 1, \ldots, N. \tag{1.17}$$

Αυτό πλέον είναι ένα γραμμικό σύστημα N εξισώσεων με N αγνώστους και μπορεί να γραφτεί στη μορφή $A\boldsymbol{U} = \boldsymbol{F}$, για $A = [\alpha_{ij}]_{i,j=1}^{N}$, $\boldsymbol{U} = (U_1, \ldots, U_N)^\mathsf{T}$ και $\boldsymbol{F} = (F_1, \ldots, F_N)^\mathsf{T}$, όπου

$$\alpha_{ij} = \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, \mathrm{d}V \text{ και } F_i = \int_\Omega f \varphi_i \, \mathrm{d}V$$

Αυτό το γραμμικό σύστημα μπορούμε να το λύσουμε με τη μέθοδο απαλοιφής Gauss ή κάποια επαναληπτική μέθοδο για να υπολογίσουμε την προσεγγιστική λύση $\boldsymbol{U}$ της μερικής διαφορικής εξίσωσης, με το $U_j \in \mathbb{R}$ να είναι η τιμή της συνάρτησης στον κόμβο j.

1.3 Ανάλυση σφάλματος στην ενεργειακή νόρμα

Ορισμός 1.3.1. Έστω V ένας διανυσματικός χώρος πάνω στο βαθμωτό σώμα $\mathbb{F}$. Μια απεικόνιση $B : V \times V \to \mathbb{F}$ καλείται *διγραμμική μορφή* πάνω στον χώρο V αν είναι γραμμική για κάθε όρισμα χωριστά, δηλαδή αν $\forall\, u, v, w \in V$ και $\forall\, \lambda, \mu \in \mathbb{F}$ ισχύουν τα εξής

i. $B(\lambda u + \mu v, w) = \lambda B(u, w) + \mu B(v, w)$

ii. $B(u, \lambda v + \mu w) = \lambda B(u, v) + \mu B(u, w)$

Αν επιπλέον ισχύει $B(u, v) = B(v, u)\ \forall\, u, v \in V$ τότε η διγραμμική μορφή B καλείται *συμμετρική*.

Για το πρόβλημα (1.14) ορίζουμε τη διγραμμική μορφή $\alpha : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ ως:

$$\alpha(u, v) = \int_\Omega \nabla u \cdot \nabla u \, \mathrm{d}V. \tag{1.18}$$

Αυτή η διγραμμική μορφή είναι προφανώς συμμετρική, $\alpha(u, v) = \alpha(v, u), \forall u, v \in \mathcal{H}$. Η γραμμικότητα προκύπτει εύκολα από την γραμμικότητα του τελεστή κλίσης. Θα δείξουμε ότι ορίζει *εσωτερικό γινόμενο* στον χώρο $\mathcal{H}$. Πράγματι υποθέτοντας ότι $\alpha(u, u) = 0$ έχουμε ότι $\nabla u = \mathbf{0}$ και άρα η u είναι σταθερή συνάρτηση. Όμως $u \in \mathcal{H}$ οπότε αφού $u = 0$ στο $\partial\Omega$ έχουμε ότι $u = 0$ σε όλο το Ω.

Ορισμός 1.3.2. Έστω η συμμετρική διγραμμική μορφή $\alpha : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$. Ορίζουμε την *ενεργειακή νόρμα*

$$\|u\|_E := \sqrt{\alpha(u, u)} = \left(\int_\Omega \nabla u \cdot \nabla u \, \mathrm{d}V \right)^{\frac{1}{2}}. \tag{1.19}$$

Η ανωτέρω νόρμα είναι η φυσική νόρμα για το πρόβλημα (1.14) και σε αυτή θα εκτιμήσουμε τα σφάλματα προσέγγισης για τη μέθοδο πεπερασμένων στοιχείων.

Ο χώρος $\mathcal{H}_h$ είναι υπόχωρος του $\mathcal{H}$ και συνεπώς η σχέση (1.14) ισχύει και για κάθε $v_h \in \mathcal{H}_h$

$$\alpha(u, v_h) = \int_\Omega \nabla u \cdot \nabla v_h \, \mathrm{d}V = \int_\Omega f v_h \, \mathrm{d}V = \langle f, v_h \rangle_{L^2(\Omega)} \quad \forall v_h \in \mathcal{H}_h. \tag{1.20}$$

Αφαιρώντας την παραπάνω σχέση από την (1.15) έχουμε

$$\alpha(u - u_h, v_h) = \alpha(u, v_h) - \alpha(u_h, v_h) = \langle f, v_h \rangle - \langle f, v_h \rangle = 0 \quad \forall v_h \in \mathcal{H}_h. \tag{1.21}$$

Η σχέση αυτή, που είναι γνωστή ως *Galerkin orthogonality*, μας λέει ότι η u_h είναι η προβολή, ως προς την ενεργειακή νόρμα, της ασθενούς λύσης u πάνω στον χώρο $\mathcal{H}_h$. Θα την χρησιμοποιήσουμε για να δείξουμε το επόμενο λήμμα.

Λήμμα 1.3.1 (Céa's lemma). Από όλες τις συναρτήσεις του χώρου $\mathcal{H}_h$, η λύση u_h της μεθόδου πεπερασμένων στοιχείων που προκύπτει από την (1.15) είναι η καλύτερη προσέγγιση της ασθενούς λύσης u, ως προς την ενεργειακή νόρμα (1.19). Δηλαδή,

$$\|u - u_h\|_E \leq \|u - v_h\|_E \qquad \forall v_h \in \mathcal{H}_h. \tag{1.22}$$

Απόδειξη. θα χρησιμοποιήσουμε δύο φορές τη σχέση (1.21) και την ανισότητα Cauchy–Schwarz [1].

$$\begin{aligned}
\|u - u_h\|_E^2 &= \alpha(u - u_h, u - u_h) \\
&= \alpha(u - u_h, u) - \alpha(u - u_h, u_h) \\
&= \alpha(u - u_h, u) \\
&= \alpha(u - u_h, u) - \alpha(u - u_h, v_h) \\
&= \alpha(u - u_h, u - v_h) \\
&\leq \|u - u_h\|_E \|u - v_h\|_E \qquad \forall v_h \in \mathcal{H}_h.
\end{aligned}$$

□

Τέλος θα αναφέρουμε, χωρίς να αποδείξουμε, την εκτίμηση για το σφάλμα της μεθόδου πεπερασμένων στοιχείων. Υποθέτοντας ότι $u \in H^k(\Omega) \cap H_0^1(\Omega)$ και ο βαθμός των πολυωνυμικών βάσεων είναι p τότε για την προσεγγιστική λύση u_h ισχύει η σχέση:

$$\|u - u_h\|_E = \|\nabla(u - u_h)\|_{L^2} \leq Ch^{s-1}|u|_{H^s(\Omega)}, \quad s = \min\{k, p+1\}, \tag{1.23}$$

με τη σταθερά C να είναι ανεξάρτητη των h και u. Βλέπουμε δηλαδή ότι αν η u είναι αρκούντως ομαλή η μέθοδος θα συγκλίνει, ως προς την ενεργειακή νόρμα, με τάξη ίδια με τον βαθμό των βάσεων που θα επιλέξουμε.

1.4 Το θεώρημα Lax-Milgram

Αναζητώντας τη λύση μιας μερικής διαφορικής εξίσωσης με τη μέθοδο πεπερασμένων στοιχείων θα θέλαμε να γνωρίζουμε αν αυτή υπάρχει και αν είναι μοναδική. Θα αναφέρουμε και θα αποδείξουμε ένα σημαντικό θεώρημα της συναρτησιακής ανάλυσης που θα μας βοηθήσει προς αυτή την κατεύθυνση.

Θεώρημα 1. Έστω H ένας χώρος Hilbert και $B : H \times H \to \mathbb{R}$ μια διγραμμική μορφή. Υποθέτουμε ότι υπάρχουν θετικές σταθερές c_1 και c_2 έτσι ώστε

i. $|B(u, v)| \leq c_1\|u\|\|v\| \quad \forall u, v \in H$ (bounded)

ii. $|B(u, u)| \geq c_2\|u\|^2 \quad \forall u \in H$ (coercive)

Τότε για κάθε $f \in H^*$ υπάρχει ένα μοναδικό $v \in H$ τέτοιο ώστε

$$B(u, v) = f(u) \quad \forall u \in H. \tag{1.24}$$

[1] Για κάθε u, v σε έναν διανυσματικό χώρο με εσωτερικό γινόμενο ισχύει ότι $|\langle u, v\rangle| \leq \|u\|\|v\|$, με την νόρμα στο δεξί μέλος να είναι η επαγόμενη νόρμα από το εσωτερικό γινόμενο του χώρου.

Απόδειξη. Σταθεροποιούμε $v \in H$. Τότε η αντιστοίχηση

$$u \mapsto B(u, v),$$

είναι φραγμένο γραμμικό συναρτησιακό στον H, δηλαδή ένα στοιχείο του H^*. Από το θεώρημα αναπαράστασης του Riesz έχουμε ότι υπάρχει μοναδικό $w \in H$ τέτοιο ώστε

$$B(u, v) = (u, w) \quad \forall u \in H.$$

Αφού για κάθε $v \in H$ μπορούμε να βρούμε ένα μοναδικό $w \in H$ μπορούμε να ορίσουμε έναν τελεστή $A : H \to H$ ως $A(v) = w$. Δείχνοντας ότι ο A είναι αντιστρέψιμος θα έχουμε αποδείξει και την μοναδικότητα της λύσης για την (1.24).

Έστω $f \in H^*$ και έστω $w \in H$ το μοναδικό στοιχείο τέτοιο ώστε $f(u) = (u, w)\, \forall u \in H$ (από το θεώρημα αναπαράστασης του Riesz και πάλι). Τότε έαν για κάθε $w \in H$ μπορούμε να βρούμε ένα μοναδικό $v \in H$ έτσι ώστε $A(v) = w$ τότε το v είναι η μοναδική λύση της

$$B(u, v) = (u, A(v)) = (u, w) = f(u) \quad \forall u \in H.$$

Θα γράψουμε ορισμένες ιδιότητες του A και θα καταλήξουμε ότι ο A είναι αντιστρέψιμος.

1. Ο A είναι γραμμικός τελεστής.

$$\begin{aligned}(u, A(a_1 v_1 + a + 2v_2)) &= B(u, a_1 v_1 + a + 2v_2) \\ &= a_1 B(u, v_1) + a_2 B(u, v_2) \\ &= a_1 (u, A(v_1)) + a_2 (u, A(v_2)) \\ &= (u, a_1 A(v_1) + a_2 A(v_2)).\end{aligned}$$

   Αφού αυτό ισχύει για τυχόντα u, a_i, v_i ο A είναι γραμμικός.

2. Ο A είναι φραγμένος (συνεχής).

$$\|A(v)\|^2 = (A(v), A(v)) = B(A(v), v) \le c_1 \|A(v)\| \|v\| \Rightarrow$$
$$\|A(v)\| \le c_1 \|v\| \quad \forall v \in H.$$

3. Ο A είναι φραγμένος από κάτω.

$$c_2 \|v\|^2 \le B(v, v) = (v, A(v)) \le \|v\| \|A(v)\| \Rightarrow$$
$$\|A(v)\| \ge c_2 \|v\| \quad \forall v \in H.$$

4. Αφού ο A είναι συνεχής και φραγμένος από κάτω άρα είναι και ένα προς ένα και το $R(A)$, δηλαδή η εικόνα του είναι κλειστό σύνολο.

5. Το $R(A)$ είναι πυκνό στον H.

   Εάν $v \in R(A)^\perp$ τότε

$$c_2 \|v\|^2 \le B(v, v) = (v, A(v)) = 0 \text{ αφού } v \perp R(A).$$

Έχουμέ ότι το $R(A)$ είναι κλειστό ($R(A) = \overline{R(A)}$) και πυκνό ($\overline{R(A)} = H$) υποσύνολο του H συνεπώς $R(A) = H$. Έπεται ότι ο A είναι ένα προς ένα και επί και άρα αντιστρέψιμος. □

Κεφάλαιο 2

Η υλοποίηση της μεθόδου πεπερασμένων στοιχείων

Είδαμε ότι για την υλοποίηση της μεθόδου πεπερασμένων στοιχείων είναι απαραίτητο να υπολογίσουμε τις βάσεις πάνω στους κόμβους και ολοκληρώματα πάνω σε κάθε τρίγωνο της διαμέρισης. Για λόγους που έχουν να κάνουν με την υλοποίηση της μεθόδου στον υπολογιστή επιλέγουμε να χρησιμοποιήσουμε έναν μετασχηματισμό από τα τρίγωνα του πλέγματος σε ένα τρίγωνο αναφοράς. Με αυτό τον τρόπο ο υπολογισμός των βάσεων καθώς και των σημείων για την αριθμητική ολοκλήρωση θα γίνει μόνο μία φορά, για το τρίγωνο αναφοράς και χρησιμοποιώντας τον μετασχηματισμό θα τα μεταφέρουμε κατάλληλα σε κάθε τρίγωνο του πλέγματος.

2.1 Μετασχηματισμός στο τρίγωνο αναφοράς

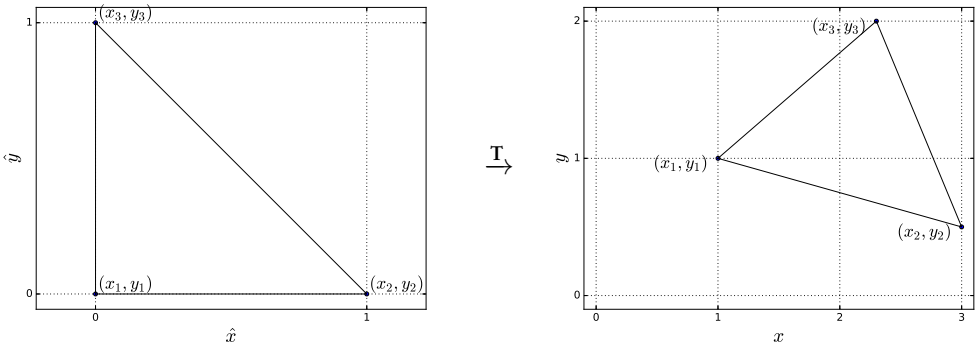
Θέλουμε να μεταφέρουμε τους υπολογισμούς των ολοκληρωμάτων από κάθε στοιχείο της τριγωνοποίησης σε ένα τρίγωνο στην αρχή των αξόνων. Θα βρούμε έναν μετασχηματισμό $\mathbf{T} : \hat{\tau} \to \tau$, όπου $\hat{\tau}$ το τρίγωνο του επιπέδου με κορυφές στα σημεία $(0,0)$, $(1,0)$ και $(0,1)$ και τ ένα τυχαίο τρίγωνο του επιπέδου με συστεταγμένες (x_1, y_1), (x_2, y_2) και (x_3, y_3), έτσι ώστε

$$\begin{aligned}\mathbf{T}(0,0) &= (x_1, y_1)\\ \mathbf{T}(1,0) &= (x_2, y_2)\\ \mathbf{T}(0,1) &= (x_3, y_3).\end{aligned}$$

Ένας τέτοιος γεωμετρικός μετασχηματισμός μπορεί να αναπαρασταθεί με πίνακες, συνεπώς θα βρούμε $b_{11}, b_{12}, b_{21}, b_{22}, c_1, c_2$ ώστε

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \tag{2.2}$$



Σχήμα 2.1: Μετασχηματισμός από το τρίγωνο αναφοράς.

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} \tag{2.3}$$

Από την (2.1) έχουμε

$$\begin{aligned} c_1 &= x_1 \\ c_2 &= y_1 \end{aligned}$$

Από τις (2.2) και (2.3) έχουμε

$$\begin{aligned} b_{11} + c_1 = x_2 \quad &\Rightarrow \quad b_{11} = x_2 - x_1 \\ b_{21} + c_2 = y_2 \quad &\Rightarrow \quad b_{21} = y_2 - y_1 \\ b_{12} + c_1 = x_3 \quad &\Rightarrow \quad b_{12} = x_3 - x_1 \\ b_{22} + c_2 = y_3 \quad &\Rightarrow \quad b_{22} = y_3 - y_1 \end{aligned}$$

Και συνεπώς ο μετασχηματισμός είναι

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \cdot \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \Rightarrow$$

$$\mathbf{x} = \mathbf{T}(\hat{\mathbf{x}}) = B\hat{\mathbf{x}} + C,$$

όπου

$$\begin{aligned} \mathbf{x} &= (x, y) \\ \hat{\mathbf{x}} &= (\hat{x}, \hat{y}) \\ B &= \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \\ C &= \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \end{aligned}$$

Ο μετασχηματισμός αυτός είναι αντιστρέψιμος

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = B^{-1} \begin{bmatrix} x - x_1 \\ y - y_1 \end{bmatrix}$$

Έστω $\hat{u}$ μια συνάρτηση βάσης στο τρίγωνο αναφοράς και u η αντίστοιχη συνάρτηση βάσης σε ένα τυχαίο τρίγωνο του επιπέδου. Θέλουμε

$$\begin{aligned}\hat{u}(\hat{x},\hat{y}) &= u(x,y)\\ &= u(\mathbf{T}(\hat{x},\hat{y}))\\ &= u(T_1(\hat{x},\hat{y}),T_2(\hat{x},\hat{y}))\\ &= u\circ\mathbf{T},\end{aligned}$$

όπου

$$\begin{aligned}T_1(\hat{x},\hat{y}) &= (x_2-x_1)\hat{x}+(x_3-x_1)\hat{y}+x_1\\ T_2(\hat{x},\hat{y}) &= (y_2-y_1)\hat{x}+(y_3-y_1)\hat{y}+y_1\end{aligned}$$

Για την Ιακωβιανή του μετασχηματισμού έχουμε

$$\boldsymbol{J}_{\mathbf{T}}=\begin{bmatrix}\frac{\partial T_1}{\partial\hat{x}} & \frac{\partial T_1}{\partial\hat{y}}\\ \frac{\partial T_2}{\partial\hat{x}} & \frac{\partial T_2}{\partial\hat{y}}\end{bmatrix}=\begin{bmatrix}x_2-x_1 & x_3-x_1\\ y_2-y_1 & y_3-y_1\end{bmatrix}$$

Το διάνυσμα της κλίσης της συνάρτησης βάσης στο τρίγωνο αναφοράς είναι

$$\hat{\nabla}\hat{u}(\hat{x},\hat{y})=\hat{\nabla}u(T_1(\hat{x},\hat{y}),T_2(\hat{x},\hat{y}))=\begin{bmatrix}\frac{\partial u}{\partial\hat{x}}\\ \frac{\partial y}{\partial\hat{y}}\end{bmatrix}$$

και χρησιμοποιώντας τον τύπο για την παραγώγιση σύνθετης συνάρτησης παίρνουμε

$$\begin{aligned}\frac{\partial u}{\partial\hat{x}} &= \frac{\partial u}{\partial T_1}\cdot\frac{\partial T_1}{\partial\hat{x}}+\frac{\partial u}{\partial T_2}\cdot\frac{\partial T_2}{\partial\hat{x}}\\ &= \frac{\partial u}{\partial x}\cdot(x_2-x_1)+\frac{\partial u}{\partial y}\cdot(y_2-y_1)\end{aligned}$$

$$\begin{aligned}\frac{\partial u}{\partial\hat{y}} &= \frac{\partial u}{\partial T_1}\cdot\frac{\partial T_1}{\partial\hat{y}}+\frac{\partial u}{\partial T_2}\cdot\frac{\partial T_2}{\partial\hat{y}}\\ &= \frac{\partial u}{\partial x}\cdot(x_3-x_1)+\frac{\partial u}{\partial y}\cdot(y_3-y_1).\end{aligned}$$

Συνεπώς

$$\begin{aligned}\hat{\nabla}\hat{u}(\hat{x},\hat{y}) &= \begin{bmatrix}\frac{\partial u}{\partial x}\cdot(x_2-x_1)+\frac{\partial u}{\partial y}\cdot(y_2-y_1)\\ \frac{\partial u}{\partial x}\cdot(x_3-x_1)+\frac{\partial u}{\partial y}\cdot(y_3-y_1)\end{bmatrix}\\ &= \begin{bmatrix}x_2-x_1 & y_2-y_1\\ x_3-x_1 & y_3-y_1\end{bmatrix}\cdot\begin{bmatrix}\frac{\partial u}{\partial x}\\ \frac{\partial u}{\partial y}\end{bmatrix}\Rightarrow\end{aligned}$$

$$\begin{aligned}\hat{\nabla}\hat{u}(\hat{x},\hat{y}) &= B^{\mathsf{T}}\nabla u(x,y)\Rightarrow\\ \nabla u(x,y) &= B^{-\mathsf{T}}\hat{\nabla}\hat{u}(\hat{x},\hat{y}).\end{aligned}$$

2.2 Συναρτήσεις βάσης για το τρίγωνο αναφοράς

Αφού έχουμε μεταφέρει τους υπολογισμούς στο τρίγωνο αναφοράς θέλουμε να κατασκευάσουμε τις συναρτήσεις βάσης. Έστω ότι θέλουμε να παράξουμε μέσα στο τρίγωνο αναφοράς τα πολυώνυμα δύο μεταβλητών βαθμού έως και n.

$$p_n(x,y) = \sum_{k=0}^{n} a_k x^i y^j \qquad i+j \leq k$$

Τότε, επειδή έχουμε $m = \frac{(n+1)(n+2)}{2}$ βαθμούς ελευθερίας θα χρειαστεί να ορίσουμε το ίδιο πλήθος από συναρτήσεις βάσης. Έστω (x_q, y_q), $q = 1, \ldots, m$ σημεία του τριγώνου. Ορίζουμε τις m το πλήθος συναρτήσεις βάσης

$$\varphi_{n,p}(x,y) = \sum_{k=1}^{n} a_{k,p} x^i y^j \qquad i+j \leq k$$

ως εξής

$$\varphi_{n,p}(x_q, y_q) = \begin{cases} 1 & p = q \\ 0 & p \neq q \end{cases} \qquad p, q = 1, \ldots, m$$

2.2.1 Γραμμικές συναρτήσεις βάσης

Για να ορίσουμε με μοναδικό τρόπο μια γραμμική συνάρτηση σε ένα τρίγωνο χρειαζόμαστε 3 σημεία. Θα χρησιμοποιήσουμε τις 3 κορυφές του τριγώνου αναφοράς (Σχήμα 2.2). Αριθμούμε τους κόμβους ως εξής

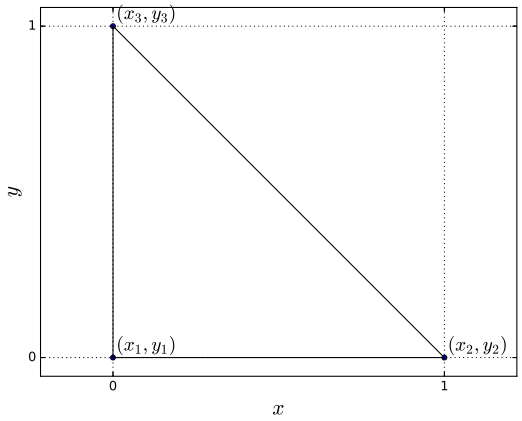
$$(x_1, y_1) = (0, 0)$$
$$(x_2, y_2) = (1, 0)$$
$$(x_3, y_3) = (0, 1)$$

και θα βρούμε a_i, b_i και c_i τέτοια ώστε τα γραμμικά πολυώνυμα $\varphi_i(x,y) = a_i x + b_i y + c_i$, $i = 1, 2, 3$ να παίρνουν τις τιμές

$$\varphi_i(x_j, y_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad i, j = 1, 2, 3$$

Θέλουμε δηλαδή

$$\varphi_1(0,0) = 1 \Rightarrow c_1 = 1$$
$$\varphi_1(1,0) = 0 \Rightarrow a_1 + c_1 = 0 \Rightarrow a_1 = -1$$
$$\varphi_1(0,1) = 0 \Rightarrow b_1 + c_1 = 0 \Rightarrow b_1 = -1$$



Σχήμα 2.2: Τα 3 σημεία στο τρίγωνο αναφοράς για την κατασκευή των γραμμικών συναρτήσεων βάσης

$$\varphi_2(0,0) = 0 \Rightarrow c_2 = 0$$
$$\varphi_2(1,0) = 1 \Rightarrow a_2 + c_2 = 1 \Rightarrow a_2 = 1$$
$$\varphi_2(0,1) = 0 \Rightarrow b_2 + c_2 = 0 \Rightarrow b_2 = 0$$

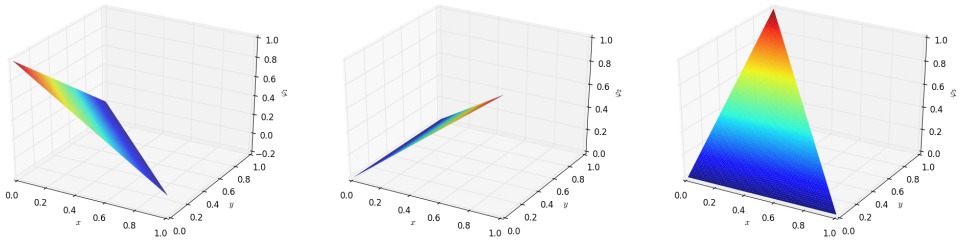
$$\varphi_3(0,0) = 0 \Rightarrow c_3 = 0$$
$$\varphi_3(1,0) = 0 \Rightarrow a_3 + c_3 = 0 \Rightarrow a_3 = 0$$
$$\varphi_3(0,1) = 1 \Rightarrow b_3 + c_3 = 1 \Rightarrow b_3 = 1$$

και συνεπώς οι 3 συναρτήσεις βάσης για τα πολυώνυμα $1^{ου}$ βαθμού στο τρίγωνο αναφοράς είναι

$$\varphi_1(x,y) = 1 - x - y$$
$$\varphi_2(x,y) = x$$
$$\varphi_3(x,y) = y$$

και η κλίση τους δίνεται από τους τύπους

$$\nabla\varphi_1(x,y) = \left(\frac{\partial\varphi_1(x,y)}{\partial x}, \frac{\partial\varphi_1(x,y)}{\partial y}\right) = (-1,-1)$$
$$\nabla\varphi_2(x,y) = \left(\frac{\partial\varphi_2(x,y)}{\partial x}, \frac{\partial\varphi_2(x,y)}{\partial y}\right) = (1,0)$$
$$\nabla\varphi_3(x,y) = \left(\frac{\partial\varphi_3(x,y)}{\partial x}, \frac{\partial\varphi_3(x,y)}{\partial y}\right) = (0,1).$$



Σχήμα 2.3: Γραμμικές συναρτήσεις βάσης στο τρίγωνο αναφοράς.

2.2.2 Τετραγωνικές συναρτήσεις βάσης

Για να ορίσουμε με μοναδικό τρόπο μια τετραγωνική συνάρτηση σε ένα τρίγωνο χρειαζόμαστε 6 σημεία. Θα χρησιμοποιήσουμε τις 3 κορυφές του τριγώνου αναφοράς καθώς και τα 3 σημεία στα μέσα των πλευρών (Σχήμα 2.4). Αριθμούμε τους κόμβους ως εξής

$$
\begin{aligned}
(x_1, y_1) &= (0, 0) \qquad & (x_4, y_4) &= \left(\frac{1}{2}, 0\right) \\
(x_2, y_2) &= (1, 0) \qquad & (x_5, y_5) &= \left(\frac{1}{2}, \frac{1}{2}\right) \\
(x_3, y_3) &= (0, 1) \qquad & (x_6, y_6) &= \left(0, \frac{1}{2}\right)
\end{aligned}
$$

και θα βρούμε a, b, c, d, e και f τέτοια ώστε τα τετραγωνικά πολυώνυμα

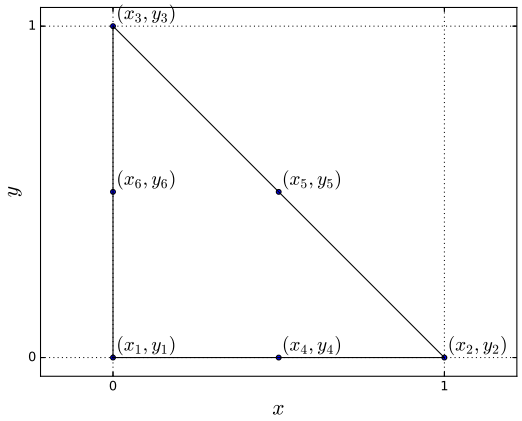
$$\varphi_i(x, y) = a_i x^2 + b_i xy + c_i y^2 + d_i x + e_i y + f_i \qquad i = 1, \ldots, 6$$

να παίρνουν τις τιμές

$$\varphi_i(x_j, y_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad i, j = 1, \ldots, 6$$

Λύνοντας τα γραμμικά συστήματα που προκύπτουν βρίσκουμε ότι οι 6 συναρτήσεις βάσης για τα πολυώνυμα $2^{ου}$ βαθμού στο τρίγωνο αναφοράς είναι

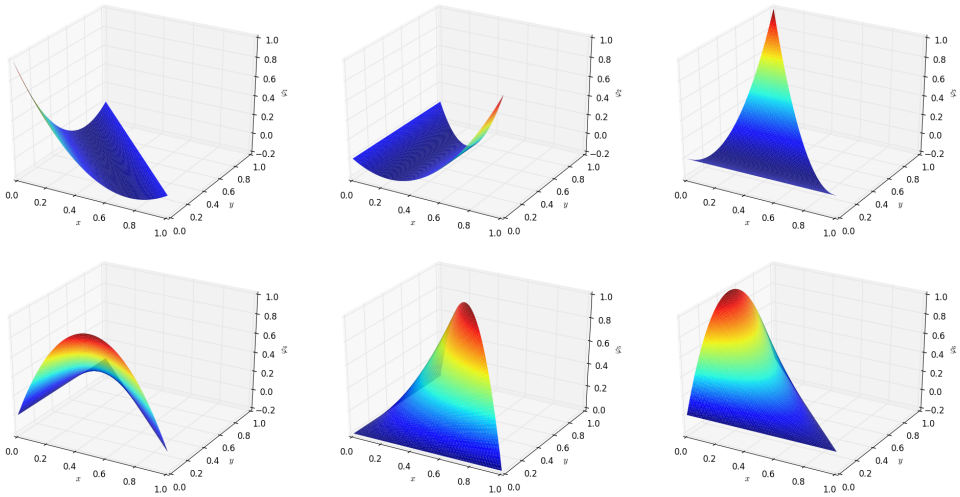
$$
\begin{aligned}
\varphi_1(x, y) &= 2x^2 + 4xy + 2y^2 - 3x - 3y + 1 \\
\varphi_2(x, y) &= 2x^2 - x \\
\varphi_3(x, y) &= 2y^2 - y \\
\varphi_4(x, y) &= -4x^2 - 4xy + 4x \\
\varphi_5(x, y) &= 4xy \\
\varphi_6(x, y) &= -4y^2 - 4xy + 4y
\end{aligned}
$$



Σχήμα 2.4: Τα 6 σημεία στο τρίγωνο αναφοράς για την κατασκευή των τετραγωνικών συναρτήσεων βάσης

και η κλίση τους δίνεται από τους τύπους

$$\begin{aligned}
\nabla\varphi_1(x,y) &= (4x+4y-3, 4x+4y-3)\\
\nabla\varphi_2(x,y) &= (4x-1, 0)\\
\nabla\varphi_3(x,y) &= (0, 4y-1)\\
\nabla\varphi_4(x,y) &= (-8x-4y+4, -4x)\\
\nabla\varphi_5(x,y) &= (4y, 4x)\\
\nabla\varphi_6(x,y) &= (-4y, -4x-8y+4)
\end{aligned}$$



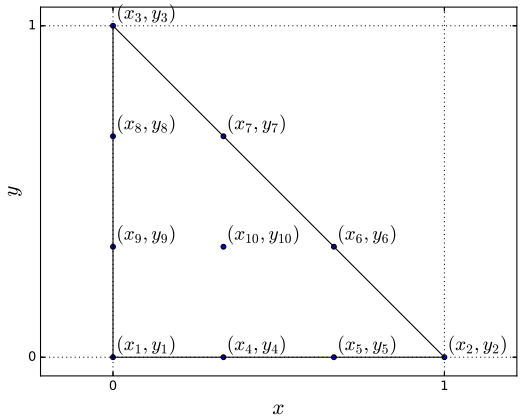
Σχήμα 2.5: Τετραγωνικές συναρτήσεις βάσης στο τρίγωνο αναφοράς.

2.2.3 Κυβικές συναρτήσεις βάσης

Για να ορίσουμε με μοναδικό τρόπο μια κυβική συνάρτηση σε ένα τρίγωνο χρειαζόμαστε 10 σημεία. Θα χρησιμοποιήσουμε τις 3 κορυφές του τριγώνου αναφοράς, 2 ισαπέχοντα σημεία σε κάθε πλευρά καθώς και το βαρύκεντρο (Σχήμα 2.6). Αριθμούμε τους κόμβους ως εξής

$$\begin{array}{lll} (x_1, y_1) = (0, 0) & (x_5, y_5) = \left(\frac{2}{3}, 0\right) & (x_9, y_9) = \left(0, \frac{1}{3}\right) \\ (x_2, y_2) = (1, 0) & (x_6, y_6) = \left(\frac{2}{3}, \frac{1}{3}\right) & (x_{10}, y_{10}) = \left(\frac{1}{3}, \frac{1}{3}\right) \\ (x_3, y_3) = (0, 1) & (x_7, y_7) = \left(\frac{1}{3}, \frac{2}{3}\right) & \\ (x_4, y_4) = \left(\frac{1}{3}, 0\right) & (x_8, y_8) = \left(0, \frac{2}{3}\right) & \end{array}$$

και θα βρούμε συντελεστές έτσι ώστε τα κυβικά πολυώνυμα



Σχήμα 2.6: Τα 10 σημεία στο τρίγωνο αναφοράς για την κατασκευή των κυβικών συναρτήσεων βάσης

$$\varphi_i(x, y) = \sum_{k=0}^{3} a_{k,i} x^p y^q \qquad p + q \leq k,\ i = 1, \ldots, 10$$

να παίρνουν τις τιμές

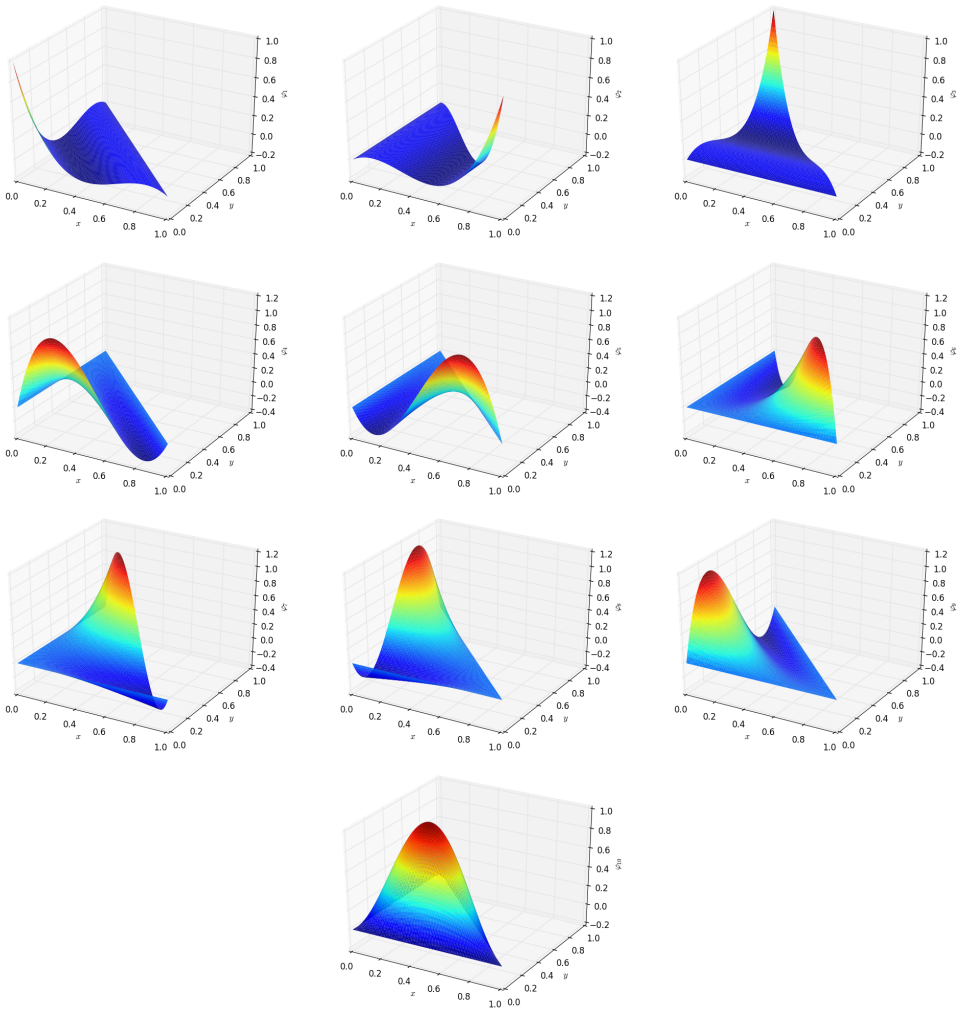
$$\varphi_i(x_j, y_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad i, j = 1, \ldots, 10$$

Λύνοντας τα γραμμικά συστήματα που προκύπτουν βρίσκουμε ότι οι 10 συναρτήσεις βάσης για τα πολυώνυμα $3^{ου}$ βαθμού στο τρίγωνο αναφοράς είναι

$$\begin{aligned}
\varphi_1(x,y) &= -\frac{9}{2}x^3 - \frac{27}{2}x^2y - \frac{27}{2}xy^2 - \frac{9}{2}y^3 + 9x^2 \\
&\qquad + 18xy + 9y^2 - \frac{11}{2}x - \frac{11}{2}y + 1 \\
\varphi_2(x,y) &= \frac{9}{2}x^3 - \frac{9}{2}x^2 + x \\
\varphi_3(x,y) &= \frac{9}{2}y^3 - \frac{9}{2}y^2 + y \\
\varphi_4(x,y) &= \frac{27}{2}x^3 + 27x^2y + \frac{27}{2}xy^2 - \frac{45}{2}x^2 - \frac{45}{2}xy + 9x \\
\varphi_5(x,y) &= -\frac{27}{2}x^3 - \frac{27}{2}x^2y + 18x^2 + \frac{9}{2}xy - \frac{9}{2}x \\
\varphi_6(x,y) &= \frac{27}{2}x^2y - \frac{9}{2}xy \\
\varphi_7(x,y) &= \frac{27}{2}xy^2 - \frac{9}{2}xy \\
\varphi_8(x,y) &= -\frac{27}{2}xy^2 - \frac{27}{2}y^3 + \frac{9}{2}xy + 18y^2 - \frac{9}{2}y \\
\varphi_9(x,y) &= \frac{27}{2}x^2y + 27xy^2 + \frac{27}{2}y^3 - \frac{45}{2}xy - \frac{45}{2}y^2 + 9y \\
\varphi_{10}(x,y) &= -27x^2y - 27xy^2 + 27xy
\end{aligned}$$

και η κλίση τους δίνεται από τους τύπους

$$\nabla\varphi_1(x,y) = \left(-\frac{27}{2}x^2 - 27xy + 18x - \frac{27}{2}y^2 + 18y - \frac{11}{2}, \right.$$
$$\left. -\frac{27}{2}x^2 - 27xy + 18x - \frac{27}{2}y^2 + 18y - \frac{11}{2}\right)$$
$$\nabla\varphi_2(x,y) = \left(\frac{27}{2}x^2 - 9x + 1, 0\right)$$
$$\nabla\varphi_3(x,y) = \left(0, \frac{27}{2}y^2 - 9y + 1\right)$$
$$\nabla\varphi_4(x,y) = \left(\frac{81}{2}x^2 + 54xy - 45x + \frac{27}{2}y^2 - \frac{45}{2}y + 9, 27x^2 + 27xy - \frac{45}{2}x\right)$$
$$\nabla\varphi_5(x,y) = \left(-\frac{81}{2}x^2 - 27xy + 36x + \frac{9}{2}y - \frac{9}{2}, -\frac{27}{2}x^2 + \frac{9}{2}x\right)$$
$$\nabla\varphi_6(x,y) = \left(27xy - \frac{9}{2}y, \frac{27}{2}x^2 - \frac{9}{2}x\right)$$
$$\nabla\varphi_7(x,y) = \left(\frac{27}{2}y^2 - \frac{9}{2}y, 27xy - \frac{9}{2}x\right)$$
$$\nabla\varphi_8(x,y) = \left(-\frac{27}{2}y^2 + \frac{9}{2}y, -27xy + \frac{9}{2}x - \frac{81}{2}y^2 + 36y - \frac{9}{2}\right)$$
$$\nabla\varphi_9(x,y) = \left(27xy + 27y^2 - \frac{45}{2}y, \frac{27}{2}x^2 + 54xy - \frac{45}{2}x + \frac{81}{2}y^2 - 45y + 9\right)$$
$$\nabla\varphi_{10}(x,y) = \left(-54xy - 27y^2 + 27y, -27x^2 - 54xy + 27x\right)$$



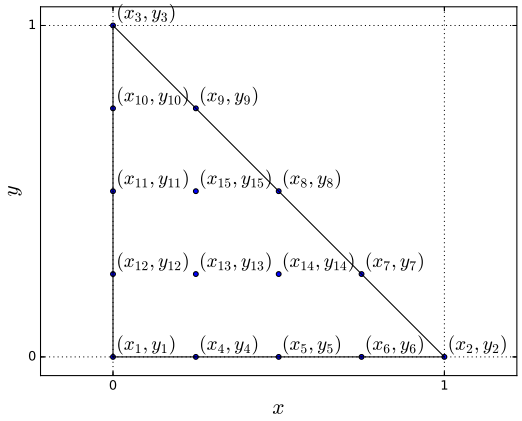
Σχήμα 2.7: Κυβικές συναρτήσεις βάσης στο τρίγωνο αναφοράς.

2.2.4 Συναρτήσεις βάσης $4^{ου}$ βαθμού

Για να ορίσουμε με μοναδικό τρόπο μια πολυωνυμική συνάρτηση $4^{ου}$ βαθμού σε ένα τρίγωνο χρειαζόμαστε 15 σημεία. Θα χρησιμοποιήσουμε τις 3 κορυφές του τριγώνου αναφοράς, 3 ισαπέχοντα σημεία σε κάθε πλευρά καθώς και 3 σημεία στο εσωτερικό του τριγώνου (Σχήμα 2.8). Αριθμούμε τους κόμβους ως εξής

$$
\begin{array}{lll}
(x_1, y_1) = (0, 0) & (x_6, y_6) = \left(\frac{3}{4}, 0\right) & (x_{11}, y_{11}) = \left(0, \frac{1}{2}\right) \\
(x_2, y_2) = (1, 0) & (x_7, y_7) = \left(\frac{3}{4}, \frac{1}{4}\right) & (x_{12}, y_{12}) = \left(0, \frac{1}{4}\right) \\
(x_3, y_3) = (0, 1) & (x_8, y_8) = \left(\frac{1}{2}, \frac{1}{2}\right) & (x_{13}, y_{13}) = \left(\frac{1}{4}, \frac{1}{4}\right) \\
(x_4, y_4) = \left(\frac{1}{4}, 0\right) & (x_9, y_9) = \left(\frac{1}{4}, \frac{3}{4}\right) & (x_{14}, y_{14}) = \left(\frac{1}{2}, \frac{1}{4}\right) \\
(x_5, y_5) = \left(\frac{1}{2}, 0\right) & (x_{10}, y_{10}) = \left(0, \frac{3}{4}\right) & (x_{13}, y_{13}) = \left(\frac{1}{4}, \frac{1}{2}\right)
\end{array}
$$

και θα βρούμε συντελεστές έτσι ώστε τα πολυώνυμα



Σχήμα 2.8: Τα 15 σημεία στο τρίγωνο αναφοράς για την κατασκευή των συναρτήσεων βάσης $4^{ου}$ βαθμού

$$\varphi_i(x, y) = \sum_{k=0}^{4} a_{k,i} x^p y^q \qquad p + q \leq k,\ i = 1, \ldots, 15$$

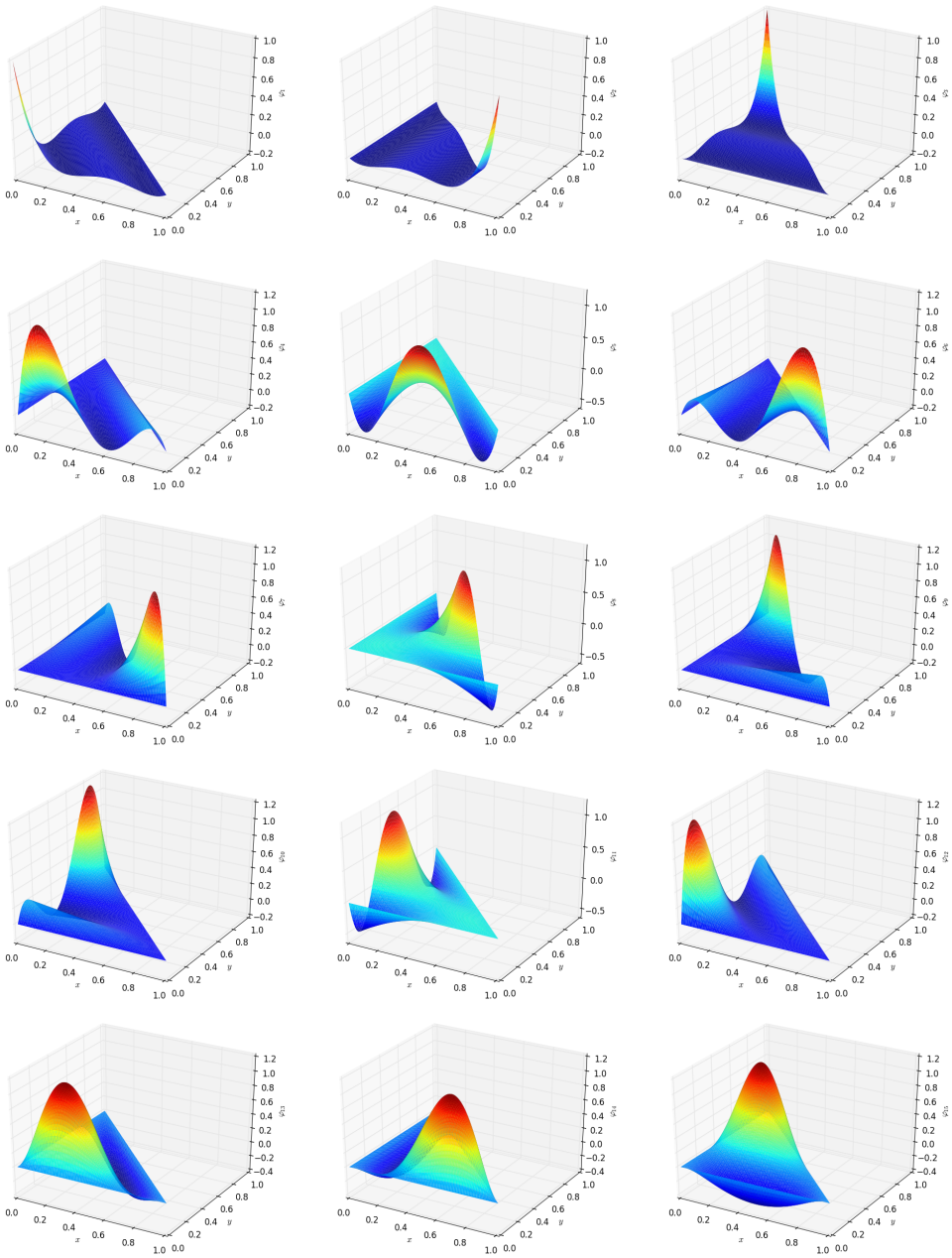
να παίρνουν τις τιμές

$$\varphi_i(x_j, y_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad i, j = 1, \ldots, 15$$

Λύνοντας τα γραμμικά συστήματα που προκύπτουν βρίσκουμε ότι οι 15 συναρτήσεις βάσης για τα πολυώνυμα $4^{ου}$ βαθμού στο τρίγωνο αναφοράς είναι

$$\begin{aligned}
\varphi_1(x,y) &= \frac{32}{3}x^4 + \frac{128}{3}x^3y + 64x^2y^2 + \frac{128}{3}xy^3 + \frac{32}{3}y^4 - \frac{80}{3}x^3 - 80x^2y \\
&\quad - 80xy^2 - \frac{80}{3}y^3 + \frac{70}{3}x^2 + \frac{140}{3}xy + \frac{70}{3}y^2 - \frac{25}{3}x - \frac{25}{3}y + 1 \\
\varphi_2(x,y) &= \frac{32}{3}x^4 - 16x^3 + \frac{22}{3}x^2 - x \\
\varphi_3(x,y) &= \frac{32}{3}y^4 - 16y^3 + \frac{22}{3}y^2 - y \\
\varphi_4(x,y) &= -\frac{128}{3}x^4 - 128x^3y - 128x^2y^2 - \frac{128}{3}xy^3 + 96x^3 + 192x^2y \\
&\quad + 96xy^2 - \frac{208}{3}x^2 - \frac{208}{3}xy + 16x \\
\varphi_5(x,y) &= 64x^4 + 128x^3y + 64x^2y^2 - 128x^3 - 144x^2y - 16xy^2 + 76x^2 + 28xy - 12x \\
\varphi_6(x,y) &= -\frac{128}{3}x^4 - \frac{128}{3}x^3y + \frac{224}{3}x^3 + 32x^2y - \frac{112}{3}x^2 - \frac{16}{3}xy + \frac{16}{3}x \\
\varphi_7(x,y) &= \frac{128}{3}x^3y - 32x^2y + \frac{16}{3}xy \\
\varphi_8(x,y) &= 64x^2y^2 - 16x^2y - 16xy^2 + 4xy \\
\varphi_9(x,y) &= \frac{128}{3}xy^3 - 32xy^2 + \frac{16}{3}xy \\
\varphi_{10}(x,y) &= -\frac{128}{3}xy^3 - \frac{128}{3}y^4 + 32xy^2 + \frac{224}{3}y^3 - \frac{16}{3}xy - \frac{112}{3}y^2 + \frac{16}{3}y \\
\varphi_{11}(x,y) &= 64x^2y^2 + 128xy^3 + 64y^4 - 16x^2y - 144xy^2 - 128y^3 + 28xy + 76y^2 - 12y \\
\varphi_{12}(x,y) &= -\frac{128}{3}x^3y - 128x^2y^2 - 128xy^3 - \frac{128}{3}y^4 + 96x^2y + 192xy^2 \\
&\quad + 96y^3 - \frac{208}{3}xy - \frac{208}{3}y^2 + 16y \\
\varphi_{13}(x,y) &= 128x^3y + 256x^2y^2 + 128xy^3 - 224x^2y - 224xy^2 + 96xy \\
\varphi_{14}(x,y) &= -128x^3y - 128x^2y^2 + 160x^2y + 32xy^2 - 32xy \\
\varphi_{15}(x,y) &= -128x^2y^2 - 128xy^3 + 32x^2y + 160xy^2 - 32xy
\end{aligned}$$

Για λόγους συντομίας παραλείπουμε να γράψουμε τους τύπους της κλίσης των συναρτήσεων βάσης $4^{ου}$ βαθμού.



Σχήμα 2.9: Συναρτήσεις βάσης $4^{ου}$ βαθμού στο τρίγωνο αναφοράς.

2.3 Αριθμητική ολοκλήρωση Gauss

Είδαμε ότι για τους πίνακες του αριστερού μέλους (stiffness και mass matrices) και για το διάνυσμα του δεξιού μέλους του γραμμικού συστήματος έχουμε να υπολογίσουμε ορισμένα ολοκληρώματα. Ο υπολογισμός θα γίνει με κάποια αριθμητική μέθοδο και η μέθοδος που θα επιλέξουμε είναι η Αριθμητική ολοκλήρωση Gauss. Το πλεονέκτημα αυτής της μεθόδου είναι ότι με λίγα σημεία, και συνεπώς λίγες επαναλήψεις στον υπολογιστή, μπορούμε να υπολογίσουμε με μεγάλη ακρίβεια (ή και ακριβώς αν εξαιρέσουμε τα σφάλματα στρογγύλευσης και αποκοπής) όσες συναρτήσεις προσεγγίζονται αρκούντως καλά από πολυώνυμα.

2.3.1 Αριθμητική ολοκλήρωση στο τρίγωνο αναφοράς

Όπως είδαμε στο κεφάλαιο 2.1 μεταφέρουμε τα τρίγωνα της διαμέρισής μας σε ένα τρίγωνο αναφοράς, στην αρχή των αξόνων και συγκεκριμένα στο τρίγωνο με κορυφές στα σημεία $(0,0)$, $(1,0)$ και $(0,1)$.

$$\hat{\tau} = \{(\hat{x},\hat{y}) : 0 \leq \hat{x},\hat{y},\hat{x}+\hat{y} \leq 1\}.$$

Στόχος μας είναι να κατασκευάσουμε μεθόδους της μορφής

$$\iint_{\hat{\tau}} f(\hat{x},\hat{y})\,\mathrm{d}\hat{x}\mathrm{d}\hat{y} \approx \frac{1}{2}\sum_{i=1}^{N_g} w_i f(\hat{x}_i,\hat{y}_i), \tag{2.4}$$

όπου N_g είναι το πλήθος των σημείων της αριθμητικής ολοκλήρωσης, $(\hat{x_i},\hat{y_i})$ είναι τα σημεία μέσα στο τρίγωνο αναφοράς που υπολογίζουμε την τιμή της συνάρτησης f και w_i είναι τα αντίστοιχα βάρη. Παρατηρήστε ότι τα βάρη είναι κανονικοποιημένα ως προς το εμβαδό του τριγώνου αναφοράς, εξού και το $\frac{1}{2}$ στο άθροισμα του δεξιού μέλους. Επιπλέον θα θέλαμε τα σημεία να είναι τα ελάχιστα ώστε να πετύχουμε την απαιτούμενη ακρίβεια και να κατανέμονται συμμετρικά μέσα στο τρίγωνο.

Η επιλογή των σημείων $(\hat{x_i},\hat{y_i})$ και των βαρών w_i θα γίνει έτσι ώστε ο τύπος (2.4) να υπολογίζει ακριβώς πολυώνυμα έως βαθμού n. Θέλουμε δηλαδή

$$\iint_{\hat{\tau}} f(\hat{x},\hat{y})\,\mathrm{d}\hat{x}\mathrm{d}\hat{y} = \frac{1}{2}\sum_{i=1}^{N_g} w_i f(\hat{x}_i,\hat{y}_i) \qquad \forall f(\hat{x},\hat{y}) \in \mathbb{P}_n(\hat{x},\hat{y})$$

όπου $\mathbb{P}_n(\hat{x},\hat{y})$ είναι ο χώρος των πολυωνύμων έως βαθμού n σε δύο διαστάσεις

$$\mathbb{P}_n(\hat{x},\hat{y}) = \mathrm{span}\{\hat{x}^i\hat{y}^j, 0 \leq i,j,i+j \leq n\} \Rightarrow$$

$$\begin{aligned}
\mathbb{P}_1(\hat{x},\hat{y}) &= \mathrm{span}\{1,\hat{x},\hat{y}\}\\
\mathbb{P}_2(\hat{x},\hat{y}) &= \mathrm{span}\{1,\hat{x},\hat{y},\hat{x}^2,\hat{x}\hat{y},\hat{y}^2\}\\
&\vdots
\end{aligned}$$

Σημειώνουμε ότι

$$\dim(\mathbb{P}_n) = \frac{(n+1)(n+2)}{2}$$

Ένας χρήσιμος τύπος για τον υπολογισμό των ολοκληρωμάτων των στοιχείων βάσης του χώρου πολυωνύμων μέσα στο τρίγωνο αναφοράς είναι

$$\iint_{\hat{\tau}} \hat{x}^i \hat{y}^j \, \mathrm{d}\hat{x}\mathrm{d}\hat{y} = \frac{i!j!}{(i+j+2)!}$$

Συνεπώς έχουμε

$$\iint_{\hat{\tau}} \{1, \hat{x}, \hat{y}, \hat{x}^2, \hat{x}\hat{y}, \hat{y}^2 \ \dots\} \, \mathrm{d}\hat{x}\mathrm{d}\hat{y} = \left\{ \frac{1}{2}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}, \frac{1}{12}, \dots \right\}$$

Τύποι ολοκλήρωσης 1ης τάξης

Θέλουμε οι τύποι να υπολογίζουν ακριβώς το ολοκλήρωμα για τις συναρτήσεις $f(\hat{x}, \hat{y}) = 1$, $\hat{x}$ και $\hat{y}$.

$$\begin{aligned}
f(\hat{x}, \hat{y}) = 1 \quad &\longrightarrow \quad \frac{1}{2} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \\
f(\hat{x}, \hat{y}) = \hat{x} \quad &\longrightarrow \quad \frac{1}{6} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{x}_i \\
f(\hat{x}, \hat{y}) = \hat{y} \quad &\longrightarrow \quad \frac{1}{6} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{y}_i.
\end{aligned}$$

Χρησιμοποιώντας ένα σημείο, δηλαδή $N_g = 1$ η λύση που παίρνουμε είναι $w_1 = 1$ και $\hat{x}_1 = \hat{y}_1 = \frac{1}{3}$ δηλαδή το βαρύκεντρο του τριγώνου.

Τύποι ολοκλήρωσης 2ης τάξης

Θέλουμε οι τύποι να υπολογίζουν ακριβώς το ολοκλήρωμα για τις συναρτήσεις $f(\hat{x}, \hat{y}) = 1$, $\hat{x}$, $\hat{y}$, $\hat{x}^2$, $\hat{x}\hat{y}$ και $\hat{y}^2$.

$$\begin{aligned}
f(\hat{x}, \hat{y}) = 1 \quad &\longrightarrow \quad \frac{1}{2} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \\
f(\hat{x}, \hat{y}) = \hat{x} \quad &\longrightarrow \quad \frac{1}{6} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{x}_i \\
f(\hat{x}, \hat{y}) = \hat{y} \quad &\longrightarrow \quad \frac{1}{6} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{y}_i \\
f(\hat{x}, \hat{y}) = \hat{x}^2 \quad &\longrightarrow \quad \frac{1}{12} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{x}_i^2 \\
f(\hat{x}, \hat{y}) = \hat{x}\hat{y} \quad &\longrightarrow \quad \frac{1}{24} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{x}_i \hat{y}_i \\
f(\hat{x}, \hat{y}) = \hat{y}^2 \quad &\longrightarrow \quad \frac{1}{12} = \frac{1}{2} \sum_{i=1}^{N_g} w_i \hat{y}_i^2.
\end{aligned}$$

Χρησιμοποιώντας ένα σημείο, δηλαδή $N_g = 1$ έχουμε 6 εξισώσεις με 3 αγνώστους και το σύστημα δεν έχει λύση. Θεωρητικά το $N_g = 2$ θα μπορούσε να δώσει λύση, αφού θα είχαμε 6 εξισώσεις με 6 αγνώστους όμως η λύση που προκύπτει δεν είναι συμμετρική. Επιλέγοντας $N_g = 3$ έχουμε 9 αγνώστους και περισσότερες από μία λύσεις. Η λύση που θα χρησιμοποιήσουμε είναι

$$
\begin{aligned}
w_1 = w_2 = w_3 &= \frac{1}{3} \\
(\hat{x}_1, \hat{y}_1) &= \left(\frac{1}{6}, \frac{1}{6}\right) \\
(\hat{x}_2, \hat{y}_2) &= \left(\frac{1}{6}, \frac{2}{3}\right) \\
(\hat{x}_3, \hat{y}_3) &= \left(\frac{2}{3}, \frac{1}{6}\right).
\end{aligned}
$$

Οι πίνακες των κόμβων και βαρών για όλους τους τύπους αριθμητικής ολοκλήρωσης Gauss που θα χρησιμοποιήσουμε σε αυτή την εργασία βρίσκονται στο Παράρτημα.

Κεφάλαιο 3

Το μοντέλο προγραμματισμού CUDA

3.1 Εισαγωγή

Για πολλά χρόνια οι κάρτες γραφικών απευθύνονταν σε επαγγελματίες που ασχολούνται με επεξεργασία εικόνας και βίντεο καθώς και σε gamers. Η επιστημονική κοινότητα και όσοι χρειάζονταν μεγάλη επεξεργαστική ισχύ ήταν αναγκασμένοι να χρησιμοποιούν μεγάλα υπολογιστικά συστήματα με χιλιάδες επεξεργαστές, συνδεδεμένα μέσω δικτύου, χρησιμοποιώντας ένα μοντέλο όπως το MPI για να μοιράσουν τον φόρτο εργασίας και για την ανταλλαγή δεδομένων από κόμβο σε κόμβο. Όποιος ήθελε να χρησιμοποιήσει την κάρτα γραφικών για επιστημονικούς υπολογισμούς έπρεπε να μάθει ένα από τα προγραμματιστικά περιβάλλοντα για γραφικά, όπως το DirectX ή το OpenGL και να μετατρέψει κατάλληλα τον κώδικά του.

Σχήμα 3.1: *Αριστερά:* Η κάρτα γραφικών GeForce GTX 1060. *Δεξιά:* Η κάρτα γραφικών Tesla P100. Και οι δύο υποστηρίζουν το CUDA Compute Capability 6.x. Παρατηρούμε ότι από την δεύτερη κάρτα απουσιάζουν οι θύρες εξόδου. Έχουμε δηλαδή μια κάρτα γραφικών που δεν συνδέεται σε οθόνη αλλα χρησιμοποιείται μόνο για επεξεργασία. (©Nvidia).

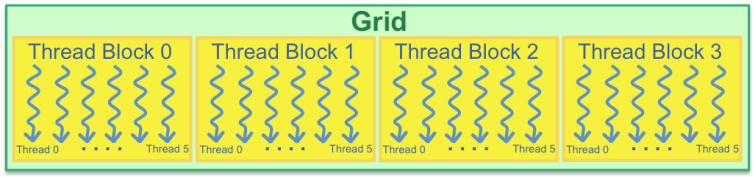
Το 2006 η Nvidia εισήγαγε μια νέα παράλληλη πλατφόρμα υπολογισμού, το CUDA που έδωσε τη δυνατότητα προγραμματισμού για τις κάρτες γραφικών της σε γλώσσες C, C++ ή Fortran. Το CUDA υλοποιεί το μοντέλο SIMT (Single Instruction, Multiple Threads). Σύμφωνα με αυτό μια ρουτίνα που στο περιβάλλον CUDA ονομάζεται kernel, τρέχει παράλληλα σε χιλιάδες threads, με το κάθε thread να αναλαμβάνει να εκτελέσει την ίδια εργασία σε ένα διαφορετικό σύνολο δεδομένων. Τα threads οργανώνονται σε blocks και τα blocks σε ένα grid. Κάθε block, που περιέχει το πολύ 1024 threads, εκτελείται ταυτόχρονα στην κάρτα γραφικών και ολοκληρώνεται μόνο όταν όλα τα threads του block έχουν ολοκληρώσει την εργασία τους. Αυτός είναι και ο λόγος που η οργάνωση των threads πρέπει να γίνεται με τέτοιο τρόπο ώστε όσα ανήκουν στο ίδιο block να απαιτούν παρόμοιο αριθμό πράξεων για την ολοκλήρωσή τους.

Ένα πρόγραμμα σε CUDA μοιάζει πάρα πολύ με το αντίστοιχο σειριακό. Έστω για παράδειγμα ότι μας δίνεται μια λίστα με 256 αριθμούς και μας ζητείται να υπολογίσουμε το τετράγωνό τους. Ένας σειριακός αλγόριθμος σε C θα μοιάζει ως εξής

```
void square(float* out, float* in){
  for (i=0; i<255; i++){
    out[i] = in[i]*in[i]
  }
}
```

Θα είχαμε δηλαδή ένα for loop που θα διέτρεχε όλα τα στοιχεία της λίστας εισόδου, θα υπολόγιζε το τετράγωνο του αριθμού και θα το αποθήκευε στην αντίστοιχη θέση στη λίστα εξόδου. Το ίδιο πρόγραμμα γραμμένο σε CUDA C θα έμοιαζε ως εξής

```
__global__ void square(float* d_out, float* d_in){
  int idx = threadIdx.x
  d_out[i] = d_in[i]*d_in[i]
}
```



Σχήμα 3.2: Η ιεραρχική δομή των threads, blocks και grid για την μονοδιάσταστη περίπτωση. Μπορούμε να δημιουργήσουμε και δομές σε περισσότερες διαστάσεις αν αυτό εξυπηρετεί το πρόβλημά μας. (©Oak Ridge National Laboratory).

Βλέπουμε ότι δεν υπάρχει πλέον το for loop αλλά έχει αντικατασταθεί από την εντολή της δεύτερης γραμμής. Η εντολή αυτή μας επιστρέφει το index του thread. Κάθε thread γνωρίζει τον αριθμό του μέσα στο block καθώς επίσης και τον αριθμό του block στο οποίο ανήκει. Έτσι κάθε ένα από τα 256 threads θα αναλάβει να υπολογίσει το τετράγωνο ενός μόνο από τους αριθμούς της λίστας και όλοι οι υπολογισμοί θα γίνουν ταυτόχρονα και ανεξάρτητα. Αν ο υπολογισμός και η αποθήκευση ενός μόνο στοιχείου στην CPU απαιτούσε 2ns τότε ο σειριακός κώδικας θα χρειαζόταν 512ns για να ολοκλήρωθεί. Αν ο αντίστοιχος υπολογισμός

για κάθε thread στην GPU απαιτούσε 20 φορές περισσότερο χρόνο, δηλαδή 40ns τότε ο υπολογισμός των τετραγώνων όλων των στοιχείων στην GPU θα απαιτούσε μόλις 40ns αφού θα γινόταν ταυτόχρονα για όλα τα στοιχεία της λίστας.

3.2 Παράλληλος αλγόριθμος για τη μέθοδο πεπερασμένων στοιχείων

Πριν κατασκευάσουμε τους πίνακες του γραμμικού συστήματος θα πρέπει να κατασκευάσουμε το πλέγμα που περιγράφει το χωρίο. Η κατασκευή του πλέγματος έγινε με τη βοήθεια του PDETool της Matlab. Τα αρχεία που χρειαζόμαστε είναι τρία και αφορούν τη γεωμετρία των κόμβων, την τοπολογία των τριγώνων και την τοπολογία του συνόρου. Για το πλέγμα του σχήματος 3.3 βλέπουμε τους τρεις πίνακες που το περιγράφουν πλήρως. Ο πίνακας *nodes*, που περιγράφει τη γεωμετρία των κόμβων, περιλαμβάνει στην i-γραμμή τις συντεταγμένες (x, y) του i-οστού κόμβου. Ο πίνακας *elements*, που περιγράφει την τοπολογία των τριγώνων, περιλαμβάνει στην j-γραμμή τρεις ακεραίους, που αντιστοιχούν στους δείκτες των τριών κόμβων από τον πίνακα nodes, οι οποίοι ανήκουν στο συγκεκριμένο τρίγωνο. Η σειρά με την οποία δίνονται είναι αντιωρολογιακή. Τέλος ο πίνακας *boundary*, που περιγράφει την τοπολογία των ακμών του συνόρου, περιλαβάνει σε κάθε γραμμή τους δείκτες των δύο κόμβων που ανήκουν σε αυτή την ακμή.

Στη συνέχεια θα υπολογίσουμε την τιμή των συναρτήσεων βάσης για τα σημεία της ολοκλήρωσης Gauss στο τρίγωνο αναφοράς και θα τα μεταφέρουμε στη μνήμη της κάρτας γραφικών μαζί με τους πίνακες της τριγωνοποίησης, τους κόμβους και τα βάρη του κανόνα ολοκλήρωσης.

id	x	y
1	0.0	0.0
2	1.0	0.0
3	1.0	1.0
4	0.0	1.0
5	0.5	0.5

nodes

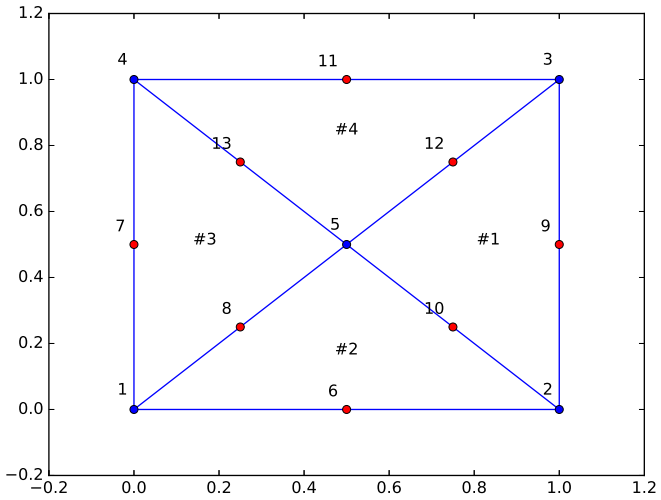
2	3	5
1	2	5
1	5	4
4	5	3

elements

2	3
1	2
4	1
3	4

boundary

Πίνακας 3.1: Οι τρεις πίνακες για την τριγωνοποίηση του σχήματος 3.3.



Σχήμα 3.3: Ένα χωρίο του επιπέδου και μία τριγωνοποίησή του. Με μπλε οι αρχικοί κόμβοι και με κόκκινο οι επιπλέον κόμβοι που θα δημιουργηθούν για την επίλυση με συναρτήσεις βάσης $2^{ου}$ βαθμού.

3.2.1 Κατασκευή νέων κόμβων

Στην περίπτωση που θα χρησιμοποιήσουμε γραμμικές συναρτήσεις βάσης μπορούμε να συνεχίσουμε γιατί αυτοί οι τρεις πίνακες περιλαμβάνουν όλες τις πληροφορίες που θα χρειαστούμε. Στην περίπτωση όμως που θα χρησιμοποιήσουμε συναρτήσεις βάσης ανώτερης τάξης θα πρέπει να υπολογίσουμε και τους κόμβους πάνω στις ακμές ή/και στο εσωτερικό των τριγώνων και να τροποποιήσουμε κατάλληλα τους παραπάνω πίνακες. Η διαδικάσία είναι η εξής:

1. Παράλληλα, χρησιμοποιώντας την GPU, θα κατασκευάσουμε τόσα theads όσο και το πλήθος των τριγώνων της διακριτοποίησης μας και κάθε thread θα γράψει σε τρεις συνεχόμενες θέσεις μιας λίστας (χρησιμοποιώντας το id που είναι μοναδικό για κάθε thread) τους δύο δείκτες των κόμβων που ανήκουν σε κάθε μια από τις τρεις ακμές. Για λόγους που θα γίνουν κατανοητοί στη συνέχεια, για κάθε ακμή, οι δύο κόμβοι που ανήκουν σε αυτή θα τοποθετηθούν στη λίστα έτσι ώστε ο δείκτης του πρώτου να είναι μικρότερος από τον δείκτη του δεύτερου κόμβου. Όπως είναι προφανές, οι ακμές που ανήκουν στο σύνορο θα εισαχθούν συνολικά από μία φορά ενώ οι εσωτερικές ακμές δύο φορές η κάθε μία, αφού κάθε εσωτερική ακμή ανήκει σε δύο τρίγωνα.

2. Στη συνέχεια θα ταξινομήσουμε τον ανωτέρω πίνακα δύο φορές, αρχικά ως προς τον πρώτο κόμβο και στη συνέχεια ως προς τον δεύτερο. Η διαδικασία αυτή, έχει πολυπλοκότητα $\mathcal{O}(n \log n)$, όπου n το μέγεθος του πίνακα. Πλέον, σε γραμμικό χρόνο χρόνο, ως προς το πλήθος των ακμών, μπορούμε να αφαιρέσουμε τις διπλές ακμές.

2	3
3	5
2	5
1	2
2	5
1	5
1	5
4	5
1	4
4	5
3	5
3	4

1	2
1	4
1	5
1	5
2	3
2	5
2	5
3	4
3	5
3	5
4	5
4	5

id		
1	1	2
2	1	4
3	1	5
4	2	3
5	2	5
6	3	4
7	3	5
8	4	5

Πίνακας 3.2: *Αριστερά:* Οι ακμές της τριγωνοποίησης. Οι εσωτερικές ακμές υπάρχουν από δύο φορές. *Κέντρο*: Οι ακμές της τριγωνοποίησης ταξινομημένες. *Δεξιά*: Οι ακμές μετά την αφαίρεση των διπλών καθώς και το id τους.

3. Παράλληλα, στη GPU, δημιουργούμε τόσα threads όσο και το πλήθος των ακμών της τριγωνοποίησης και για κάθε ακμή δημιουργούμε τους νέους κόμβους για τις μεθόδους υψηλότερης τάξης. Επεκτείνουμε τον παλιό πίνακα nodes προσθέτοντας στο τέλος τους νέους κόμβους και σε μια ξεχωριστή λίστα κρατάμε την αντιστοίχηση node-edge, δηλαδή στην i-οστή θέση αυτής της λίστας βάζουμε το index του πίνακα nodes στο οποίο βρίσκεται ο i-οστός νέος κόμβος.

id	x	y
⋮	⋮	⋮
6	0.50	0.00
7	0.00	0.50
8	0.25	0.25
9	1.00	0.50
10	0.75	0.25
11	0.50	1.00
12	0.75	0.75
13	0.25	0.75

id edge	id node
1	6
2	7
3	8
4	9
5	10
6	11
7	12
8	13

Πίνακας 3.3: *Αριστερά:* Οι νέοι κόμβοι για τις μεθόδους υψηλότερης τάξης. *Δεξιά*: Ο πίνακας αντιστοίχησης node-edge.

4. Παράλληλα, για κάθε τρίγωνο και για κάθε ακμή του τριγώνου βρίσκουμε με δυαδική αναζήτηση (πολυπλοκότητα $\mathcal{O}(\log n)$) το index της ακμής από τον πίνακα ακμών που κατασκευάσαμε προηγουμένως και χρησιμοποιώντας τον πίνακα αντιστοίχησης node-edge επεκτείνουμε τον πίνακα elements προσθέτοντας το index των νέων κόμβων στα αντίστοιχα τρίγωνα (με τη σωστή φορά). Για μεθόδους $3^{\text{ης}}$ και $4^{\text{ης}}$ τάξης δημιουργούμε και τους

κόμβους στο εσωτερικό των τριγώνων και τους προσθέτουμε στους κατάλληλους πίνακες.

2	3	5	9	12	10
1	2	5	6	10	8
1	5	4	8	13	7
4	5	3	13	12	11

Πίνακας 3.4: Ο τελικός πίνακας elements για την ανωτέρω τριγωνοποίηση και για την μέθοδο πεπερασμένων στοιχείων $2^{\text{ης}}$ τάξης. Για κάθε τρίγωνο, εκτός από τους αρχικούς κόμβους διακρίνουμε και τους νέους κόμβους στο μέσο των ακμών.

3.2.2 Κατασκευή του γραμμικού συστήματος

Είδαμε ότι για την κατασκευή του γραμμικού συστήματος έχουμε να υπολογίσουμε τα εξής ολοκληρώματα

$$\alpha_{ij} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \,\mathrm{d}V \quad \text{και} \quad F_i = \int_\Omega f\varphi_i \,\mathrm{d}V$$

όπου φ_i και φ_i οι συναρτήσεις βάσης που αντιστοιχούν στους κόμβους i και j αντίστοιχα. Όμως τα τρίγωνα που σχηματίζουν το χωρίο Ω δεν έχουν επικαλύψεις και συνεπώς έχουμε

$$\alpha_{ij} = \sum_{\tau\in\mathcal{T}} \int_\tau \nabla\varphi_j \cdot \nabla\varphi_i \,\mathrm{d}\mathbf{x}$$

Για κάθε τρίγωνο $\tau \in \mathcal{T}$ μπορούμε να κατασκευάσουμε τον τοπικό πίνακα stiffness που θα περιλαμβάνει την συνεισφορά στο στοιχείο α_{ij} μόνο από το συγκεκριμένο τρίγωνο. Όταν ολοκληρωθεί η κατασκευή όλων των τοπικών πινάκων θα κατασκευάσουμε τον ολικό πίνακα αθροίζοντας τα κοινά στοιχεία. Για παράδειγμα, για την τριγωνοποίηση του σχήματος 3.3 θα έχουμε $\alpha_{58} = \alpha_{58}^2 + \alpha_{58}^3$ με τον άνω δείκτη να δείχνει το τρίγωνο από τον οποίο προήλθε το αντίστοιχο τοπικό στοιχείο. Ο αλγόριθμος είναι ο εξής (Για λόγους που έχουν να κάνουν με την αναγνωσιμότητα του κώδικα θα χρησιμοποιήσουμε κώδικα γραμμένο σε Python. Ο κώδικας σε CUDA C μπορεί να βρεθεί στο Παράρτημα.)

1. Θα δημιουργήσουμε τόσα threads όσο και το πλήθος των τριγώνων της τριγωνοποίησής μας. Όπως ήδη είδαμε κάθε thread θα έχει ένα μοναδικό id που θα αντιστοιχεί στο id του τριγώνου για το οποίο θα αναλάβει να κατασκευάσει και να αποθηκεύσει, ανεξάρτητα από τα υπόλοιπα threads, τον τοπικό πίνακα stiffness.

2. Κάθε thread θα υπολογίσει τους πίνακες του μετασχηματισμού από το κεφάλαιο 2.1

```
1 def affine(node1, node2, node3):
2     B = np.array([[node2[0] - node1[0],
3                    node3[0] - node1[0]],
```

```
                  [node2[1] - node1[1],
                   node3[1] - node1[1]]])
    C = np.array([node1[0], node1[1]])
    invtranspB = linalg.inv(B.T)
    detB = math.fabs(linalg.det(B))
    return B, C, invtranspB, detB
```

3. Για τον υπολογισμό των ολοκληρωμάτων θα χρησιμοποιήσουμε τον τύπο

$$\begin{aligned}\int_{\tau} \nabla\varphi_j \cdot \nabla\varphi_i \,\mathrm{d}\mathbf{x} &= \int_{\hat{\tau}} (B^{-\mathsf{T}}\hat{\nabla}\hat{\varphi}_j) \cdot (B^{-\mathsf{T}}\hat{\nabla}\hat{\varphi}_i) |\det(B)| \,\mathrm{d}\hat{\mathbf{x}} \\ &= \frac{|\det(B)|}{2} \sum_{1\le k\le m} w_k (B^{-\mathsf{T}}\hat{\nabla}\hat{\varphi}_j(\hat{x}_k,\hat{y}_k)) \cdot (B^{-\mathsf{T}}\hat{\nabla}\hat{\varphi}_i(\hat{x}_k,\hat{y}_k))\end{aligned}$$

όπου $\hat{\varphi}_i$ η συνάρτηση βάσης στο τρίγωνο αναφοράς που αντιστοιχεί στον κόμβο i του αρχικού τριγώνου, m το πλήθος των σημείων που χρησιμοποιούμε για την ολοκλήρωση Gauss, $(\hat{x}_k, \hat{y}_k)$ τα αντίστοιχα σημεία μέσα στο τρίγωνο αναφοράς και B ο πίνακας του μετασχηματισμού από το κεφάλαιο 2.1.

```
def localstiffness(gradphi, weights, invtranspB,
                   detB, i, j):
    area = 0.5 * detB
    gradphii = np.asarray(gradphi[i])
    gradphij = np.asarray(gradphi[j])
    sum = 0
    for m in range((len(gradphi[0]))):
        temp1 = np.dot(invtranspB, gradphii[m])
        temp2 = np.dot(invtranspB, gradphij[m])
        temp3 = weights[m] * np.dot(temp1,temp2)
        sum += temp3
    result = area * sum
    return result
```

Αντίστοιχα υπολογίζουμε και τα στοιχεία του δεξιού μέλους

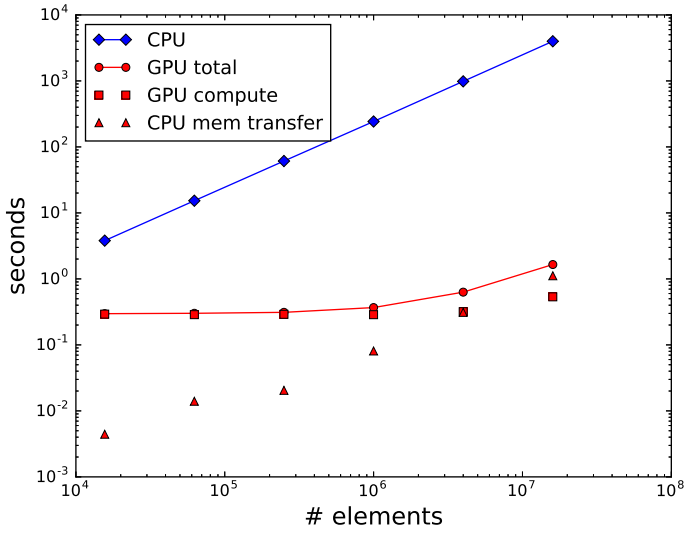
```
def localforces(phi, weights, points, B, C, detB):
    area = 0.5 * detB
    result = np.empty(len(phi), dtype=np.float64)
    for m in range(len(phi)):
        temp = 0
        for n in range(len(weights)):
            coords = np.dot(B, [points[n][0], points[n][1]]) + C
            temp += (weights[n] * problem.f(coords) * phi[m][n])
        result[m] = area*temp
    return result
```

4. Είδαμε ότι ο πίνακας stiffness είναι συμμετρικός, συνεπώς μπορούμε να υπολογίσουμε μόνο το άνω τριγωνικό μέρος κάθε τοπικού πίνακα και να αντιγράψουμε τις κατάλληλες τιμές στο κάτω τριγωνικό μέρος.

```
rowA = np.empty(NT*Noofbases**2, dtype=np.int32)
colA = np.empty(NT*Noofbases**2, dtype=np.int32)
valA = np.empty(NT*Noofbases**2, dtype=np.float64)
b = np.empty((N,1), dtype=np.float64)
for k in range(NT):
    B, C, invtranspB, detB = calc.affine(nodes[elements[k][0]-1],
                                         nodes[elements[k][1]-1],
                                         nodes[elements[k][2]-1])
```

```
    vforces = calc.localforces(phi, weights2, points2, B, C, detB)
    for i in range(Noofbases):
        b[elements[k][i]-1,0] += vforces[i]
        for j in range(i, Noofbases):
            rowA[k*Noofbases**2 + Noofbases*i + j] = elements[k][i]-1
            colA[k*Noofbases**2 + Noofbases*i + j] = elements[k][j]-1
            tmpA = calc.localstiffness(gradphi, weights1, invtranspB,
 detB, i, j)
            valA[k*Noofbases**2 + Noofbases*i + j] = tmpA
            if i != j:
                rowA[k*Noofbases**2 + Noofbases*j + i] = elements[k][
j]-1
                colA[k*Noofbases**2 + Noofbases*j + i] = elements[k][
i]-1
                valA[k*Noofbases**2 + Noofbases*j + i] = tmpA
```

elements	CPU	GPU comp	GPU mem
15616	3.8	0.293	0.004
62464	15.3	0.287	0.01
249856	61.1	0.291	0.02
999424	243	0.287	0.08
3997696	985	0.319	0.31
15990784	3990	0.535	1.12



Σχήμα 3.4: Χρόνοι (σε δευτερόλεπτα) για την κατασκευή του γραμμικού συστήματος του προβλήματος του κεφαλαίου 4.4 χρησιμοποιώντας γραμμικές συναρτήσεις βάσης. Για την CPU χρησιμοποιήσαμε 1 thread στον επεξεργαστή Intel Xeon E5-2687W. Η GPU είναι η Nvidia Tesla K20c. Για τη GPU δίνουμε δύο χρόνους. Ο πρώτος αφορά στον υπολογισμό και ο δεύτερος στην μεταφορά των δεδομένων από τη μνήμη της κάρτας γραφικών στη μνήμη RAM.

3.2.3 Υπολογισμός ενεργειακής νόρμας

Για τον υπολογισμό της ενεργειακής νόρμας υλοποιήσαμε στην GPU τον τύπο 3.1. Κάθε thread υπολογίζει το σφάλμα από ένα τρίγωνο. Όταν ο υπολογισμός ολοκληρωθεί για όλα τα τρίγωνα, τα αποτελέσματα επιστρέφονται στη μνήμη RAM, αθροίζονται από τον επεξεργαστή και επιστρέφεται η τετραγωνική ρίζα του αθροίσματος.

$$\begin{aligned}\|u-u_h\|_E^2 = \|\nabla(u-u_h)\|_{L^2(\Omega)}^2 &= \int_\Omega \nabla(u-u_h)\cdot\nabla(u-u_h)\\ &= \sum_{\tau\in\mathcal{T}}\int_\tau \big((u-u_h)_x^2+(u-u_h)_y^2\big)\\ &= \sum_{\tau\in\mathcal{T}}\int_\tau \Big(\underbrace{u_x^2}_{1} \underbrace{-2u_x(u_h)_x}_{2} \underbrace{+(u_h)_x^2}_{3} +u_y^2-2u_y(u_h)_y+(u_h)_y^2\Big).\end{aligned} \tag{3.1}$$

$$1 \to \int_\tau (u_x(x,y))^2 = \int_{\hat\tau}(u_x(T(\hat x,\hat y)))^2\,|\det B| = \frac{|\det B|}{2}\sum_{i=1}^{\#gauss} w_i\big(u_x(T(\hat x_i,\hat y_i))\big)^2.$$

$$2 \to \int_\tau u_x(x,y)\cdot(u_h)_x(x,y) = \int_{\hat\tau} u_x(T(\hat x,\hat y))\cdot\big(\underbrace{B^{-\mathsf{T}}\nabla\hat u_h(\hat x,\hat y)}_{\text{Μόνο το 1ο στοιχείο}}\big)|\det B| = \int_{\hat\tau} u_x(T(\hat x,\hat y))\cdot\big(B^{-\mathsf{T}}\sum_{j=1}^{\#\text{βάσεων}} u_j\nabla\varphi_j(\hat x,\hat y)\big)|\det B| = \frac{|\det B|}{2}\sum_{i=1}^{\#gauss} w_i\Big(u_x(T(\hat x_i,\hat y_i))\cdot\big(B^{-\mathsf{T}}\sum_{j=1}^{\#\text{βάσεων}} u_j\nabla\varphi_j(\hat x_i,\hat y_i)\big)\Big).$$

$$3 \to \int_\tau ((u_h)_x(x,y))^2 = \int_{\hat\tau}\big(\underbrace{B^{-\mathsf{T}}\nabla\hat u_h(\hat x,\hat y)}_{\text{Μόνο το 1ο στοιχείο}}\big)^2|\det B| = \int_{\hat\tau}\big(B^{-\mathsf{T}}\sum_{j=1}^{\#\text{βάσεων}} u_j\nabla\varphi_j(\hat x,\hat y)\big)^2|\det B| = \frac{|\det B|}{2}\sum_{i=1}^{\#gauss} w_i\big(B^{-\mathsf{T}}\sum_{j=1}^{\#\text{βάσεων}} u_j\nabla\varphi_j(\hat x_i,\hat y_i)\big)^2.$$

Αντίστοιχα και για τις παραγώγους ως προς y.

```
def energynorm(nodes, elements, mode, u):
    points1, weights1, points2, weights2 = gauss.quadrature(mode)
    Noofbases = len(elements[0])
    phi, gradphi = basis.basevalues(points1, points2, Noofbases, mode)
    N = len(nodes) # No of nodes
    NT = len(elements) # No of triangles
    energystiff = 0
    energymass = 0
    for k in range(NT):
        B, C, invtranspB, detB = calc.affine(nodes[elements[k][0]-1],
                                             nodes[elements[k][1]-1],
                                             nodes[elements[k][2]-1])
        area = 0.5*detB
        sum2 = 0
        for j in range(len(weights1)):
            sum1 = np.array([0.0, 0.0])
            for i in range(Noofbases):
                sum1 += u[elements[k][i]-1, 0] * np.array(gradphi[i][j])
            sum1 = np.dot(invtranspB, sum1)
            point = np.dot(B, [points1[j][0], points1[j][1]]) + C
            exactgrad = np.array(problem.exactderivatives(point))
            tmp1 = exactgrad - sum1
            tmp2 = np.dot(tmp1, tmp1)
            sum2 += weights1[j] * tmp2
        energystiff += area * sum2
    return math.sqrt(energystiff)
```

3.3 Διαχείριση μνήμης

Σε αυτό το σημείο πρέπει να αποφασίσουμε πως θα αποθηκεύουμε τα στοιχεία του πίνακα stiffness. Ας υποθέσουμε ότι έχουμε ένα πρόβλημα με 1.000.000 κόμβους και συνεπώς έναν πίνακα με 10^{12} στοιχεία. Χρησιμοποιώντας διπλή ακρίβεια (δηλαδή 8 byte για κάθε στοιχείο), η συνολική μνήμη που θα χρειαστούμε είναι περίπου 7 TiB [1]. Η κρίσιμη παρατήρηση είναι ότι επειδή οι συναρτήσεις βάσης έχουν μικρό στήριγμα ο πίνακας είναι αραιός και συνεπώς μπορούμε να αποθηκεύσουμε μόνο τα μη μηδενικά στοιχεία σε χώρο cn (αντί για n^2), όπου n το πλήθος των κόμβων και $c \ll n$ μια σταθερά που εξαρτάται από την επιλογή των συναρτήσεων βάσης (βάσεις υψηλότερης τάξης έχουν περισσότερες αλληλεπιδράσεις με γειτονικές βάσεις).

Θα μπορούσαμε να αποθηκεύσουμε τον πίνακα stiffness με το λεγόμενο *coordinate format* αλλά η παράλληλη κατασκευή του μας αποτρέπει. Με το coordinate format αποθηκεύουμε σε τρεις λίστες μεγέθους n η κάθε μία τους δείκτες γραμμής και στήλης και την τιμή μόνο των μη μηδενικών στοιχείων του πίνακα. Πλέον όμως υπάρχει το πρόβλημα ότι έχει χαθεί η πρόσβαση σε συγκεκριμένο στοιχείο του πίνακα σε χρόνο $\mathcal{O}(1)$. Για να προσθέσουμε σε κάποιο στοιχείο τη συνεισφορά από γειτονικό τρίγωνο θα έπρεπε να διατρέξουμε τις δύο λίστες row index και column index με δυαδική αναζήτηση σε χρόνο $\mathcal{O}(\log n)$, αν με επιπλέον υπολογιστικό κόστος τις κρατούσαμε ταξινομημένες ή διαφορετικά με σειριακή αναζήτηση σε χρόνο $\mathcal{O}(n)$. Υπάρχει όμως και ένα επιπλέον πρόβλημα λόγω της παράλληλης κατασκευής του πίνακα. Συγκεκριμένα δύο threads που υπολογίζουν τους τοπικούς πίνακες stiffness δύο γειτονικών τριγώνων υπήρχε πιθανότητα να

[1] 1 kB (kilobyte) = 10^3 bytes ενώ 1 KiB (kibibyte) = 2^{10} bytes.
Αντίστοιχα 1 TB (terabyte) = $(10^3)^4$ bytes ενώ 1 TiB (tebibyte) = $(2^{10})^4$ bytes.

προσπαθήσουν να γράψουν στην ίδια θέση μνήμης ταυτόχρονα κάτι που θα δημιουργούσε συγκρούσεις και αλλοίωση δεδομένων.

row id	3	1	4	1	5	$\cdots$
col id	5	1	2	3	2	$\cdots$
value	α_{35}	α_{11}	α_{42}	α_{13}	α_{52}	$\cdots$

Πίνακας 3.5: Οι τρεις λίστες του coo format ($\alpha_{ij} \neq 0$).

Θα επιλέξουμε να αποθηκεύσουμε τον πίνακα με μια παραλλαγή του coordinate format. Αντί να δημιουργήσουμε τρεις λίστες μεγέθους n, όπου n το πλήθος των κόμβων, θα δημιουργήσουμε τρεις λίστες μεγέθους $t \cdot b^2$, όπου t το πλήθος των τριγώνων και b το πλήθος των συναρτήσεων βάσης της μεθόδου. Τότε κάθε thread, σύμφωνα με το thread id του θα γράψει σε b^2 συνεχόμενες θέσεις μνήμης τις τιμές από τον τοπικό πίνακα stiffness. Προφανώς επειδή κάθε ακμή ij της τριγωνοποίησης, όπου i και j οι δύο κόμβοι της ακμής, ανήκει σε δύο τρίγωνα, θα έχουμε δύο φορές την αντιστοίχηση "γραμμή i – στήλη j" με την τιμή α_{ij}^{ℓ} να είναι ίση με τη συνεισφορά μόνο από το συγκεκριμένο τρίγωνο. Αντίστοιχα για κάθε κόμβο i θα έχουμε k εγγραφές α_{ii}^{ℓ}, όπου k το πλήθος των τριγώνων που έχουν την i ως κορυφή και ℓ ο δείκτης του τριγώνου. Όταν ολοκληρωθεί ο υπολογισμός όλων των τοπικών πινάκων θα επιστρέφονται οι τρεις λίστες από την μνήμη της GPU στη μνήμη RAM και θα δημιουργείται ο πίνακας μεγέθους n σε coordinate format αθροίζοντας μεταξύ τους τις τιμές από τα κοινά στοιχεία.

Είδαμε ότι προσωρινά οι πίνακες αποθηκεύονται στην ιδιαίτερα γρήγορη αλλά περιορισμένη μνήμη της κάρτας γραφικών. Μια σύγχρονη κάρτα Tesla έχει από 5 έως 24 GB μη αναβαθμίσιμης μνήμης GDDR3 και συνεπώς είναι περιορισμένο και το μέγεθος του προβλήματος που μπορούμε να κατασκευάσουμε σε έναν κύκλο. Έστω για παράδειγμα ότι έχουμε ένα πρόβλημα στο επίπεδο με 500.000 κόμβους και 1.000.000 τρίγωνα και έστω ότι θα χρησιμοποιήσουμε γραμμικές συναρτήσεις βάσης. Η συνολική μνήμη που θα χρειαστούμε, χρησιμοποιώντας μονή ακρίβεια (32bit) για τους δείκτες και διπλή ακρίβεια (64bit) για τις τιμές αναλύεται ως εξής

$$
\begin{aligned}
5 \times 10^5 \times 2 \times 8 &= 8 \times 10^6 \text{ bytes (nodes)} \\
10^6 \times 3 \times 4 &= 1,2 \times 10^7 \text{ bytes (elements)} \\
10^6 \times 3 \times 4 &= 1,2 \times 10^7 \text{ bytes (b id)} \\
10^6 \times 3 \times 8 &= 2,4 \times 10^7 \text{ bytes (b val)} \\
10^6 \times 3^2 \times 4 &= 3,6 \times 10^7 \text{ bytes (A row)} \\
10^6 \times 3^2 \times 4 &= 3,6 \times 10^7 \text{ bytes (A col)} \\
10^6 \times 3^2 \times 8 &= 7,2 \times 10^7 \text{ bytes (A val)}
\end{aligned}
$$

Το σύνολο είναι περίπου 0,186 GiB. Με αντίστοιχους υπολογισμούς βρίσκουμε ότι για το ίδιο πρόβλημα και για τετραγωνικές συναρτήσεις βάσης χρειαζόμαστε 0,62 GiB, για κυβικές 1,62 GiB και για $4^{ου}$ βαθμού 3,54 GiB.

Για μεγαλύτερα προβλήματα, για τα οποία δεν επαρκεί η μνήμη μπορούμε να κατασκευάσουμε τον πίνακα επαναληπτικά. Συγκεκριμένα μπορούμε να κα-

row id	column id	local value
2	2	α_{22}^{1}
2	3	α_{23}^{1}
2	5	α_{25}^{1}
3	3	α_{33}^{1}
3	5	α_{35}^{1}
3	2	α_{32}^{1}
5	5	α_{55}^{1}
5	2	α_{52}^{1}
5	3	α_{53}^{1}
1	1	α_{11}^{2}
1	2	α_{12}^{2}
1	5	α_{15}^{2}
2	2	α_{22}^{2}
2	5	α_{25}^{2}
2	1	α_{21}^{2}
5	5	α_{55}^{2}
5	1	α_{51}^{2}
5	2	α_{52}^{2}
1	1	α_{11}^{3}
⋮	⋮	⋮

Πίνακας 3.6: Οι τρεις λίστες του τροποποιημένου coo format για την τριγωνοποίηση του σχήματος 3.3 υποθέτοντας ότι έχουμε χρησιμοποιήσει γραμμικές συναρτήσεις βάσης. Διακρίνουμε τις $9=3^2$ συνεχόμενες εγγραφές που έγιναν από κάθε thread για το αντίστοιχο τρίγωνο. Θυμίζουμε ότι το πλήθος των συναρτήσεων βάσης για τις γραμμικές FEM είναι 3.

τασκευάσουμε μέρος του πίνακα, να επιστρέψουμε τα δεδομένα στη μνήμη RAM και να συνεχίσουμε με το επόμενο μέρος του χωρίου. Επίσης μπορούμε να χρησιμοποιήσουμε και επιπλέον κάρτες γραφικών στον ίδιο υπολογιστή, που εκτός από περισσότερη μνήμη θα δώσουν και περισσότερη επεξεργαστική ισχύ. Σε κάθε περίπτωση όμως θα πρέπει να ελαχιστοποιήσουμε τις μεταφορές από τη μνήμη RAM στη μνήμη της GPU γιατί όπως είδαμε στο σχήμα 3.4 οι μεταφορές μέσω του διαύλου PCI Express απαιτούν περισσότερο χρόνο από τον υπολογισμό. Αυτός είναι και ο λόγος που ίσως είναι προτιμότερο να κατασκευάσουμε με την κάρτα γραφικών μόνο το άνω τριγωνικό μέρος κάθε συμμετρικού τοπικού πίνακα και το κάτω τριγωνικό μέρος να κατασκευάζεται από τον επεξεργαστή στην μνήμη RAM. Με αυτόν τον τρόπο μπορούμε στον ίδιο κύκλο να κατασκευάσουμε σχεδόν διπλάσιο πλήθος τριγώνων.

Κεφάλαιο 4

Αριθμητικά αποτελέσματα

Θα κατασκευάσουμε σε χωρία διαφορετικής γεωμετρίας μερικές διαφορικές εξισώσεις, των οποίων θα γνωρίζουμε την ακριβή λύση, ώστε να μελετήσουμε την τάξη σύγκλισης αλλά και τους χρόνους κατασκευής των πινάκων του γραμμικού συστήματος και επίλυσης αυτού.

Οι υπολογισμοί θα γίνουν σε ένα σύστημα με έναν διπύρηνο με Hyper Threading i5-5200U με την ταχύτητα του κάθε πυρήνα στα 2.2GHz και 3MB μνήμη cache, 8GB ($2 \times$ 4GB) DDR3 μνήμη RAM στα 800MHz και την κάρτα γραφικών Nvidia GeForce 920m. Η συγκεκριμένη κάρτα γραφικών διαθέτει 2 streaming multi– processors με 192 shaders στα 954MHz ο κάθε ένας και 4GB DDR3 μνήμη, με τον δίαυλο να είναι 64-bit και την ταχύτητα στα 900MHz. Τέλος η κάρτα γραφικών υποστηρίζει το CUDA Compute Capability 3.5.

4.1 Ελλειπτικό χωρίο

Το πρώτο αριθμητικό παράδειγμα που θα κατασκευάσουμε αφορά το ελλειπτικό χωρίο του επιπέδου που περιγράφεται από την εξίσωση $1 - x^2 - 4y^2 \geq 0$. Ορίζοντας ως $u(x, y) = 1 - x^2 - 4y^2$ και περνώντας τη συνάρτηση u από τον Λαπλασιανό τελεστή βρίσκουμε $-\Delta u = f = 10$. Τότε το πρόβλημα $-\Delta u = 10$ στο $\Omega = \{(x, y) : 1-x^2-4y^2 \geq 0\} \subseteq \mathbb{R}^2$ με $u = 0$ στο $\partial\Omega = \{(x, y) : 1-x^2-4y^2 = 0\}$ έχει ως μοναδική λύση την $u(x, y) = 1 - x^2 - 4y^2$.

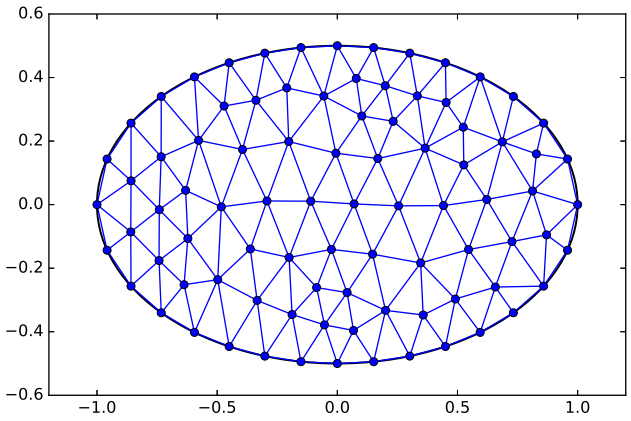
Θα πάρουμε μια ακολουθία διαμερίσεων του χωρίου σε τρίγωνα και για κάθε διαμέριση θα χρησιμοποιήσουμε τη μέθοδο πεπερασμένων στοιχείων με συναρτήσεις βάσης $1^{ου}$ έως και $4^{ου}$ βαθμού πολυώνυμα. Τα χαρακτηριστικά των χωρίων φαίνονται στον πίνακα 4.1.

Όπως είδαμε η ακριβής λύση είναι μια δευτεροβάθμια συνάρτηση, συνεπώς θα περιμέναμε η προσεγγιστική λύση με βάσεις $2^{ου}$ βαθμού και άνω να ταυτίζεται με την ακριβή λύση, εκτός από κάποια σφάλματα στρογγύλευσης και αποκοπής. Αντ' αυτού, όπως φαίνεται και στον πίνακα 4.2, η προσεγγιστική λύση διαφέρει από την ακριβή και μάλιστα η τάξη σύγκλισης δεν είναι αυτή που αναμέναμε. Ο λόγος είναι ότι έχουμε εισάγει σφάλματα κατά την κατασκευή της τριγωνοποίησης. Το σύνορο του χωρίου είναι μια έλλειψη όμως κατά δημιουργία του πλέγματος μετατρέπεται σε ένα πολύγωνο και συνεπώς η καμπύλη προσεγγίζε-

h	Nodes	Elements
0.22963	87	140
0.11482	313	560
0.05742	1185	2240
0.02883	4609	8960
0.01445	18177	35840
0.00723	72193	143360

Πίνακας 4.1: Οι έξι διαμερίσεις του ελειπτικού χωρίου που θα χρησιμοποιήσουμε.

ται από ευθύγραμμα τμήματα. Επομένως η λύση δεν αφορά το αρχικό χωρίο αλλά το προσεγγιστικό και αυτό δημιουργεί την απόκλιση που παρατηρούμε.



Σχήμα 4.1: Το ελλειπτικό χωρίο και μια αραιή τριγωνοποίησή του.

	$\|(u-u_h)\|_E$			
h	Linear	Quadratic	Cubic	Quartic
0.22963	0.205625	0.076964	0.057481	0.053198
0.11482	0.108838	0.027811	0.020592	0.018943
0.05742	0.055582	0.009927	0.007322	0.006715
0.02883	0.027988	0.003526	0.002595	0.002377
0.01445	0.014025	0.001249	0.000919	0.000841
0.00723	0.007017	0.000443	0.000326	0.000299

Πίνακας 4.2: Σφάλματα προσέγγισης μετρημένα στην ενεργειακή νόρμα.

Όπως φαίνεται στον πίνακα 4.3 η τάξη σύγκλισης p για τη μέθοδο με τις γραμμικές συναρτήσεις βάσεις τείνει στη μονάδα, όπως ακριβώς περιμένουμε από τη θεωρία. Όμως για τις μεθόδους ανώτερης τάξης, η τάξη σύγκλισης δεν τείνει στο

2, 3 και 4 αλλά και για τις τρεις μεθόδους έχουμε $p \to 1.5$ καθώς $h \to 0$. Αυτό δείχνει πόσο σημαντικό είναι να έχουμε ακριβή αναπαράσταση του συνόρου του χωρίου καθώς σε αντίθετη περίπτωση επηρεάζεται κατά πολύ η σύγκλιση και χάνουμε τα πλεονεκτήματα της χρήσης μεθόδων ανώτερης τάξης.

Συγκεκριμένα έστω το πρόβλημα

$$\begin{aligned} -\Delta u &= f \text{ στο } \Omega \\ u &= 0 \text{ στο } \partial\Omega, \end{aligned} \tag{4.1}$$

όπου Ω το ελλειπτικό χωρίο του παραδείγματος. Μετά την τριγωνοποίηση, θα έχουμε μια προσέγγιση του αρχικού χωρίου, το πολυγωνικό χωρίο $\tilde{\Omega}$, του οποίου το κάθε τρίγωνο θα έχει το πολύ μία ακμή που θα προσεγγίζει το σύνορο και οι κόμβοι αυτής της ακμής θα βρίσκονται πάνω στο σύνορο του αρχικού χωρίου Ω. Πλέον έχουμε το πρόβλημα

$$\begin{aligned} -\Delta \tilde{u} &= f \text{ στο } \tilde{\Omega} \\ \tilde{u} &= 0 \text{ στο } \partial\tilde{\Omega}, \end{aligned} \tag{4.2}$$

και η λύση $\tilde{u}_h$ που υπολογίζουμε βασίζεται σε αυτό το πρόβλημα. Συνεπώς ισχύει ακόμα η εξής σχέση για το σφάλμα

$$|\tilde{u} - \tilde{u}_h|_{H^1(\tilde{\Omega})} \leq Ch^{s-1}|\tilde{u}|_{H^s(\tilde{\Omega})}, \quad 1 < s = \min\{k, p+1\}. \tag{4.3}$$

Εμείς όμως υπολογίζουμε το σφάλμα της προσεγγιστικής λύσης $\tilde{u}_h$ από την ακριβή λύση u στο $\tilde{\Omega}$ και για αυτό το σφάλμα έχουμε

$$|u - \tilde{u}_h|_{H^1(\tilde{\Omega})} \leq |u - \tilde{u}|_{H^1(\tilde{\Omega})} + |\tilde{u} - \tilde{u}_h|_{H^1(\tilde{\Omega})}. \tag{4.4}$$

Είδαμε ότι ο δεύτερος όρος έχει την βέλτιστη τάξη σύγκλισης. Θα εξετάσουμε το σφάλμα από τον πρώτο όρο. Λόγω της ευστάθειας του ελλειπτικού προβλήματος στο χωρίο $\tilde{\Omega}$ έχουμε την ακόλουθη σχέση

$$|u - \tilde{u}|^2_{H^1(\tilde{\Omega})} \leq C_s|u - \tilde{u}|^2_{H^{1/2}(\partial\tilde{\Omega})} = C_s \sum_{F \subset \partial\tilde{\Omega}} |u - \tilde{u}|^2_{H^{1/2}(F)}. \tag{4.5}$$

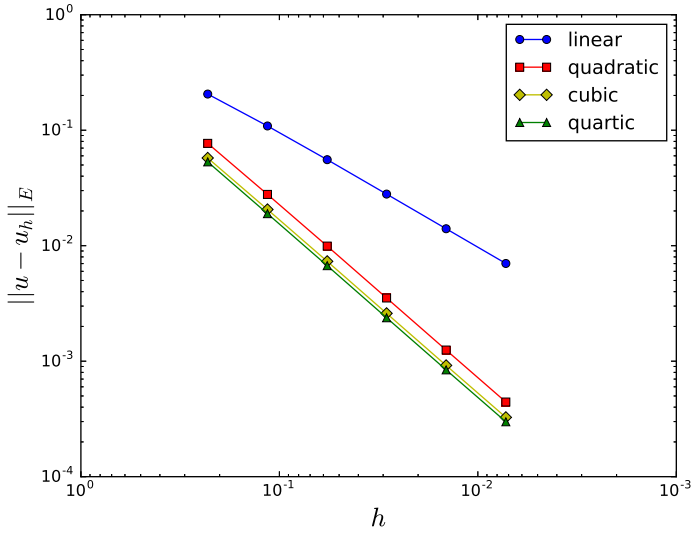
Σε αυτό το σημείο πρέπει να παρατηρήσουμε ότι σε κάθε ακμή του συνόρου F, το $u|_F$ είναι μια τετραγωνική πολυωνυμική συνάρτηση και $u|_{\partial F} = 0$, δηλαδή στους δύο κόμβους της ακμής που βρίσκονται πάνω στο σύνορο του χωρίου Ω η συνάρτηση μηδενίζεται. Όμως $\tilde{u}|_F = 0$ και συνεπώς μπορούμε να σκεφτούμε την $\tilde{u}|_F$ σαν τη γραμμική παρεμβολή της u στο $F \subset \partial\tilde{\Omega}$, $\tilde{u}|_F = \mathcal{I}_1 u|_F$. Από τις ιδιότητες της γραμμικής παρεμβολής έχουμε

$$|u-\tilde{u}|^2_{H^1(\tilde{\Omega})} \leq C_s \sum_{F \subset \partial\tilde{\Omega}} |u-\mathcal{I}_1 u|^2_{H^{1/2}(F)} \leq C \sum_{F \subset \partial\tilde{\Omega}} h_F^3 |u|^2_{H^2(F)} \leq Ch^3|u|^2_{H^2(\partial\tilde{\Omega})}. \tag{4.6}$$

Συνεπώς για $p = 1$ το σφάλμα από τον δεύτερο όρο κυριαρχεί και άρα η μέθοδος θα συγκλίνει με τη βέλτιστη τάξη. Όμως για $p \geq 2$ το ανωτέρω σφάλμα, που είναι της τάξης $\mathcal{O}(h^{3/2})$ και είναι ανεξάρτητο του p, θα κυριαρχεί.

Linear	Quadratic	Cubic	Quartic
0.91784	1.46853	1.48099	1.48971
0.96960	1.48651	1.49217	1.49645
0.99573	1.50247	1.50513	1.50722
1.00046	1.50218	1.50331	1.50428
0.99910	1.49551	1.49381	1.49334

Πίνακας 4.3: Τάξη σύγκλισης p.



Σχήμα 4.2: Σφάλματα προσέγγισης συναρτήσει του h για τις τέσσερις μεθόδους.

Η χρήση μεθόδων ανώτερης τάξης εισάγει ένα μεγαλύτερο υπολογιστικό κόστος για να επιτευχθεί η ταχύτερη σύγκλιση. Εκτός από περισσότερες συναρτήσεις βάσης σε κάθε τρίγωνο του πλέγματος, πρέπει να χρησιμοποιήσουμε και περισσότερα σημεία για τον υπολογισμό των ολοκληρωμάτων, αν θέλουμε αυτοί οι υπολογισμοί να γίνουν με τη μεγαλύτερη δυνατή ακρίβεια. Στον πίνακα 4.4 φαίνεται ο χρόνος που χρειάστηκε για τον υπολογισμό του stiffness matrix A και του διανύσματος του δεξιού μέλους b χρησιμοποιώντας τη CPU και τη GPU. Οι χρόνοι που αφορούν τη GPU περιλαμβάνουν και τη μεταφορά των δεδομένων από τη RAM στην μνήμη της κάρτας γραφικών και πίσω.

Όπως είναι φανερό, η κατασκευή των πινάκων παράλληλα, και συγκεκριμένα σε κάρτες γραφικών, επιτυγχάνει πολύ μεγάλη μείωση στον χρόνο εκτέλεσης. Συγκεκριμένα, τα αποτελέσματα έδειξαν επιτάχυνση έως και 800 φορές σε σχέση με τον ίδιο υπολογισμό σε ένα thread στη CPU. Ένα άλλο ενδιαφέρον χαρακτηριστικό που παρατηρούμε είναι ότι δεν έχουμε φτάσει ακόμα τα όρια της GPU. Μειώνοντας τη διάμετρο h του πλέγματος στο μισό δημιουργούμε τετραπλάσιο φόρτο υπολογισμού. Οι κόμβοι και τα τρίγωνα τετραπλασιάζονται και ο υπολογισμός στη CPU χρειάζεται τέσσερις φορές περισσότερο χρόνο, όπως ακριβώς

περιμέναμε. Από την άλλη δεν παρατηρούμε το ίδιο για τον υπολογισμό στη GPU. Συγκεκριμένα, ενώ για το πλέγμα με $h = 0.01445$ και χρησιμοποιώντας τη μέθοδο υψηλότερης τάξης ο υπολογισμός απαιτεί 0.36 δευτερόλεπτα στο πλέγμα με $h = 0.00723$ ο υπολογισμός χρειάζεται 1.06 δευτερόλεπτα, δηλαδή περίπου τρεις φορές περισσότερο χρόνο. Αυτό δείχνει ότι οι μεταφορές και οι εγγραφές στη μνήμη κυριαρχούν σε σχέση με τον καθαυτό υπολογισμό.

Υπάρχουν δύο τρόποι για να επιτύχουμε τη βέλτιστη χρήση της κάρτας γραφικών. Μπορούμε είτε να χρησιμοποιήσουμε ακόμα πιο πυκνό πλέγμα είτε μέθοδο μεγαλύτερης τάξης. Με τον δεύτερο τρόπο επιτυγχάνουμε κάθε thread στη GPU να "απασχολείται" περισσότερο χρόνο υπολογίζοντας, παρά γράφοντας στη μνήμη. Συγκεκριμένα χρησιμοποιώντας μια μέθοδο με n το πλήθος βαθμούς ελευθερίας σε κάθε τρίγωνο, κάθε thread χρειάζεται $\frac{n(n+1)}{2}$ επαναλήψεις (όσο δηλαδή και το πλήθος των στοιχείων στο τριγωνικό μέρος του συμμετρικού local stiffness matrix) και για κάθε τέτοια επανάληψη, αφού θα χρησιμοποιούμε πολυώνυμα υψηλότερου βαθμού, θα χρειάζεται να υπολογίσουμε τιμές σε μεγαλύτερο πλήθος σημείων για την Gauss ολοκλήρωση.

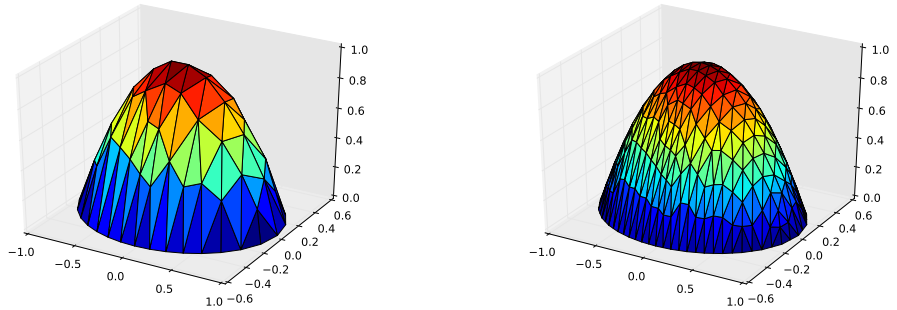
	Linear		Quadratic		Cubic		Quartic	
h	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
0.22963	0.04	0.16	0.09	0.12	0.25	0.12	0.79	0.12
0.11482	0.13	0.16	0.33	0.12	0.99	0.12	3.18	0.12
0.05742	0.52	0.16	1.36	0.13	3.98	0.13	13	0.14
0.02883	2.13	0.16	5.33	0.13	17.1	0.15	52	0.19
0.01445	8.61	0.16	21.5	0.16	64.2	0.23	202	0.36
0.00723	34.1	0.19	86.7	0.26	264	0.54	815	1.06

Πίνακας 4.4: Χρόνος κατασκευής (σε δευτερόλεπτα) των πινάκων του γραμμικού συστήματος. Στην περίπτωση της CPU χρησιμοποιήσαμε σειριακό αλγόριθμο με 1 thread.

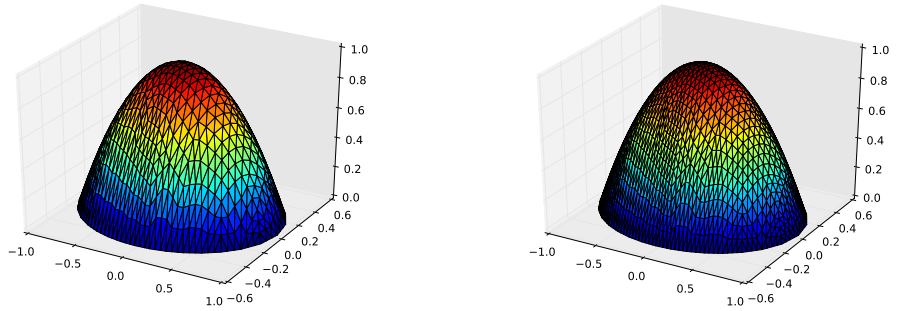
Μετά την δημιουργία του γραμμικού συστήματος θα πρέπει προφανώς να το λύσουμε για να βρούμε την προσεγγιστική λύση. Όπως βλέπουμε στον πίνακα 4.5 η χρήση μεδόθων υψηλής τάξης επηρεάζει τον χρόνο επίλυσης. Για παράδειγμα χρησιμοποιώντας στο πλέγμα με $h = 0.02883$ βάσεις $4^{\text{ου}}$ βαθμού καταλήγουμε με ένα σύστημα 71169×71169, ίδιου μεγέθους με το να χρησιμοποιήσουμε τετραγωνικές βάσεις στο πλέγμα με $h = 0.01445$ ή γραμμικές στο πλέγμα με $h = 0.00723$. Η επίλυση απαιτεί περίπου 3, 2 και 1.5 δευτερόλεπτα αντίστοιχα και μάλιστα η διαφορά γίνεται ακόμη μεγαλύτερη όσο αυξάνει η διάσταση του γραμμικού συστήματος. Ο λόγος για αυτή την αύξηση στον χρόνο επίλυσης είναι ότι με μεθόδους ανώτερης τάξης ο πίνακας A του γραμμικού συστήματος γίνεται πιο πυκνός, δηλαδή έχει λιγότερα μη μηδενικά στοιχεία γιατί οι βάσεις έχουν μεγαλύτερο στήριγμα. Δηλαδή υπάρχουν περισσότερες αλληλεπιδράσεις μεταξύ των ελεύθερων κόμβων.

h	Linear	Quadratic	Cubic	Quartic
0.22963	0.0004 (55)	0.0009 (249)	0.0019 (583)	0.0036 (1057)
0.11482	0.001 (249)	0.004 (1057)	0.0119 (2425)	0.033 (4353)
0.05742	0.004 (1057)	0.026 (4353)	0.1217 (9889)	0.0.389 (17665)
0.02883	0.023 (4353)	0.249 (17665)	1.18 (39937)	2.97 (71169)
0.01445	0.139 (17665)	1.97 (71169)	10.9 (160513)	29 (285697)
0.00723	1.41 (71169)	14.7 (285697)	111 (643585)	399 (1144833)

Πίνακας 4.5: Χρόνοι (σε δευτερόλεπτα) για την επίλυση του γραμμικού συστήματος. Σε παρένθεση το πλήθος των βαθμών ελευθερίας. Η επίλυση έγινε με τον sparse solver της βιβλιοθήκης SciPy και χρησιμοποιήθηκαν 4 threads.



Linear Quadratic

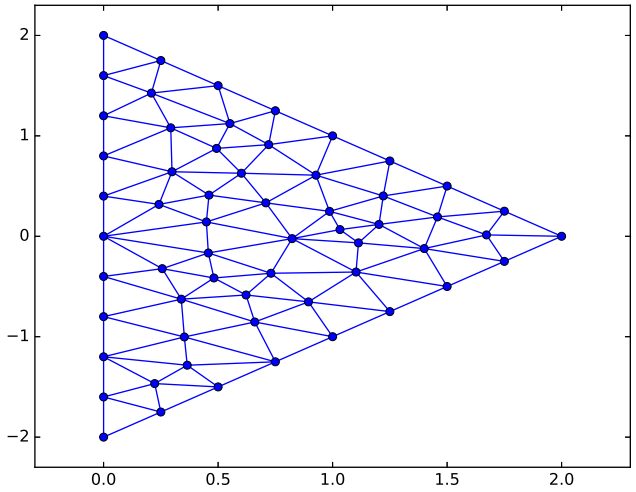


Cubic Quartic

Σχήμα 4.3: Η λύση στο πλέγμα με $h = 0.22963$.

4.2 Τριγωνικό χωρίο

Με τον ίδιο τρόπο θα κατασκευάσουμε μια μερική διαφορική εξίσωση σε ένα τριγωνικό χωρίο του επιπέδου που η ακριβής λύση της θα είναι ένα κυβικό πολυώνυμο στις 2 διαστάσεις. Έστω Ω το χωρίο που περικλείεται από τις ευθείες $x = 0$, $y = x - 2$ και $y = 2 - x$. Τότε το πρόβλημα $-\Delta u = 8 - 4x$ στο Ω με $u = 0$ $\forall (x, y) \in \partial\Omega$ έχει ως μοναδική λύση την $u(x, y) = -x(y-x+2)(y+x-2)$.



Σχήμα 4.4: Το τριγωνικό χωρίο και το πλέγμα με $h = 0.48659$.

h	Nodes	Elements
0.48659	59	90
0.24330	207	360
0.12165	773	1440
0.06082	2985	5760
0.03041	11729	23040
0.01521	46497	92160
0.00760	185153	368640

Πίνακας 4.6: Τα πλέγματα που θα χρησιμοποιήσουμε για το τριγωνικό χωρίο.

Όπως φαίνεται από τους πίνακες 4.7 και 4.8 πράγματι η τάξη σύγκλισης τείνει, για χωρία με μεγάλη διάμετρο, στο 1 και 2 για τις γραμμικές και τετραγωνικές βάσεις αντίστοιχα. Επίσης για τις βάσεις μεγαλύτερου βαθμού παρατηρούμε ότι τα σφάλματα είναι λίγες τάξεις μεγέθους μεγαλύτερα από το έψιλον της μηχανής, που σημαίνει ότι πράγματι η μέθοδος βρίσκει την ακριβή λύση. Όσο όμως

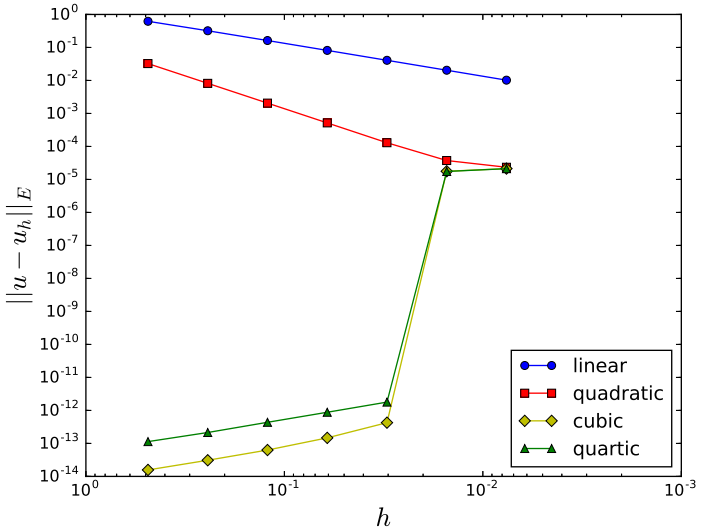
η διάμετρος του χωρίου μικραίνει παρατηρούμε την τάξη σύγκλισης να αλλάζει, προς το χειρότερο, και το ίδιο συμβαίνει και για τις μεθόδους που θα έπρεπε να βρίσκουν την ακριβή λύση. Μάλιστα παρατηρούμε πολύ γρήγορα όλες οι μέθοδοι να βρίσκουν λύση με σφάλμα ενεργειακής νόρμας περίπου 10^{-5}. Ο λόγος που συμβαίνει αυτό είναι η κακή ποιότητα του πλέγματος που χρησιμοποιήθηκε. Όπως φαίνεται στο σχήμα 4.4 υπάρχουν τρίγωνα με κακή ποιότητα, δηλαδή τρίγωνα με πολύ μεγάλη γωνία. Στη συνέχεια θα παρουσιάσουμε τα αποτελέσματα για το ίδιο χωρίο σε καλύτερης ποιότητας τριγωνοποίηση.

	$\|(u-u_h)\|_E$			
h	Linear	Quadratic	Cubic	Quartic
0.48659	0.620356	0.032405	1.57e-14	1.13e-13
0.24330	0.320458	0.008176	3.09e-14	2.14e-13
0.12165	0.162186	0.002058	6.30e-14	4.36e-13
0.06082	0.081422	0.000517	1.47e-13	8.79e-13
0.03041	0.040763	0.000129	4.25e-13	1.79e-12
0.01521	0.020389	3.73e-05	1.77e-05	1.74e-05
0.00760	0.010196	2.34e-05	2.13e-05	2.12e-05

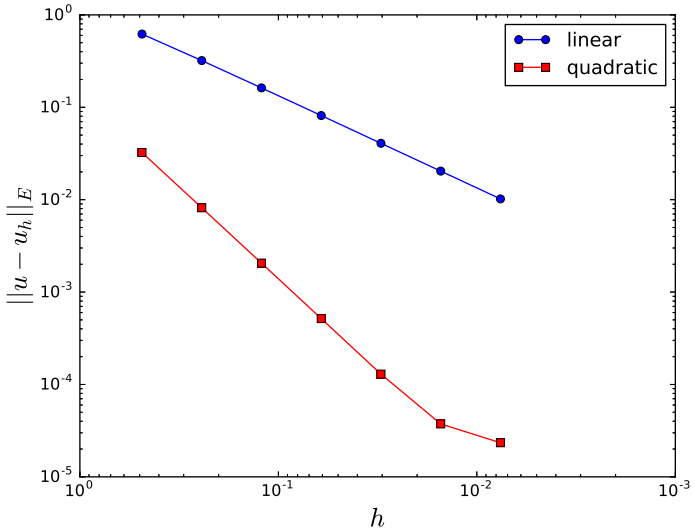
Πίνακας 4.7: Σφάλματα προσέγγισης μετρημένα στην ενεργειακή νόρμα.

Linear	Quadratic	Cubic	Quartic
0.95296	1.98675	-0.97480	-0.91665
0.98249	1.98991	-1.02831	-1.02723
0.99416	1.99421	-1.22564	-1.01086
0.99818	1.99694	-1.52924	-1.02250
0.99951	1.79173	-25.3100	-23.2215
0.99987	0.67597	-0.27149	-0.28001

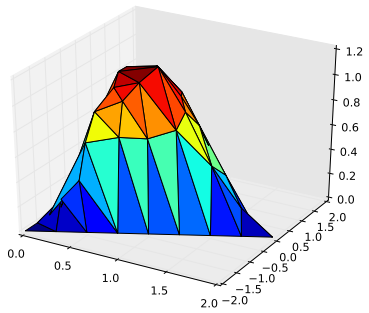
Πίνακας 4.8: Τάξη σύγκλισης p.



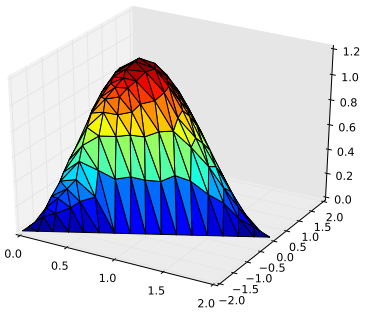
Σχήμα 4.5: Σφάλματα προσέγγισης συναρτήσει του h για τις τέσσερις μεθόδους.



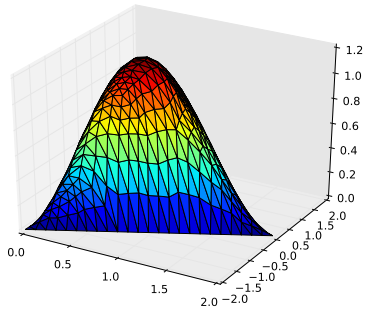
Σχήμα 4.6: Σφάλματα προσέγγισης συναρτήσει του h μόνο για γραμμικές και τετραγωνικές συναρτήσεις βάσης.



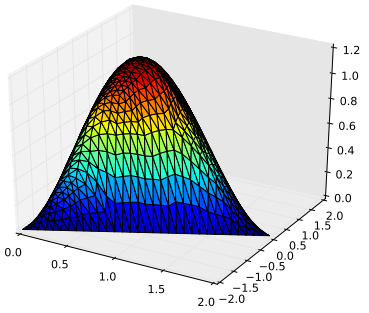
Linear



Quadratic



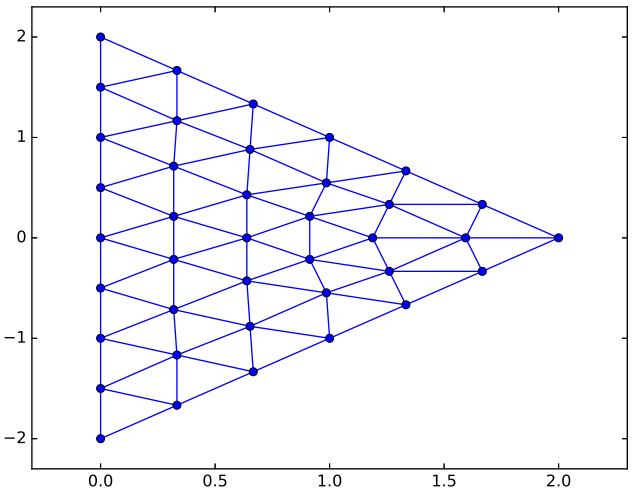
Cubic



Quartic

Σχήμα 4.7: Η λύση στο πλέγμα με $h = 0.48659$.

Τα χαρακτηριστικά για το δεύτερο πλέγμα που θα χρησιμοποιήσουμε για το τριγωνικό χωρίο φαίνονται παρακάτω. Σε αυτή τη τριγωνοποίηση δεν υπάρχει τρίγωνο με πολύ μεγάλη γωνία και συνεπώς περιμένουμε να έχουμε καλύτερα αποτελέσματα.



Σχήμα 4.8: Το τριγωνικό χωρίο και το πλέγμα με $h = 0.5$.

h	Nodes	Elements
0.50000	39	56
0.25000	133	224
0.12500	489	896
0.06250	1873	3584
0.03125	7329	14336
0.01563	28993	57344
0.00781	115329	229376

Πίνακας 4.9: Τα χαρακτηριστικά για το δεύτερο πλέγμα που θα χρησιμοποιήσουμε για το τριγωνικό χωρίο.

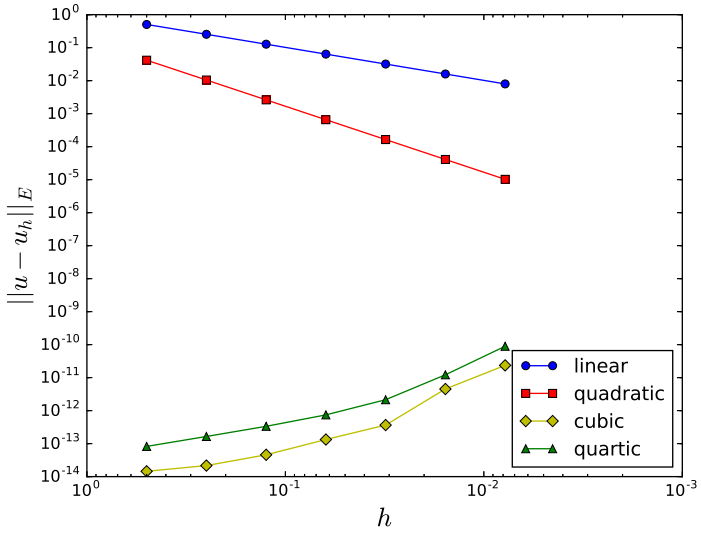
Πράγματι τα αποτελέσματα δείχνουν ότι πλέον οι γραμμικές και τετραγωνικές FEM συγκλίνουν με τη βέλτιστη τάξη ακόμη και για τα πυκνά πλέγματα ενώ οι μέθοδοι ανώτερης τάξης παρουσιάζουν την αναμενόμενη συμπεριφορά και βρίσκουν την ακριβή λύση.

	$\|(u-u_h)\|_E$			
h	Linear	Quadratic	Cubic	Quartic
0.50000	0.504617	0.041743	1.45e-14	8.24e-14
0.25000	0.254231	0.010461	2.16e-14	1.65e-13
0.12500	0.127385	0.002618	4.59e-14	3.35e-13
0.06250	0.063729	0.000655	1.33e-13	7.43e-13
0.03125	0.031869	0.000164	3.65e-13	2.14e-12
0.01563	0.015935	4.10e-05	4.52e-12	1.22e-11
0.00781	0.007968	1.02e-05	2.36e-11	8.92e-11

Πίνακας 4.10: Σφάλματα προσέγγισης μετρημένα στην ενεργειακή νόρμα.

Linear	Quadratic	Cubic	Quartic
0.98905	1.99656	-0.57503	-1.00002
0.99694	1.99829	-1.08612	-1.02194
0.99917	1.99915	-1.53759	-1.15154
0.99978	1.99957	-1.45528	-1.52663
0.99994	1.99979	-3.62941	-2.50818
1.00008	2.00008	-2.38199	-2.87272

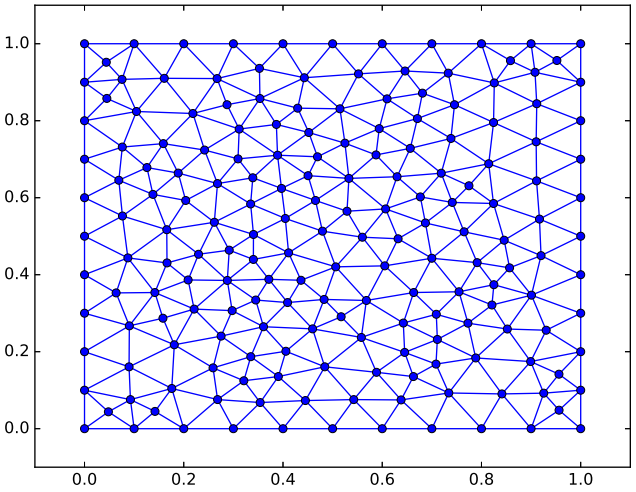
Πίνακας 4.11: Τάξη σύγκλισης p.



Σχήμα 4.9: Σφάλματα προσέγγισης συναρτήσει του h για τις τέσσερις μεθόδους.

4.3 Τετραγωνικό χωρίο

Έστω το χωρίο Ω που περικλείεται από τις τέσσερις ευθείες $x = 0$, $x = 1$, $y = 0$ και $y = 1$ και έστω το πρόβλημα να βρεθεί u έτσι ώστε $-\Delta u = 2(x+y-x^2-y^2)$ στο Ω με $u = 0$ στο $\partial\Omega$. Η ακριβής λύση αυτού του προβλήματος είναι $u(x, y) = x(1 - x)y(1 - y)$.



Σχήμα 4.10: Το τετραγωνικό χωρίο και το πλέγμα με $h = 0.11385$.

h	Nodes	Elements
0.11385	185	328
0.05693	697	1312
0.02846	2705	5248
0.01423	10657	20992
0.00712	42305	83968
0.00356	168577	335872

Πίνακας 4.12: Τα πλέγματα που θα χρησιμοποιήσουμε για το χωρίο του σχήματος 4.10.

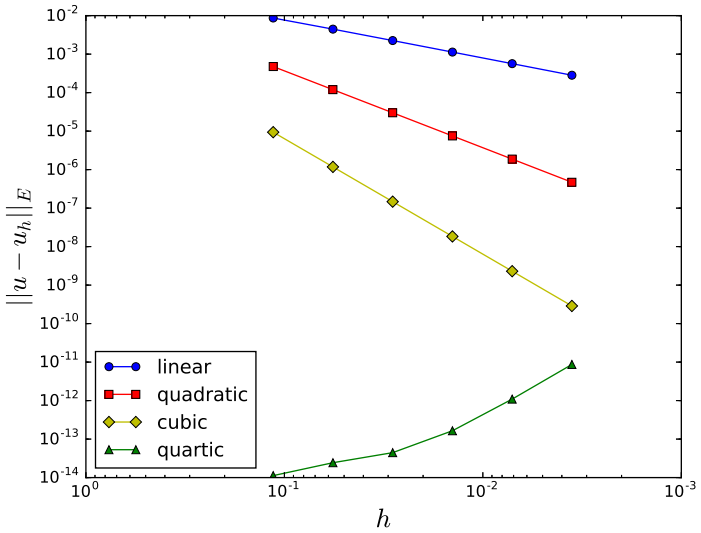
Η λύση είναι πολυώνυμο $4^{ου}$ βαθμού στις δύο διαστάσεις και συνεπώς περιμένουμε όλες οι μέθοδοι, εκτός από τις FEM $4^{ου}$ βαθμού να αφήνουν σφάλμα. Πράγματι, όπως βλέπουμε στον πίνακα 4.13 τα αριθμητικά αποτελέσματα είναι τα αναμενόμενα από τη θεωρία.

	$\|(u-u_h)\|_E$			
h	Linear	Quadratic	Cubic	Quartic
0.11385	0.008677	0.000476	9.39e-06	1.13e-14
0.05693	0.004458	0.000119	1.18e-06	2.44e-14
0.02846	0.002253	3.00e-05	1.47e-07	4.45e-14
0.01423	0.001130	7.50e-06	1.84e-08	1.65e-13
0.00712	0.000566	1.88e-06	2.30e-09	1.09e-12
0.00356	0.000283	4.70e-07	2.87e-10	8.68e-12

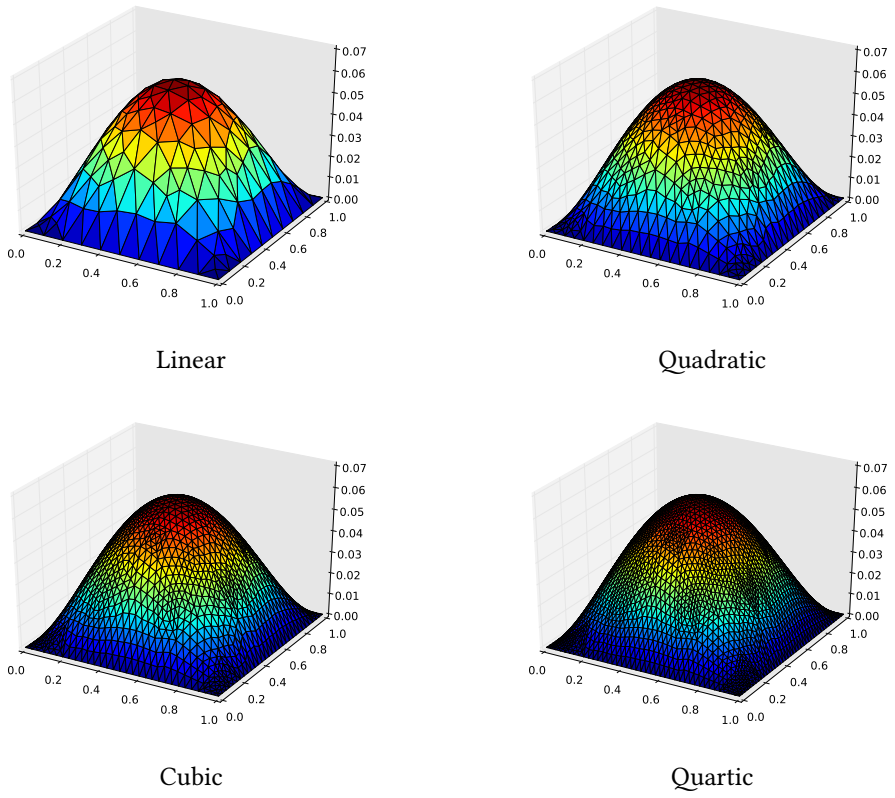
Πίνακας 4.13: Σφάλματα προσέγγισης μετρημένα στην ενεργειακή νόρμα.

Linear	Quadratic	Cubic	Quartic
0.96094	1.99469	2.99626	-1.11389
0.98476	1.99526	2.99944	-0.86815
0.99482	1.99732	3.00016	-1.88920
0.99844	1.99871	3.00038	-2.73058
0.99971	1.99966	3.00069	-2.99095

Πίνακας 4.14: Τάξη σύγκλισης p.



Σχήμα 4.11: Σφάλματα προσέγγισης συναρτήσει του h για τις τέσσερις μεθόδους.

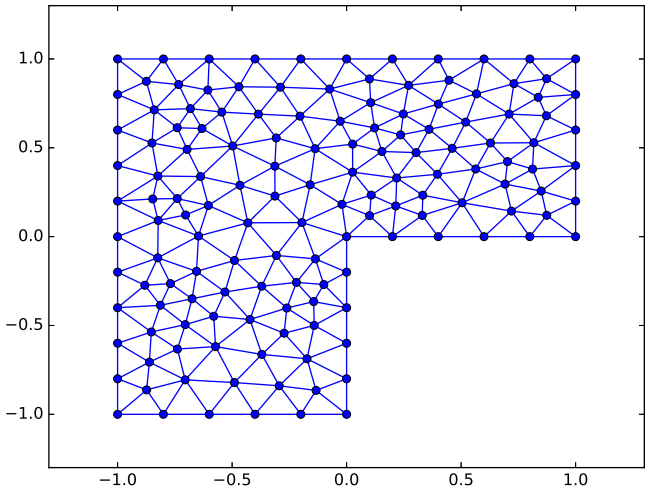


Σχήμα 4.12: Η λύση στο πλέγμα με $h = 0.11385$.

4.4 L-χωρίο

Οι προηγούμενες συναρτήσεις ήταν πολυώνυμα χαμηλού βαθμού έτσι ώστε οι υψηλής τάξης μέθοδοι να βρίσκουν την ακριβή λύση. Θα κατασκευάσουμε τώρα ένα αριθμητικό παράδειγμα σε ένα πιο γενικό χωρίο και με τη συνάρτηση να μην είναι πλέον πολυώνυμο.

Έστω $\Omega = [-1,1]^2 \setminus (0,1) \times (-1,0)$ και έστω το πρόβλημα να βρεθεί u έτσι ώστε $-\Delta u = 2\pi^2 \sin(\pi x)\sin(\pi y)$ στο Ω με $u = 0$ στο $\partial\Omega$. Η ακριβής λύση αυτού του προβλήματος είναι $u(x,y) = \sin(\pi x)\sin(\pi y)$.



Σχήμα 4.13: Το χωρίο $\Omega = [-1,1]^2 \setminus (0,1) \times (-1,0)$ και το πλέγμα με $h = 0.23603$.

h	Nodes	Elements
0.23603	143	244
0.11802	529	976
0.05901	2033	3904
0.02950	7969	15616
0.01475	31553	62464
0.00738	125569	249856
0.00369	500993	999424

Πίνακας 4.15: Τα πλέγματα που θα χρησιμοποιήσουμε για το χωρίο του σχήματος 4.13.

Η λύση πλέον δεν είναι πολυώνυμο, συνεπώς δεν περιμένουμε καμία μέθοδος να βρει την ακριβή λύση. Πράγματι, όπως βλέπουμε στον πίνακα 4.16 και οι τέσσερις μέθοδοι αφήνουν σφάλμα όμως η τάξη σύγκλισης ταυτίζεται πλέον με τα θεωρητικά δεδομένα. Συγκεκριμένα έχουμε τάξη σύγκλισης που τείνει στο 1,2,3

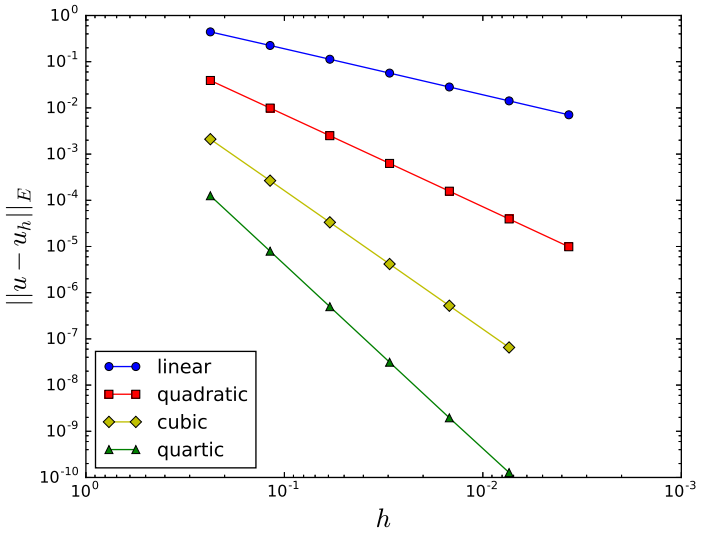
και 4 για τις μεθόδους πεπερασμένων στοιχείων με πολυωνυμικές βάσεις $1^{ου}$ έως $4^{ου}$ βαθμού αντίστοιχα.

	$\|(u-u_h)\|_E$			
h	Linear	Quadratic	Cubic	Quartic
0.23603	0.442223	0.039252	0.002102	0.000125
0.11802	0.224584	0.009924	0.000266	7.92e-06
0.05901	0.113057	0.002502	3.34e-05	4.98e-07
0.02950	0.056665	0.000628	4.17e-06	3.12e-08
0.01475	0.028354	0.000158	5.22e-07	1.96e-09
0.00738	0.014181	3.94e-05	6.52e-08	1.28e-10
0.00369	0.007091	9.86e-06		

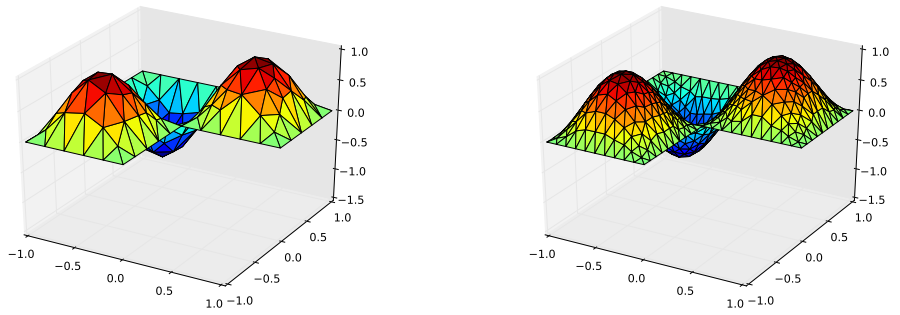
Πίνακας 4.16: Σφάλματα προσέγγισης μετρημένα στην ενεργειακή νόρμα.

Linear	Quadratic	Cubic	Quartic
0.97753	1.98377	2.98173	3.98674
0.99022	1.98794	2.99556	3.99094
0.99652	1.99311	2.99902	3.99478
0.99888	1.99635	2.99982	3.99723
0.99966	1.99813	2.99999	3.93810
0.99990	1.99905		

Πίνακας 4.17: Τάξη σύγκλισης p.

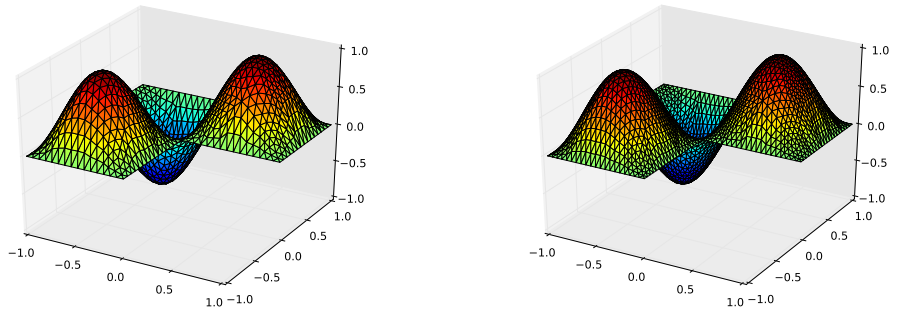


Σχήμα 4.14: Σφάλματα προσέγγισης συναρτήσει του h για τις τέσσερις μεθόδους.



Linear

Quadratic



Cubic

Quartic

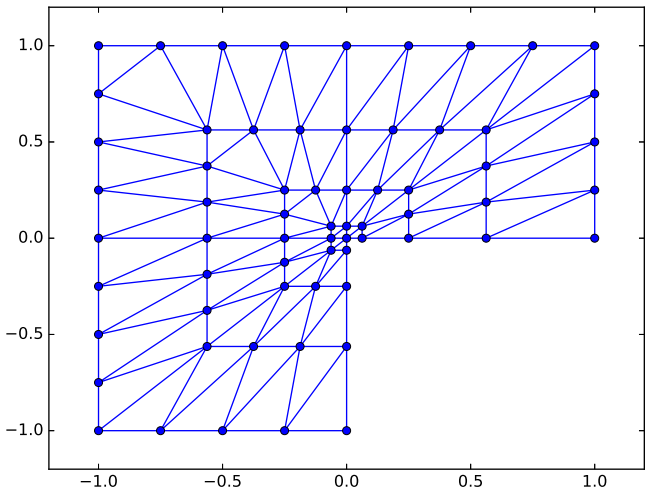
Σχήμα 4.15: Η λύση στο πλέγμα με $h = 0.23603$.

4.5 L-χωρίο με ιδιάζον πρόβλημα (singularity)

Στο ίδιο χωρίο, $\Omega = [-1,1]^2 \setminus (0,1) \times (-1,0)$, Θα λύσουμε το πρόβλημα Laplace

$$\begin{aligned} -\Delta u &= 0 \quad \text{στο } \Omega \\ u &= r^{\frac{2}{3}} \sin\left(\frac{2\theta}{3}\right) \quad \text{στο } \partial\Omega, \end{aligned}$$

όπου $r = \sqrt{x^2+y^2}$ και $\theta = \arctan\left(\frac{y}{x}\right)$. Η ακριβής λύση αυτού του προβλήματος είναι $u(x,y) = r^{\frac{2}{3}} \sin\left(\frac{2\theta}{3}\right)$ και έχουμε $u \in H^{\frac{5}{3}-\epsilon} \forall\, \epsilon > 0$. Θα χρησιμοποιήσουμε τα πλέγματα από το προηγούμενο πρόβλημα καθώς και adapted πλέγματα όπως αυτό του σχήματος 4.16. Επειδή τα adapted πλέγματα πυκνώνουν ανομοιογενώς (έχουμε περισσότερους κόμβους στο ιδιάζον σημείο–singular point) θα μετρήσουμε τα σφάλματα όχι συναρτήσει του h αλλά συναρτήσει των βαθμών ελευθερίας (DoF).



Σχήμα 4.16: Το χωρίο $\Omega = [-1,1]^2 \setminus (0,1) \times (-1,0)$ και το adapted πλέγμα με 33 βαθμούς ελευθερίας.

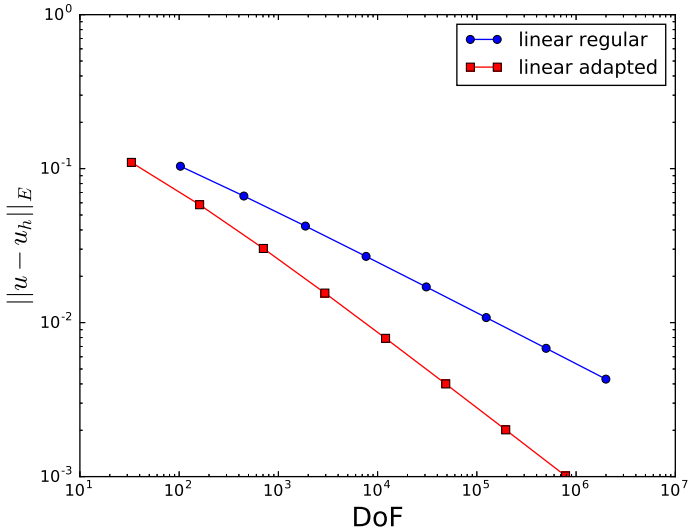
Όπως βλέπουμε στον πίνακα 4.19, με τα regular πλέγματα δεν επιτυγχάνουμε τη βέλτιστη τάξη σύγκλισης, αλλά τάξη $-\frac{1}{3}$. Αντιθέτως με τα adapted πλέγματα η τάξη σύγκλισης είναι η βέλτιστη $\left(-\frac{1}{2}\right)$ συναρτήσει των βαθμών ελευθερίας. Για μεθόδους ανώτερης τάξης, για να επιτύχουμε τη βέλτιστη τάξη θα έπρεπε να χρησιμοποιήσουμε adapted πλέγματα που θα πυκνώνουν ακόμη γρηγορότερα προς το ιδιάζον σημείο. Συγκεκριμένα με τα ίδια adapted πλέγματα που χρησιμοποιήσαμε για τις γραμμικές συναρτήσεις βάσης αλλά χρησιμοποιώντας τετραγωνικές συναρτήσεις βάσης τα αριθμητικά αποτελέσματα έδειξαν τάξη σύγκλισης $-\frac{2}{3}$ ενώ η βέλτιστη τάξη είναι -1.

Regular		Adapted	
DoF	$\|(u-u_h)\|_E$	DoF	$\|(u-u_h)\|_E$
103	0.103672	33	0.109851
449	0.066406	161	0.058279
1873	0.042374	705	0.030330
7649	0.026928	2945	0.015563
30913	0.017061	12033	0.007910
124289	0.010787	48641	0.003996
498433	0.006812	195585	0.002011
1996289	0.004297	784385	0.001010

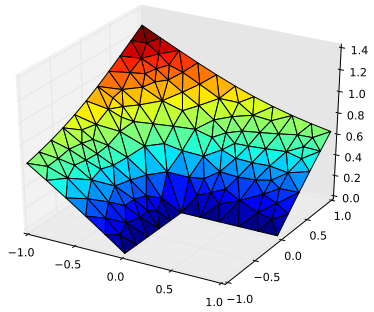
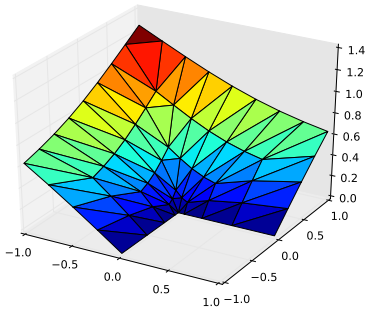
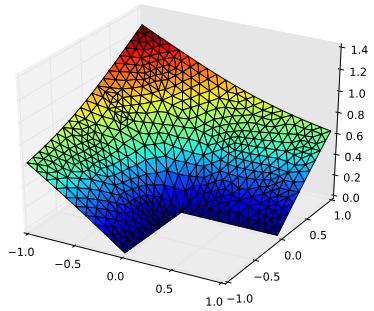
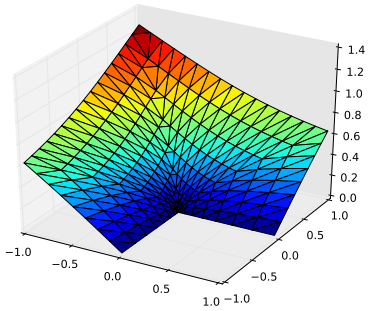
Πίνακας 4.18: Σφάλματα προσέγγισης μετρημένα στην ενεργειακή νόρμα για γραμμικές συναρτήσεις βάσης.

Regular	Adapted
-0.32132	-0.45725
-0.32406	-0.47111
-0.32704	-0.48134
-0.32922	-0.48815
-0.33070	-0.49252
-0.33160	-0.49522
-0.33234	-0.49709

Πίνακας 4.19: Τάξη σύγκλισης p συναρτήσει των βαθμών ελευθερίας για γραμμικές συναρτήσεις βάσης.



Σχήμα 4.17: Σφάλματα προσέγγισης συναρτήσει των βαθμών ελευθερίας για γραμμικές συναρτήσεις βάσης.

Regular	Adapted
	
DoF: 103	DoF: 33
	
DoF: 449	DoF: 161

Σχήμα 4.18: Η λύση στο regular και στο adapted πλέγμα για γραμμικές συναρτήσεις βάσης.

Κεφάλαιο 5

Συμπεράσματα

Σε προβλήματα που προσεγγίζουμε με τη μέθοδο πεπερασμένων στοιχείων, ειδικά στις τρεις διαστάσεις ή σε προβλήματα με πολλούς βαθμούς ελευθερίας που η επίλυση του γραμμικού συστήματος με κάποια επαναληπτική μέθοδο είναι μονόδρομος, η κατασκευή του είναι αυτή που κυριαρχεί στον συνολικό χρόνο της επίλυσης. Αυτό γίνεται ιδιαίτερα εμφανές σε χρονοεξαρτώμενα προβλήματα, που οι πίνακες δύναται να είναι διαφορετικοί σε κάθε χρονικό βήμα (προβλήματα σε εξελισσόμενα χωρία, μη γραμμικά προβλήματα κτλ.) και σε προβλήματα που επιλύονται με προσαρμοστικές μεθόδους και συνεπώς οι κόμβοι αλλάζουν σε κάθε επανάληψη.

Στην εργασία αυτή είδαμε ότι η χρήση της κάρτας γραφικών για την παράλληλη κατασκευή του γραμμικού συστήματος μπορεί να επιταχύνει τον υπολογισμό κατά πολλές τάξεις μεγέθους. Συγκεκριμένα, ακόμη και για τα μεγαλύτερα προβλήματα που προσεγγίσαμε (της τάξης των μερικών εκατομμυρίων βαθμών ελευθερίας), και σε αντίθεση με τους αντίστοιχους σειριακούς υπολογισμούς στον επεξεργαστή, "αποτύχαμε" να δούμε γραμμική αύξηση στον χρόνο κατασκευής των πινάκων και ειδικά για τα πολύ μεγάλα προβλήματα οι μεταφορές των αποτελεσμάτων από την κάρτα γραφικών στη μνήμη RAM ήταν αυτές που κυριάρχησαν. Το περιορισμένο εύρος ζώνης του διαύλου PCI Express μας δίνει το κίνητρο για περαιτέρω έρευνα όσον αφορά την επίλυση του συστήματος απευθείας στη GPU, με επαναληπτικές μεθόδους, ώστε αποφύγουμε τις χρονοβόρες μεταφορές των δεδομένων στη μνήμη RAM.

Παράρτημα

Κεφάλαιο 2.3

Πίνακες κόμβων και βαρών

Θέλουμε οι πίνακες του αριστερού μέλους, stiffness και mass, να υπολογίζονται ακριβώς. Θέλουμε δηλαδή να υπολογίζονται με ακρίβεια τα ολοκληρώματα

$$\int_\tau \nabla\varphi_j \cdot \nabla\varphi_i \,\mathrm{d}V \quad \text{και} \quad \int_\tau \varphi_j \cdot \varphi_i \,\mathrm{d}V$$

όπου φ_i οι συναρτήσεις βάσης της μεθόδου μας και $\tau \in \mathcal{T}$ τα τρίγωνα της τριγωνοποίησής μας. Σημειώνουμε ότι έξω από κάθε τρίγωνο οι συναρτήσεις βάσης παίρνουν την τιμή 0 γι' αυτό δεν χρειάζεται να ολοκληρώσουμε σε όλο το Ω.

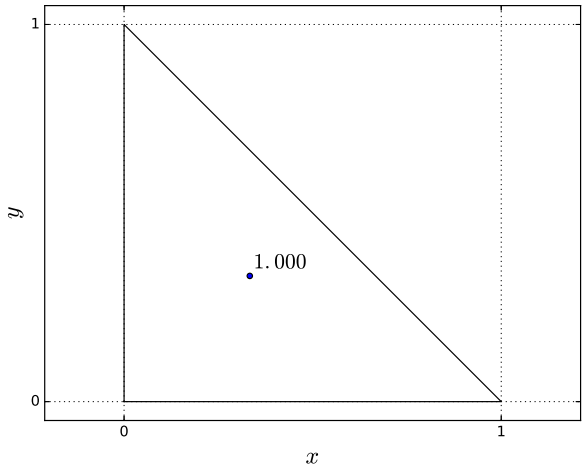
Επειδή οι συναρτήσεις βάσης που θα χρησιμοποιήσουμε είναι τα πολυώνυμα $1^{\text{ου}}$ έως και $4^{\text{ου}}$ βαθμού σε δύο διαστάσεις, θα χρειαστούμε τύπους αριθμητικής ολοκλήρωσης για πολυώνυμα έως και $8^{\text{ου}}$ βαθμού. Για παράδειγμα χρησιμοποιώντας συναρτήσεις βάσης $4^{\text{ου}}$ βαθμού θα χρειαστούμε τύπους αριθμητικής ολοκλήρωσης $6^{\text{ου}}$ βαθμού για τον πίνακα stiffness και $8^{\text{ου}}$ βαθμού για τον πίνακα mass.

w_i	$\hat{x}_i$	$\hat{y}_i$
1	$\frac{1}{3}$	$\frac{1}{3}$

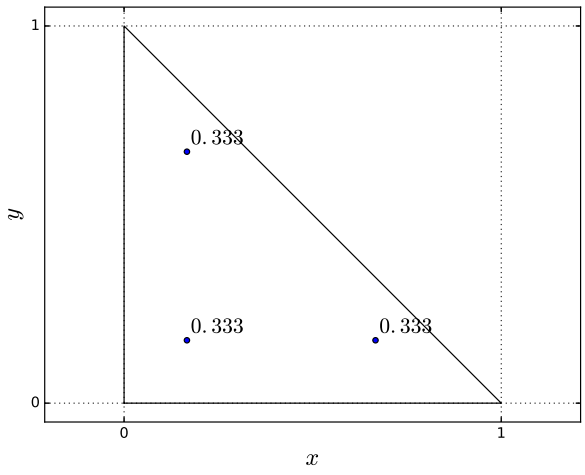
Πίνακας 5.1: Κόμβοι και βάρη για πολυώνυμα $1^{\text{ου}}$ βαθμού.

w_i	$\hat{x}_i$	$\hat{y}_i$
$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$
$\frac{1}{3}$	$\frac{1}{6}$	$\frac{2}{3}$
$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{6}$

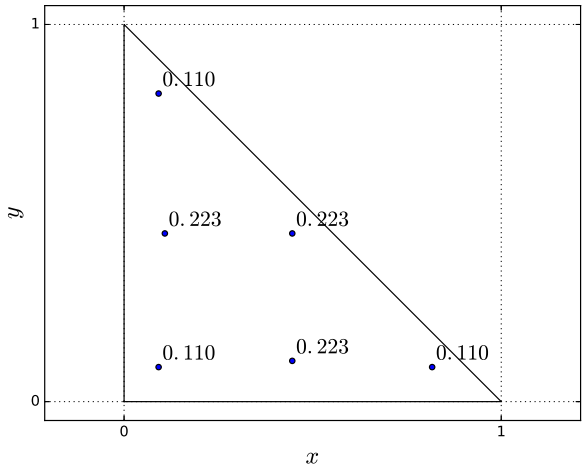
Πίνακας 5.2: Κόμβοι και βάρη για πολυώνυμα $2^{\text{ου}}$ βαθμού.



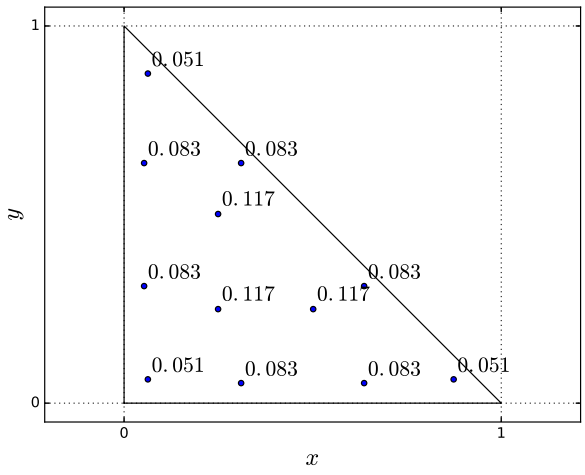
Σχήμα 5.1: Κόμβος για πολυώνυμα $1^{ου}$ βαθμού (δίπλα στον κόμβο το αντίστοιχο βάρος).



Σχήμα 5.2: Κόμβοι για πολυώνυμα $2^{ου}$ βαθμού (δίπλα σε κάθε κόμβο το αντίστοιχο βάρος στρογγυλοποιημένο σε τρία δεκαδικά ψηφία).



Σχήμα 5.3: Κόμβοι για πολυώνυμα $4^{ου}$ βαθμού (δίπλα σε κάθε κόμβο το αντίστοιχο βάρος στρογγυλοποιημένο σε τρία δεκαδικά ψηφία).



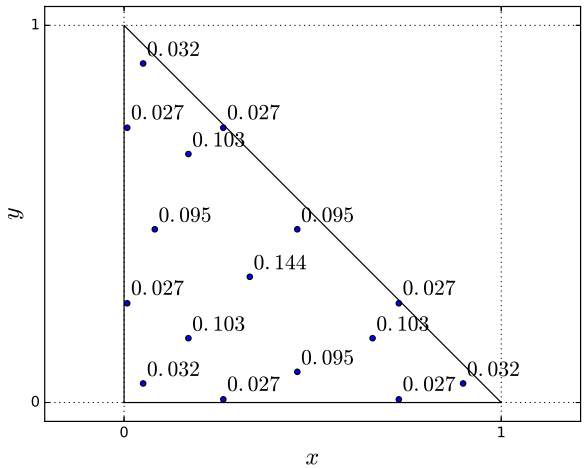
Σχήμα 5.4: Κόμβοι για πολυώνυμα $6^{ου}$ βαθμού (δίπλα σε κάθε κόμβο το αντίστοιχο βάρος στρογγυλοποιημένο σε τρία δεκαδικά ψηφία).

w_i	$\hat{x}_i$	$\hat{y}_i$
0.109951743655322	0.091576213509771	0.091576213509771
0.109951743655322	0.816847572980459	0.091576213509771
0.109951743655322	0.091576213509771	0.816847572980459
0.223381589678011	0.445948490915965	0.108103018168070
0.223381589678011	0.445948490915965	0.445948490915965
0.223381589678011	0.108103018168070	0.445948490915965

Πίνακας 5.3: Κόμβοι και βάρη για πολυώνυμα $4^{ου}$ βαθμού.

w_i	$\hat{x}_i$	$\hat{y}_i$
0.116786275726379	0.249286745170910	0.249286745170910
0.116786275726379	0.249286745170910	0.501426509658179
0.116786275726379	0.501426509658179	0.249286745170910
0.050844906370207	0.063089014491502	0.063089014491502
0.050844906370207	0.063089014491502	0.873821971016996
0.050844906370207	0.873821971016996	0.063089014491502
0.082851075618374	0.053145049844817	0.310352451033784
0.082851075618374	0.053145049844817	0.636502499121399
0.082851075618374	0.310352451033784	0.053145049844817
0.082851075618374	0.636502499121399	0.053145049844817
0.082851075618374	0.310352451033784	0.636502499121399
0.082851075618374	0.636502499121399	0.310352451033784

Πίνακας 5.4: Κόμβοι και βάρη για πολυώνυμα $6^{ου}$ βαθμού.



Σχήμα 5.5: Κόμβοι για πολυώνυμα $8^{ου}$ βαθμού (δίπλα σε κάθε κόμβο το αντίστοιχο βάρος στρογγυλοποιημένο σε τρία δεκαδικά ψηφία).

w_i	$\hat{x}_i$	$\hat{y}_i$
0.144315607677787	0.333333333333333	0.333333333333333
0.095091634267285	0.459292588292723	0.459292588292723
0.095091634267285	0.459292588292723	0.081414823414554
0.095091634267285	0.081414823414554	0.459292588292723
0.103217370534718	0.170569307751760	0.170569307751760
0.103217370534718	0.170569307751760	0.658861384496480
0.103217370534718	0.658861384496480	0.170569307751760
0.032458497623198	0.050547228317031	0.050547228317031
0.032458497623198	0.050547228317031	0.898905543365938
0.032458497623198	0.898905543365938	0.050547228317031
0.027230314174435	0.008394777409958	0.263112829634638
0.027230314174435	0.008394777409958	0.728492392955404
0.027230314174435	0.263112829634638	0.728492392955404
0.027230314174435	0.263112829634638	0.008394777409958
0.027230314174435	0.728492392955404	0.008394777409958
0.027230314174435	0.728492392955404	0.263112829634638

Πίνακας 5.5: Κόμβοι και βάρη για πολυώνυμα $8^{ου}$ βαθμού.

Κεφάλαιο 3.2

```
#include <stdio.h>
#include <math.h>
__global__ void assembly(double coords[][2],
                         int *elems,
                         int Nbases,
                         int Ngauss1,
                         int Ngauss2,
                         int NT,
                         double pts1[][2],
                         double pts2[][2],
                         double *w1,
                         double *w2,
                         double *gphi,
                         double *phi,
                         int *brow,
                         double *bval,
                         int *Arow,
                         int *Acol,
                         double *Aval)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;

    if (idx >= NT){
        return;
    }

    double B[2][2] = {{coords[elems[Nbases*idx + 1]-1][0] -
                       coords[elems[Nbases*idx]-1][0], /*x2-x1*/
                       coords[elems[Nbases*idx + 2]-1][0] -
                       coords[elems[Nbases*idx]-1][0]}, /*x3-x1*/
                      {coords[elems[Nbases*idx + 1]-1][1] -
                       coords[elems[Nbases*idx]-1][1], /*y2-y1*/
                       coords[elems[Nbases*idx + 2]-1][1] -
                       coords[elems[Nbases*idx]-1][1]}}; /*y3-y1*/
    double C[2] = {coords[elems[Nbases*idx]-1][0], coords[elems[Nbases*idx
    ]-1][1]};
    double detB = B[0][0]*B[1][1] - B[0][1]*B[1][0];
    double invtrB[2][2] = {{B[1][1]/detB, -B[1][0]/detB},
                           {-B[0][1]/detB, B[0][0]/detB}};

    detB = fabs(detB);

    double p[2];
    double f;
    double temp;
    double sum;
    double tmp1[2];
    double tmp2[2];
    int i, j, k;

    for (i = 0; i < Nbases; i++){

        temp = 0.0;
        for (k = 0; k < Ngauss2; k++){
            p[0] = B[0][0]*pts2[k][0] + B[0][1]*pts2[k][1] + C[0];
            p[1] = B[1][0]*pts2[k][0] + B[1][1]*pts2[k][1] + C[1];

           f = 2 * pow(M_PI, 2) * sin(M_PI*p[0]) * sin(M_PI*p[1]);

            temp += w2[k] * f * phi[Ngauss2*i + k];
        }
        temp *= 0.5 * detB;

```

```
        brow[Nbases*idx + i] = elems[Nbases*idx + i]-1;
        bval[Nbases*idx + i] = temp;

        for (j = i; j < Nbases; j++){
            sum = 0.0;
            for (k = 0; k < Ngauss1; k++){
                tmp1[0] = invtrB[0][0]*gphi[2*Ngauss1*i+2*k] +
                          invtrB[0][1]*gphi[2*Ngauss1*i+2*k+1];
                tmp1[1] = invtrB[1][0]*gphi[2*Ngauss1*i+2*k] +
                          invtrB[1][1]*gphi[2*Ngauss1*i+2*k+1];
                tmp2[0] = invtrB[0][0]*gphi[2*Ngauss1*j+2*k] +
                          invtrB[0][1]*gphi[2*Ngauss1*j+2*k+1];
                tmp2[1] = invtrB[1][0]*gphi[2*Ngauss1*j+2*k] +
                          invtrB[1][1]*gphi[2*Ngauss1*j+2*k+1];
                sum += w1[k] * (tmp1[0]*tmp2[0] + tmp1[1]*tmp2[1]);
            }
            sum *= 0.5 * detB;

            Arow[Nbases*Nbases*idx + Nbases*i + j] = elems[Nbases*idx + i
    ]-1;
            Acol[Nbases*Nbases*idx + Nbases*i + j] = elems[Nbases*idx + j
    ]-1;
            Aval[Nbases*Nbases*idx + Nbases*i + j] = sum;

            if (i != j){
                Arow[Nbases*Nbases*idx + Nbases*j + i] = elems[Nbases*idx +
     j]-1;
                Acol[Nbases*Nbases*idx + Nbases*j + i] = elems[Nbases*idx +
     i]-1;
                Aval[Nbases*Nbases*idx + Nbases*j + i] = sum;
            }
        }
    }
}
```

```
__global__ void edgecreation(int NT,
                             int edges[][2],
                             int elems[][3])
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    int i, node1, node2;

    if (idx >= NT){
        return;
    }
    for (i=0; i < 3; i++){
        node1 = elems[idx][i%3];
        node2 = elems[idx][(i+1)%3];

        if (node1 < node2){
            edges[3*idx + i][0] = node1;
            edges[3*idx + i][1] = node2;
        }
        else{
            edges[3*idx + i][0] = node2;
            edges[3*idx + i][1] = node1;
        }
    }

}
```

```
__global__ void newnodes(int NE,
                         int old_N,
                         double coords[][2],
                         int edges[][2],
```

```
                          int mode,
                          int *edgenodesrel)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    int node1, node2;
    double x1, x2, y1, y2;
    if (idx >= NE){
        return;
    }
    node1 = edges[idx][0];
    node2 = edges[idx][1];
    x1 = coords[node1-1][0];
    y1 = coords[node1-1][1];
    x2 = coords[node2-1][0];
    y2 = coords[node2-1][1];
    if (mode == 2){
        coords[old_N + idx][0] = (x1 + x2)/2;
        coords[old_N + idx][1] = (y1 + y2)/2;
        edgenodesrel[idx] = old_N + idx + 1;
    }
    else if (mode == 3){
        coords[old_N + 2*idx][0] = (2*x1 + x2)/3;
        coords[old_N + 2*idx][1] = (2*y1 + y2)/3;
        edgenodesrel[2*idx] = old_N + 2*idx + 1;

        coords[old_N + 2*idx + 1][0] = (x1 + 2*x2)/3;
        coords[old_N + 2*idx + 1][1] = (y1 + 2*y2)/3;
        edgenodesrel[2*idx + 1] = old_N + 2*idx + 2;
    }
    else{
        coords[old_N + 3*idx][0] = (3*x1 + x2)/4;
        coords[old_N + 3*idx][1] = (3*y1 + y2)/4;
        edgenodesrel[3*idx] = old_N + 3*idx + 1;

        coords[old_N + 3*idx + 1][0] = (x1 + x2)/2;
        coords[old_N + 3*idx + 1][1] = (y1 + y2)/2;
        edgenodesrel[3*idx + 1] = old_N + 3*idx + 2;

        coords[old_N + 3*idx + 2][0] = (x1 + 3*x2)/4;
        coords[old_N + 3*idx + 2][1] = (y1 + 3*y2)/4;
        edgenodesrel[3*idx + 2] = old_N + 3*idx + 3;
    }

}
```

```
#include <math.h>
#include <stdio.h>
__global__ void finalnodes(int old_N,
                           int NE,
                           int NT,
                           int mode,
                           int intnodesCount,
                           double d_coordinates[][2],
                           int d_elements[][3],
                           int *d_edgenodesrel,
                           int d_sorted_edges[][2],
                           int *d_trinodesrel)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;

    if (idx >= NT){
        return;
    }
    double x1, y1, x2, y2, x3, y3, x4, y4;
    int i, j, T1, T2, L, R, M, order;
    for (i=0; i<3; i++){
```

```
        if (d_elements[idx][i%3] < d_elements[idx][(i+1)%3]){
            T1 = d_elements[idx][i%3];
            T2 = d_elements[idx][(i+1)%3];
            order = 0;
        }
        else{
            T2 = d_elements[idx][i%3];
            T1 = d_elements[idx][(i+1)%3];
            order = 1;
        }
        //Binary search
        L = 0;
        R = NE-1;
        while (L <= R){
            M = L + floor(double(R-L)/2);
            if (d_sorted_edges[M][0] == T1){
                break;
            }
            else if (d_sorted_edges[M][0] < T1){
                L = M + 1;
            }
            else{
                R = M - 1;
            }
        }
        //Serial search
        if (d_sorted_edges[M][1] == T2){

        }
        else if (d_sorted_edges[M][1] < T2){
            M++;
            while (d_sorted_edges[M][1] < T2){
                M++;
            }
        }
        else{
            M--;
            while (d_sorted_edges[M][1] > T2){
                M--;
            }
        }
        if (order == 0){
            for (j=0; j<mode-1; j++){
                d_trinodesrel[idx*((mode-1)*3+intnodesCount) + (mode-1)*i +
 j] = d_edgenodesrel[M*(mode-1) + j];
            }
        }
        else{
            for (j=0; j<mode-1; j++){
                d_trinodesrel[idx*((mode-1)*3+intnodesCount) + (mode-1)*i +
 j] = d_edgenodesrel[M*(mode-1) + (mode-2) - j];
            }
        }
    }
    if (mode == 3){
        x1 = d_coordinates[d_elements[idx][0]-1][0];
        y1 = d_coordinates[d_elements[idx][0]-1][1];
        x2 = d_coordinates[d_elements[idx][1]-1][0];
        y2 = d_coordinates[d_elements[idx][1]-1][1];
        x3 = d_coordinates[d_elements[idx][2]-1][0];
        y3 = d_coordinates[d_elements[idx][2]-1][1];

        d_coordinates[old_N+idx][0] = (x1+x2+x3)/3;
        d_coordinates[old_N+idx][1] = (y1+y2+y3)/3;

```

```
        d_trinodesrel[idx*7 + 6] = old_N+idx+1;
    }
    else if (mode == 4){
        x1 = d_coordinates[d_trinodesrel[idx*12 + 8]-1][0];
        y1 = d_coordinates[d_trinodesrel[idx*12 + 8]-1][1];
        x2 = d_coordinates[d_trinodesrel[idx*12 + 3]-1][0];
        y2 = d_coordinates[d_trinodesrel[idx*12 + 3]-1][1];
        x3 = d_coordinates[d_trinodesrel[idx*12 + 7]-1][0];
        y3 = d_coordinates[d_trinodesrel[idx*12 + 7]-1][1];
        x4 = d_coordinates[d_trinodesrel[idx*12 + 4]-1][0];
        y4 = d_coordinates[d_trinodesrel[idx*12 + 4]-1][1];

        d_coordinates[old_N+3*idx][0] = (2*x1+x2)/3;
        d_coordinates[old_N+3*idx][1] = (2*y1+y2)/3;
        d_coordinates[old_N+3*idx+1][0] = (x1+2*x2)/3;
        d_coordinates[old_N+3*idx+1][1] = (y1+2*y2)/3;
        d_coordinates[old_N+3*idx+2][0] = (x3+x4)/2;
        d_coordinates[old_N+3*idx+2][1] = (y3+y4)/2;

        d_trinodesrel[idx*12 + 9] = old_N+3*idx+1;
        d_trinodesrel[idx*12 + 10] = old_N+3*idx+2;
        d_trinodesrel[idx*12 + 11] = old_N+3*idx+3;
    }
}
```

```
#include <stdio.h>
#include <math.h>
__global__ void energynorm(double coords[][2],
                           int *elems,
                           int Nbases,
                           int Ngauss,
                           int NT,
                           double pts[][2],
                           double *w,
                           double *gphi,
                           double *u,
                           double *d_list1)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;

    if (idx >= NT){
        return;
    }

    double B[2][2] = {{coords[elems[Nbases*idx + 1]-1][0] - coords[elems[
    Nbases*idx]-1][0], /*x2-x1*/
                       coords[elems[Nbases*idx + 2]-1][0] - coords[elems[
    Nbases*idx]-1][0]}, /*x3-x1*/
                      {coords[elems[Nbases*idx + 1]-1][1] - coords[elems[
    Nbases*idx]-1][1], /*y2-y1*/
                       coords[elems[Nbases*idx + 2]-1][1] - coords[elems[
    Nbases*idx]-1][1]}}; /*y3-y1*/
    double C[2] = {coords[elems[Nbases*idx]-1][0], coords[elems[Nbases*idx
    ]-1][1]};
    double detB = B[0][0]*B[1][1] - B[0][1]*B[1][0];
    double invtrB[2][2] = {{B[1][1]/detB, -B[1][0]/detB},
                           {-B[0][1]/detB, B[0][0]/detB}};
    detB = fabs(detB);

    double tmp;
    double p[2];
    double exactgrad[2];
    double total = 0.0;
    int i, j;
    for (i = 0; i < Ngauss; i++){
        double sum[2] = {0.0, 0.0};
```

```
        for (j = 0; j < Nbases; j++){
            sum[0] += u[elems[Nbases*idx + j]-1] * gphi[2*Ngauss*j+2*i];
            sum[1] += u[elems[Nbases*idx + j]-1] * gphi[2*Ngauss*j+2*i+1];
        }
        tmp = invtrB[0][0]*sum[0] + invtrB[0][1]*sum[1];
        sum[1] = invtrB[1][0]*sum[0] + invtrB[1][1]*sum[1];
        sum[0] = tmp;

        p[0] = B[0][0]*pts[i][0] + B[0][1]*pts[i][1] + C[0];
        p[1] = B[1][0]*pts[i][0] + B[1][1]*pts[i][1] + C[1];

        exactgrad[0] = M_PI*sin(M_PI*p[1])*cos(M_PI*p[0]);
        exactgrad[1] = M_PI*sin(M_PI*p[0])*cos(M_PI*p[1]);

        sum[0] = exactgrad[0] - sum[0];
        sum[1] = exactgrad[1] - sum[1];

        tmp = sum[0]*sum[0] + sum[1]*sum[1];
        total += 0.5*detB*w[i]*tmp;
    }

    d_list1[idx] = total;
}
```

Βιβλιογραφία

[1] Manolis Georgoulis, Andrea Cangiani, *Computational Methods for Partial Differential Equations. Part III: Introduction to Finite Element Methods for PDEs (lecture notes)*, November 2015.

[2] Vassilios A. Dougalis, *Finite Element Methods for the Numerical Solution of Partial Differential Equations (lecture notes)*, 2013.

[3] Claes Johnson, *Numerical Solution of Partial Differential Equations by the Finite Element Method*, Dover Publications, 2009.

[4] Endre Süli, *Lecture Notes on Finite Element Methods for Partial Differential Equations*, December 2012.

[5] Long Chen, *Programming of Finite Element Methods in Matlab (lecture notes).*

[6] D. A. Dunavant, *High degree efficient symmetrical Gaussian quadrature rules for the triangle*, International Journal for Numerical Methods in Engineering, 21:1129-1148 (1985).

[7] Dr. Shaozhong Deng, *Quadrature Formulas in Two Dimensions (lecture notes)*, University of North Carolina at Charlotte, Spring 2010.