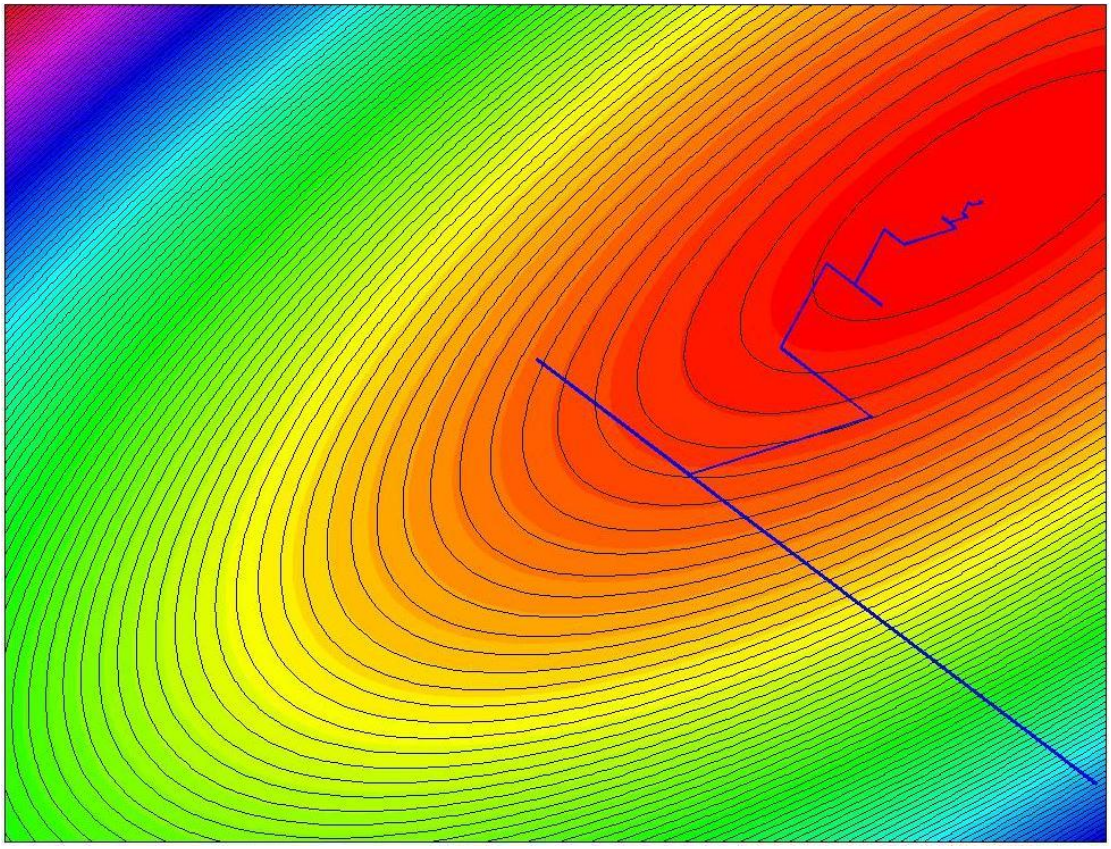
**ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ**

Τμήμα Μηχανικών Παραγωγής & Διοικησης

**Διπλωματική εργασία**

# «Διερεύνηση ιδιοτήτων και αποτελεσματικότητας αλγορίθμων βελτιστοποίησης μη-γραμμικών προβλημάτων»



**Γρηγόριος Φούσας**

Χανιά 2010

# Περιεχόμενα

Πρόλογος ..... 3
Εισαγωγή ..... 5
Κεφάλαιο 1: Θεωρία αλγορίθμων μη γραμμικού προγραμματισμού ..... 7
1.1 Γενική προσέγγιση ..... 7
1.2 Δομή Αλγορίθμων Βελτιστοποίησης ..... 11
1.3 Καθορισμός κατεύθυνσης αναζήτησης ..... 11
1.3.1 Μέγιστη κατάβαση ..... 12
1.3.2. Μέθοδοι συζυγών κλίσεων ..... 12
1.3.3. Μέθοδοι Σχεδόν-Newton ..... 13
1.4 Αλγόριθμος αναζήτησης επί γραμμής ..... 14
1.4.1 Φάση εντοπισμού αγκύλης ..... 15
1.4.2 Φάση υποδιαίρεσης της αγκύλης ..... 15
1.4.3. Χρυσή τομή ..... 17
1.5 Επανέναρξη ..... 18
1.6 Κανονικοποίηση ..... 18
1.7 Συνθήκη σύγκλισης ..... 19
1.8 Αλγόριθμος RPROP ..... 20
Κεφάλαιο 2: Υπολογιστικός Κώδικας ..... 23
2.1 Γενικά ..... 23
2.2 Εισαγωγή και εκτύπωση δεδομένων ..... 23
2.3 Διαγράμματα ροής υπορουτινών ..... 23
2.3.1 Διάγραμμα ροής του προγράμματος ..... 24
2.3.2 Διάγραμμα ροής του κυρίου σώματος (main) του προγράμματος ..... 24
2.3.3 Διάγραμμα ροής κατεύθυνσης αναζήτησης ..... 26
2.3.4 Διάγραμμα ροής της υπορουτίνας εντοπισμού της αγκύλης ..... 27
2.3.5 Διάγραμμα ροής της υπορουτίνας υποδιαίρεσης ..... 28
2.3.6 Διάγραμμα ροής του αλγόριθμου RPROP ..... 31
Κεφάλαιο 3: Παρουσίαση Συναρτήσεων ..... 33
3.1 Rosenbrock function ..... 33
3.2 Broyden tridiagonal function ..... 34
3.3 Variably dimensioned function ..... 35
3.4 Nazareth trigonometric function ..... 36
3.5 Zakharov function ..... 37
3.6 Trigonometric function ..... 38

3.7 Dixon & Price function ..... 39
Κεφάλαιο 4: Παρουσίαση αποτελεσμάτων ..... 41
4.1 Rosenbrock function ..... 41
4.2 Broyden tridiagonal function ..... 47
4.3 Variably dimensioned function ..... 51
4.4 Nazareth trigonometric function ..... 52
4.5 Zakharov function ..... 56
4.6 Trigonometric function ..... 58
4.7 Dixon & Price function ..... 60
4.8 Συνολικά αποτελέσματα ..... 64
4.9 Σύγκριση RPROP – Polack Ribiere ..... 66
Κεφάλαιο 5: Συμπεράσματα ..... 67
Βιβλιογραφία ..... 69
Παράρτημα ..... 71
Εγχειρίδιο χρήσης προγράμματος ..... 71
Υπολογιστικός κώδικας ..... 74
Παράδειγμα εισαγωγής συνάρτησης προς λύση ..... 77
Εισαγωγή δεδομένων ..... 77
Κώδικας σε C ..... 79

*Η εικόνα του εξωφύλλου είναι η γραφική λύση του προβλήματος βελτιστοποίησης της συνάρτησης Broyden Tridiagonal με την μέθοδο RPROP.*

# Πρόλογος

Η παρακάτω μελέτη αποτελεί το «γραπτό τμήμα» της συστηματικής έρευνας που εκπόνησα στα πλαίσια της διπλωματικής μου εργασίας για το τμήμα «Μηχανικών Παραγωγής και Διοίκησης» του Πολυτεχνείου Κρήτης. Το σημαντικότερο όμως είναι ότι αποτέλεσε μια εξαιρετική ευκαιρία για εμένα να μελετήσω πιο συστηματικά έναν τομέα των μαθηματικών που με ελκύει ιδιαίτερα: τον τομέα των μη γραμμικών συναρτήσεων σε συνδυασμό πάντα με τις μεθόδους βελτιστοποίησης που συνιστούν ένα ευρύ πεδίο μελέτης και ενδιαφέροντος.

Ο αυξημένος χρόνος που χρειάστηκα για να ολοκληρώσω το παρόν δε συνάδει σε καμία περίπτωση με το ενδιαφέρον μου για το αντικείμενο. Αντιθέτως, οφείλεται αποκλειστικά σε προσωπικούς μου παράγοντες. Για το λόγο αυτό θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου και επιτηρητή της διπλωματικής κύριο Μάρκο Παπαγεωργίου, για την πολύτιμη βοήθειά του, την υποστήριξή του αλλά κυρίως για την υπομονή που επέδειξε.

Εύχομαι, εκτός από τα οφέλη που αποκόμισα εγώ από την έρευνα και τη μετέπειτα συγγραφή της, η εργασία αυτή να αποτελέσει χρήσιμη βάση για πιο ενδελεχή έρευνα στο πεδίο από μελλοντικούς συναδέλφους.

# Εισαγωγή

Η παρούσα εργασία είναι το αποτέλεσμα της συστηματικής διερεύνησης των μεθόδων βελτιστοποίησης μη γραμμικών συναρτήσεων και της σύγκρισης των μεθόδων αυτών με τη νεότερη RPROP, η οποία μέχρι σήμερα δεν έχει αποτελέσει ευρύ αντικείμενο έρευνας ώστε να έχουμε αρκετά στοιχεία για την αποτελεσματικότητά της.

Το ενδιαφέρον μας σχετικά με την RPROP προκύπτει από το γεγονός πως ναι μεν αποτελεί κι αυτή μια επαναληπτική μέθοδο ελαχιστοποίησης, αλλά παρουσιάζει πολύ μικρότερο υπολογιστικό φόρτο ανά επανάληψη από τις υπόλοιπες. Συνεπώς, αποτελεί ενδιαφέρον αντικείμενο προς εξέταση το ενδεχόμενο να αποτελεί η RPROP, παρά τον χαρακτηριστικά υψηλότερο αριθμό επαναλήψεών της, ιδανικότερη μέθοδο βελτιστοποίησης για κάποιες τουλάχιστον μη γραμμικές συναρτήσεις.

Για τον παραπάνω σκοπό χρησιμοποιήθηκε αρχικά το πρόγραμμα που αναπτύχθηκε από τον Ιωάννη Μαρινάκη, σε παλαιότερη διπλωματική εργασία για το Πολυτεχνείο Κρήτης. Ο κώδικας του προαναφερθέντος προγράμματος αναπτύχθηκε και συμπεριέλαβε 7 ακόμα μη γραμμικές συναρτήσεις, ενώ κάποιες αδυναμίες του εντοπίστηκαν και βελτιώθηκαν. Στη συνέχεια, με τον εξελιγμένο πλέον κώδικα, οι συναρτήσεις βελτιστοποιήθηκαν για διάφορα μεγέθη προβλημάτων ενώ κρατήθηκαν οι χρόνοι που χρειάστηκαν από την καθεμία διαφορετική μέθοδο.

Ανεξάρτητα από το πειραματικό κομμάτι, για την καταγραφή των αποτελεσμάτων με τρόπο που να πληρεί τις απαιτήσεις μιας διπλωματικής εργασίας, χρησιμοποιήθηκε η παρακάτω δομή η οποία αναπτύχθηκε σε 5 κεφάλαια.

Στο πρώτο κεφάλαιο συνοψίζεται και επεξηγείται η μαθηματική θεωρία επίλυσης μη γραμμικών συναρτήσεων με τις μεθόδους μέγιστης κατάβασης, σχεδόν Newton, συζυγών κλίσεων και RPROP.

Στο δεύτερο κεφάλαιο αναλύεται ο προγραμματιστικός κώδικας, οι ρουτίνες και οι υπό-ρουτίνες του με σχόλια και διαγράμματα ροής.

Στο τρίτο κεφάλαιο παρουσιάζονται οι χρησιμοποιούμενες μη γραμμικές συναρτήσεις που αναλύονται τόσο αλγεβρικά όσο και διαγραμματικά.

Στο τέταρτο κεφάλαιο παρουσιάζονται τα αποτελέσματα των δοκιμών που εκτελέσθηκαν και κατατάσσονται οι μέθοδοι με βάση την αποτελεσματικότητά τους, ενώ στο πέμπτο κεφάλαιο παρατίθενται τα συμπεράσματα του ερευνητή από το σύνολο της μελέτης. Συνοπτικά αναφέρουμε ότι η αρχική «ερευνητική πρόταση» περί πιθανής υπεροχής της RPROP ισχύει μόνο για συγκεκριμένες μη γραμμικές συναρτήσεις και όχι για το σύνολο.

# Κεφάλαιο 1: Θεωρία αλγορίθμων μη γραμμικού προγραμματισμού

## 1.1 Γενική προσέγγιση

Στην εργασία αυτή θα ασχοληθούμε με το πρόβλημα εύρεσης μιας τοπικής λύσης του προβλήματος

Ελαχιστοποίηση: $f(\mathbf{x}), \mathbf{x} \in R^n$ (1.1)

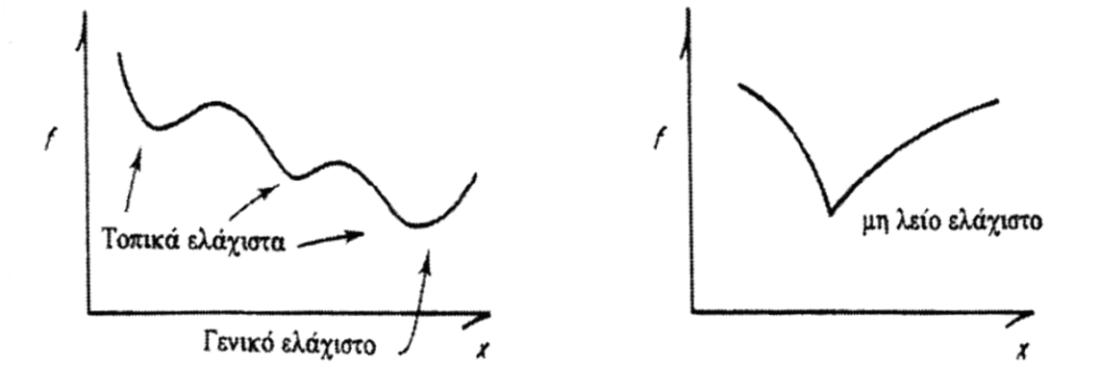
Η $f(\mathbf{x})$ είναι η **αντικειμενική συνάρτηση** (ή **αντικειμενικό κριτήριο**) ενώ το **σημείο ελάχιστου** ή **ελάχιστο** σημειώνεται $\mathbf{x}^*$ .**Προβλήματα μεγιστοποίησης** μπορούν να αναχθούν στο (1.1) με τον απλό μετασχηματισμό.

$$\max_x f(\mathbf{x}) = -\min_x[-f(\mathbf{x})] \quad (1.2)$$

Συνήθως υποθέτουμε ότι το $\mathbf{x}^*$ υπάρχει, είναι μοναδικό και μπορεί να προσδιορισθεί από μια χρησιμοποιούμενη μέθοδο. Για πολλά όμως προβλήματα οι παραδοχές αυτές μπορεί να μην εκπληρούνται. Το $\mathbf{x}^*$ μπορεί να μην υπάρχει αν η $f$ δεν είναι κάτω φραγμένη (π.χ. $f = \mathbf{x}^3$) και, σπανιότερα, ακόμη και αν η $f$ είναι κάτω φραγμένη (π.χ. $f = e^{-\mathbf{x}}$).

Αν το $\mathbf{x}^*$ υπάρχει, μπορεί να μην είναι μοναδικό (π.χ $f = \max${-x-1, 0, x-1} ή $f = \cos\mathrm{x}$). Τότε μπορεί ένα **τοπικό ελάχιστο** να μην είναι **γενικό ελάχιστο** (ή **απόλυτο ελάχιστο**), όπως φαίνεται στο Σχ. 2.1. Τοπικά ελάχιστα μπορεί να υπάρχουν και να παρουσιάζουν ενδιαφέρον ακόμη και αν η $f$ δεν είναι κάτω φραγμένη (π.χ. $f = \mathrm{x}^3 - 3\mathrm{x}$).

Ο προσδιορισμός **γενικών ελάχιστων** δεν είναι θεωρητικά ή πρακτικά εφικτός με βεβαιότητα στη γενική περίπτωση. Ένας πρακτικός τρόπος είναι ο προσδιορισμός πολλών (ή όλων των ) τοπικών ελάχιστων και η επιλογή του βέλτιστου εξ αυτών.




**Σχήμα 1.1**: Τύποι ελαχίστων

Όμως πρέπει να σημειώσουμε ότι η ύπαρξη τοπικών ελάχιστων δε σημαίνει ότι υπάρχει αναγκαστικά γενικό ελάχιστο, όπως για παράδειγμα στη συνάρτηση $f(\mathrm{x}) = (0.5 + \mathrm{x}^2)\mathrm{e}^{-\mathrm{x}^2}$.

Ένα **τοπικό ελάχιστο** ορίζεται ως εξής

$f(\mathbf{x}) \geq f(\mathbf{x}^*)$ για κάθε $\mathbf{x}$ μιας αρκούντως μικρής περιοχής περί το $\mathbf{x}^*$, ενώ ένα **αυστηρό τοπικό ελάχιστο** ως εξής

$f(\mathbf{x}) > f(\mathbf{x}^*)$ για κάθε $\mathbf{x} \neq \mathbf{x}^*$ μιας αρκούντως μικρής περιοχής περί το $\mathbf{x}^*$.

Στην παρούσα εργασία θα θεωρήσουμε μόνο συναρτήσεις όπου οι πρώτες και δεύτερες παράγωγοι υπάρχουν και είναι συνεχείς, αποκλείοντας μη λεία ελάχιστα όπως αυτό του Σχ.1.1.

Έστω το **διάνυσμα μερικών παραγώγων** ή **διάνυσμα κλίσης** της συνάρτησης $f(\mathbf{x}), \mathbf{x} \in \mathrm{R}^{\mathrm{n}}$ ,σε κάθε σημείο $\mathbf{x}$ ορίζεται ως εξής

$$\begin{bmatrix} \frac{\partial f}{\partial \mathrm{x}_1} \\ \frac{\partial f}{\partial \mathrm{x}_2} \\ \vdots \\ \frac{\partial f}{\partial \mathrm{x}_\mathrm{n}} \end{bmatrix}_{\mathbf{x}} = \nabla f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}}\Big|_{\mathbf{x}} \quad (1.3)$$

Εφόσον θεωρούμε ότι $f(\mathbf{x}) \in \mathrm{C}^2$ (συνάρτηση συνεχής και δύο φορές συνεχώς παραγωγίσιμη), υπάρχει ένας πίνακας **δεύτερων μερικών παραγώγων** ή **Χεσιανός πίνακας** $\nabla^2 f(\mathbf{x})$ με στοιχεία (i, j) ίσα προς $\frac{\partial^2 f}{\partial x_i \partial x_j}$. Ο Χεσιανός πίνακας είναι τετραγωνικός και συμμετρικός.

Επιπλέον ορίζουμε σαν γραμμή (Σχ. 1.2) το σύνολο σημείων

$$\mathbf{x}(\alpha) = \mathbf{x}' + \alpha \mathbf{s} \ \ \forall \alpha \in \mathrm{R} \ (1.4)$$

όπου $\mathbf{x}'$ είναι ένα δεδομένο σημείο της γραμμής (για α=0) και **s** είναι η **κατεύθυνση** της γραμμής. Συχνά έχουμε την (1.4) $\forall \alpha \geq 0$,δηλαδή μισή γραμμή.

Η κατεύθυνση **s** μπορεί να τυποποιηθεί έτσι ώστε

$$\|\mathbf{s}\|_2 = \sqrt{\mathbf{s}^{\mathrm{T}}\mathbf{s}} = \sqrt{\sum_i \mathrm{s}_i{}^2} = 1 \ (1.5)$$

πράγμα που δεν αλλάζει τη γραμμή, παρά μόνο τις τιμές α συγκεκριμένων σημείων. Η νόρμα $\|.\|_2$ μετρά το μέγεθος ενός διανύσματος.

Για τον υπολογισμό παραγώγων κατά μήκος της γραμμής $\mathbf{x}(\alpha)$ στην (1.4) έχουμε

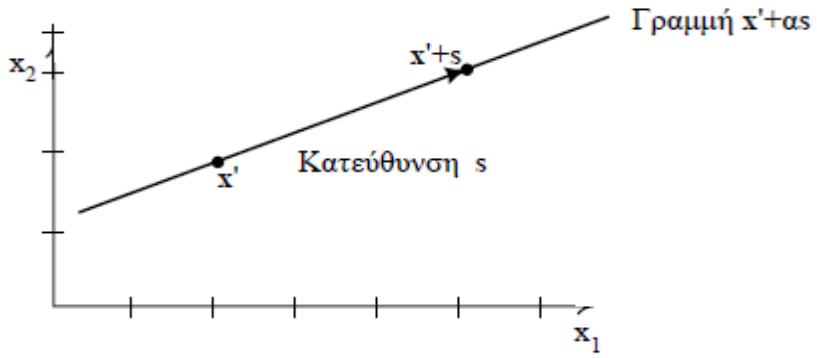
$$\frac{\mathrm{d}}{\mathrm{d}\alpha} = \sum_{\mathrm{i}} \frac{\mathrm{dx_i}(\alpha)}{\mathrm{d}\alpha} \frac{\partial}{\partial \mathrm{x_i}} = \sum_{\mathrm{i}} \mathrm{s_i} \frac{\partial}{\partial \mathrm{x_i}} = \mathbf{s}^{\mathrm{T}} \nabla. \ (1.6)$$

Έτσι, έχουμε την κλίση (πρώτη παράγωγο) της $f(\mathbf{x}(\alpha))$ κατά μήκος της γραμμής $\mathbf{x}(\alpha)$

$$\frac{\mathrm{d}f}{\mathrm{d}\alpha} = \mathbf{s}^{\mathrm{T}} \nabla f = \nabla f^{\mathrm{T}} \mathbf{s} \ (1.7)$$

και την **καμπυλότητα** (δεύτερη παράγωγο)

$$\frac{\mathrm{d}^2 f}{\mathrm{d}\alpha^2} = \frac{\mathrm{d}}{\mathrm{d}\alpha} \frac{\mathrm{d}f}{\mathrm{d}\alpha} = \mathbf{s}^{\mathrm{T}} \nabla (\nabla f^{\mathrm{T}} \mathbf{s}) = \mathbf{s}^{\mathrm{T}} (\nabla^2 f) \mathbf{s}. \ (1.8)$$




**Σχήμα 1.2**: Γραμμή στο χώρο $\mathbf{x} \in \mathrm{R}^2$

Στη συνέχεια θα χρησιμοποιούμε τους συμβολισμούς $\mathbf{g} = \nabla f$και $\mathbf{G} = \nabla^2 f$.

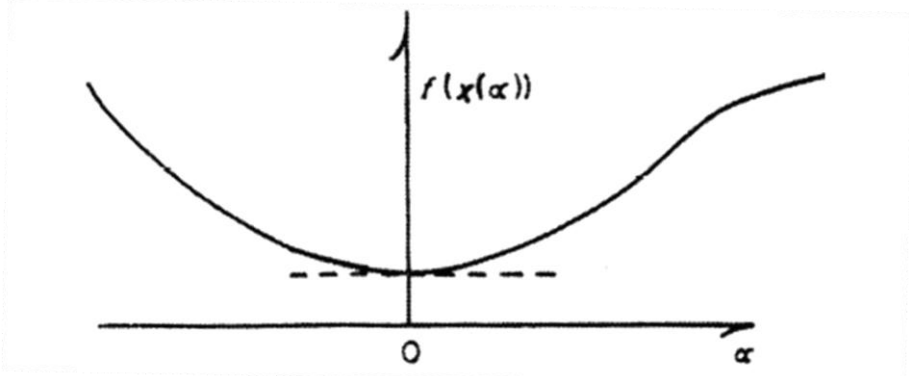
Προφανώς η κλίση και η καμπυλότητα κατά μήκος μιας γραμμής εξαρτώνται από το μέγεθος του **s** εκτός αν υπάρχει τυποποίηση του **s**. Αν, για κάποιο σημείο $\mathbf{x}'$ με διάνυσμα κλίσης $\mathbf{g}'$, θεωρήσουμε όλες τις τυποποιημένες κατευθύνσεις βάση της (1.5) τότε οι κατευθύνσεις $\pm \mathbf{g}'/\|\mathbf{g}'\|_2$ είναι εκείνες με τη μέγιστη και ελάχιστη κλίση όπως συνεπάγεται από τη (1.7). Οι ίδιες κατευθύνσεις είναι ορθογώνιες στο περίγραμμα και το εφαπτόμενο επίπεδο της $f(\mathbf{x})$ στο σημείο $\mathbf{x}'$.

Για τον προσδιορισμό συνθηκών ελαχιστοποίησης παρατηρούμε (Σχ.1.3) ότι για κάθε γραμμή $\mathbf{x}(a) = \mathbf{x}^* + a\mathbf{s}$, πρέπει η συνάρτηση $f(\mathbf{x}(\mathrm{a}))$ να έχει στο $\mathbf{x}^*$ μηδενική κλίση και μη αρνητική καμπυλότητα, δηλαδή, με τις (1.7), (1.8), έχουμε $\mathbf{s}^{\mathrm{T}}\mathbf{g}^* = 0$ και $\mathbf{s}^{\mathrm{T}}\mathbf{G}^*\mathbf{s} \geq 0$ για κάθε $\mathbf{s}$.Θέτοντας $\mathbf{s} = \mathbf{e_1}, \mathbf{s} = \mathbf{e_2}$ κλπ., καταλήγουμε τότε στις ακόλουθες **αναγκαίες συνθήκες τοπικού ελάχιστου** $\mathbf{x}^*$.

$$\mathbf{g}^* = \mathbf{0} \quad (1.9)$$

$$\mathbf{s}^{\mathrm{T}}\mathbf{G}^*\mathbf{s} \geq 0 \ \ \forall \mathbf{s} \quad (1.10)$$

Η (1.9) ονομάζεται **αναγκαία συνθήκη πρώτου βαθμού** και η (1.10) αναγκαία συνθήκη δεύτερου βαθμού. Η τελευταία συνθήκη είναι εξ ορισμού ταυτόσημη με τη συνθήκη ότι ο $\mathbf{G}^*$ είναι ένας **θετικά ημιορισμένος πίνακας**.



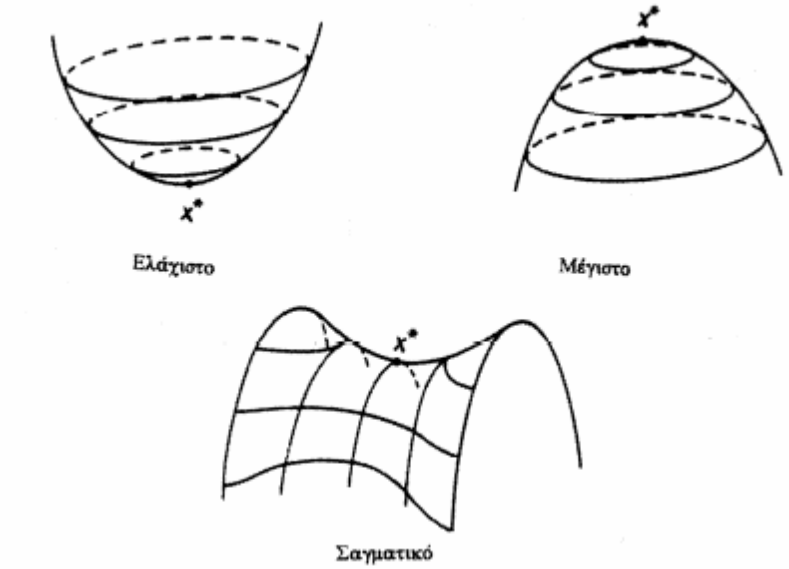

**Σχήμα 1.3**. Μηδενική κλίση και μη αρνητική καμπυλότητα για α=0.

Ικανές συνθήκες για ένα αυστηρό τοπικό ελάχιστο x* είναι η (1.19) και ο $\mathbf{G}^*$ θετικά ορισμένος, δηλαδή

$$\mathbf{s}^{\mathrm{T}}\mathbf{G}^{*}\mathbf{s} > 0 \; \forall \mathbf{s} \neq \mathbf{0} \quad (1.11)$$

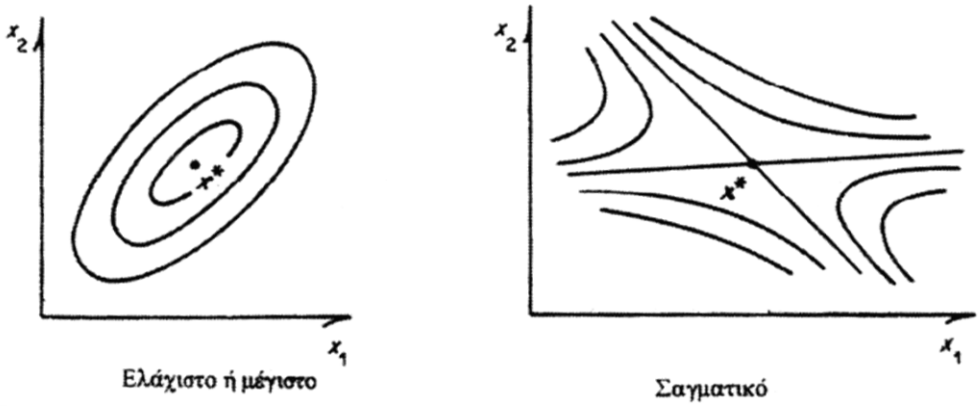
Η διαφορά μεταξύ των αναγκαίων συνθηκών και ικανών συνθηκών ελαχιστοποίησης υφίσταται μόνο σε μερικές περιπτώσεις μηδενικής καμπυλότητας. Επί παραδείγματι, η $f(\mathrm{x}) = \mathrm{x}^4$ έχει ένα απόλυτο ελάχιστο για $\mathrm{x}^* = 0$, αλλά δεν ικανοποιεί τις ικανές συνθήκες, ενώ η $f(\mathrm{x}) = -\mathrm{x}^4$ έχει ένα απόλυτο μέγιστο για $\mathrm{x}^* = 0$ αλλά ικανοποιεί τις αναγκαίες συνθήκες ελαχιστοποίησης.

Τα σημεία που ικανοποιούν τη συνθήκη (1.9) μόνη, λέγονται **στάσιμα σημεία**. Ας σημειωθεί ότι η (1.9) ικανοποιείται από **τοπικά ελάχιστα**, **τοπικά μέγιστα**, και **σαγματικά σημεία**, βλ. Σχ.1.4 και υψομετρικές καμπύλες στο Σχ.1.5.




**Σχήμα 1.4**: Τύποι στάσιμων σημείων.

Παρατηρούμε στο Σχ.1.4 ότι η καμπυλότητα στο $\mathbf{x}^*$ όλων των γραμμών είναι θετική ($\mathbf{G}^* > 0$) για τοπικά ελάχιστα, αρνητική ($\mathbf{G}^* < 0$) για τοπικά μέγιστα και μεικτή ($\mathbf{G}^*$μη ορισμένος) για σαγματικά σημεία.




**Σχήμα 1.5:** Περιγράμματα στάσιμων σημείων.

Η αναγκαία συνθήκη (1.9) αποτελεί ένα μη γραμμικό σύστημα n εξισώσεων με n αγνώστους, τις συνιστώσες $\mathrm{x}_{\mathrm{i}}^*$. Το σύστημα μπορεί να έχει

μία μοναδική ή καμία ή πολλαπλές λύσεις. Στην τελευταία περίπτωση έχουμε πολλά υποψήφια ελάχιστα, οπότε οι (1.10) ή/και (1.11) μπορούν να χρησιμοποιηθούν για τον διαχωρισμό των τοπικών ελάχιστων από τα άλλα στάσιμα σημεία. Η αναλυτική λύση, μέσω επίλυσης του συστήματος εξισώσεων (1.9) ,είναι εφικτή μόνο για σχετικά απλά προβλήματα βελτιστοποίησης. Στη συνέχεια , θα ασχοληθούμε αποκλειστικά με μεθόδους **αριθμητικής επίλυσης** του προβλήματος(1.1).

## 1.2 Δομή Αλγορίθμων Βελτιστοποίησης

Για την επίλυση του προβλήματος ελαχιστοποίησης θα χρησιμοποιήσουμε ένα επαναληπτικό αλγόριθμο. Οι επαναληπτικοί αλγόριθμοι παράγουν μια ακολουθία σημείων $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \ldots$ ή συντομότερα $\{\mathbf{x}^{(k)}\}$ (ο άνω δείκτης υποδεικνύει τον αριθμό επανάληψης) που πρέπει να συγκλίνει σε ένα σταθερό σημείο $\mathbf{x}^*$ ανταποκρινόμενο στη λύση του προβλήματος. Τα $\mathbf{x}^{(k)}$ ονομάζονται **επαναληπτικά σημεία** ή απλώς **επαναληπτικά**. Επειδή ο ακριβής προσδιορισμός του $\mathbf{x}^*$ δεν είναι συνήθως εφικτός, ο αλγόριθμος διακόπτεται όταν ικανοποιείται ένα κριτήριο σύγκλισης. Το αντικειμενικό κριτήριο $f(\mathbf{x}^{(k)})$ βελτιώνεται σε κάθε επανάληψη.

Οι επαναληπτικοί αλγόριθμοι έχουν συνήθως την ακόλουθη δομή, εκκινώντας ένα δεδομένο από το χρήστη σημείο εκκίνησης $\mathbf{x}^{(1)}$:

α)Προσδιορισμός μιας κατεύθυνσης αναζήτησης $\mathbf{s}^{(k)}$.

β)Προσδιορισμός βέλτιστου βήματος $\alpha^{(k)}$ που ελαχιστοποιεί τη συνάρτηση $f(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)})$ ως προς α. (1.12)

γ)Υπολογισμός του νέου επαναληπτικού $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$.

δ)Αν ικανοποιείται το κριτήριο σύγκλισης ,ο αλγόριθμος σταματά με $\mathbf{x}^{(k)} \approx \mathbf{x}^*$.

Τα βήματα (α), (β), (γ), (δ) εκτελούνται σε κάθε επανάληψη του αλγόριθμου. Εναλλακτικές μέθοδοι διαφέρουν στον τρόπο καθορισμού της κατεύθυνσης αναζήτησης $\mathbf{s}^{(k)}$ στο βήμα (α) του αλγόριθμου. Στο βήμα (β) έχουμε το **υποπρόβλημα αναζήτησης επί γραμμής** που χρησιμοποιεί υπολογισμούς τιμών της συνάρτησης $f(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)})$ και της παραγώγου της $\mathrm{d}f(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)})/\mathrm{d}\alpha$ για αντίστοιχες τιμές α (βλ. υποκεφάλαιο 1.4).

## 1.3 Καθορισμός κατεύθυνσης αναζήτησης

Διάφορες μέθοδοι βελτιστοποίησης μπορούν να χρησιμοποιηθούν στο βήμα (α) του προηγούμενου αλγόριθμου για τον καθορισμό της κατεύθυνσης αναζήτησης $\mathbf{s}^{(k)}$. Όλες οι μέθοδοι χρησιμοποιούν μια κατεύθυνση κατάβασης που ικανοποιεί την

$$\left(\mathbf{s}^{(k)}\right)^{\mathrm{T}}\mathbf{g}^{(k)} < 0 \qquad (1.13)$$

ιδιότητα κατάβασης που εγγυάται ότι η κλίση $\mathrm{d}f/\mathrm{d}\alpha$ είναι πάντοτε αρνητική για α=0 (εκτός αν το $\mathbf{x}^{(\mathrm{k})}$ είναι στατικό σημείο) και άρα η αντικειμενική συνάρτηση είναι βελτιώσιμη για κάποιο $\alpha^{(\mathrm{k})} > 0$.

Μερικές από αυτές είναι οι μέθοδοι μέγιστης κατάβασης (steepest descent), συζυγών κατευθύνσεων (conjugate gradient), και σχεδόν-Newton (quasi-Newton).

### 1.3.1 Μέγιστη κατάβαση

Πρόκειται για την απλούστερη αλλά δυστυχώς τη λιγότερο αποτελεσματική μέθοδο καθορισμού της κατεύθυνσης αναζήτησης. Βασίζεται στην κατεύθυνση μέγιστης κατάβασης, δηλαδή παίρνουμε

$$\mathbf{s}^{(\mathrm{k})} = -\mathbf{g}^{(\mathrm{k})} \qquad (1.14)$$

Η μέθοδος μέγιστης κατάβασης τυπικά οδηγεί σε μία σύντομη προσέγγιση του ελάχιστου, αν το αρχικό $\mathbf{x}^{(1)}$ έχει επιλεγεί μακριά από το ελάχιστο. Πάντως, όσο αφορά την εγγύτητα με το ελάχιστο, η μέθοδος δεν θεωρείται πολύ αποδοτική.

### 1.3.2. Μέθοδοι συζυγών κλίσεων

Οι μέθοδοι συζυγών κατευθύνσεων πλησιάζουν την απλότητα της μεθόδου μέγιστης κατάβασης και την αποδοτικότητα των μεθόδων σχεδόν-Newton. Υπάρχουν δύο κύριες μέθοδοι συζυγών κατευθύνσεων, η μέθοδος **Fletcher-Reeves (FR)** και η μέθοδος **Polak-Ribiere (PR)**.

Η μέθοδος Fletcher-Reeves για παραγωγή συζυγών κατευθύνσεων έχει ως εξής:

$$\mathbf{s}^{(\mathrm{k}+1)} = -\mathbf{g}^{(\mathrm{k}+1)} + \beta^{(\mathrm{k})}\mathbf{s}^{(\mathrm{k})} \quad (1.15)$$

όπου $\beta^{(0)} = 0$ και για $\mathrm{k} \geq 1$

$$\beta^{(\mathrm{k})} = \left(\mathbf{g}^{(\mathrm{k}+1)}\right)^{\mathrm{T}}\mathbf{g}^{(\mathrm{k}+1)}/(\mathbf{g}^{(\mathrm{k})})^{\mathrm{T}}\mathbf{g}^{(\mathrm{k})} \quad (1.16)$$

Για k=0 ισχύει $\mathbf{s}^{(1)} = -\mathbf{g}^{(1)}$, δηλαδή η μέθοδος εκκινεί με την κατεύθυνση μέγιστης κατάβασης.

Για την μέθοδο Polak-Ribiere ισχύει

$$\beta^{(\mathrm{k})} = \left(\mathbf{g}^{(\mathrm{k}+1)} - \mathbf{g}^{(\mathrm{k})}\right)^{\mathrm{T}}\mathbf{g}^{(\mathrm{k}+1)}/(\mathbf{g}^{(\mathrm{k})})^{\mathrm{T}}\mathbf{g}^{(\mathrm{k})} \quad (1.17)$$

Η απλή μορφή της (1.15) δεν απαιτεί αριθμητικές πράξεις πινάκων για τον υπολογισμό της $\mathbf{s}^{(\mathrm{k})}$. Η μέθοδος Fletcher- Reeves απαιτεί μόνο την αποθήκευση τριών και η μέθοδος Polak-Ribiere μόνο τεσσάρων n-διανυσμάτων. Γι αυτό, αυτές οι μέθοδοι μπορούν να εφαρμοστούν χωρίς δυσκολία σε μεγάλα προβλήματα εκατοντάδων ή και χιλιάδων μεταβλητών.

## 1.3.3. Μέθοδοι Σχεδόν-Newton

Στην παρούσα εργασία θα ασχοληθούμε με δύο τύπους μεθόδων σχεδόν-Newton, τον **τύπο DFP** που προτάθηκε το 1963 από τους Davidon, Fletcher και Powell και τον **τύπο BFGS** που προτάθηκε το 1970 ανεξάρτητα από τους Broyden, Fletcher, Goldfard και Shanno.

Για τις μεθόδους σχεδόν-Newton έχουμε έτσι σε κάθε επανάληψη:

α) $s^{(k)} = -\mathbf{H}^{(k)}\mathbf{g}^{(k)}$

β) Αναζήτηση επί $\mathbf{s}^{(k)}$ δίδει $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$ (1.18)

γ) Μετατροπή του $\mathbf{H}^{(k)}$ σε $\mathbf{H}^{(k+1)}$.

Ο πίνακας εκκίνησης $\mathbf{H}^{(1)}$ μπορεί να είναι οιοσδήποτε θετικά ορισμένος πίνακας, συνήθως $\mathbf{H}^{(1)} = \mathbf{I}$.

Για τη μετατροπή του $\mathbf{H}^{(k)}$ σε $\mathbf{H}^{(k+1)}$ χρησιμοποιούμε την ακόλουθη **συνθήκη σχεδόν-Newton** (Παπαγεωργίου, 1998)

$$\mathbf{H}^{(k+1)}\mathbf{y}^{(k)} = \boldsymbol{\delta}^{(k)} \text{ (1.19)}$$

όπου

$$\boldsymbol{\delta}^{(k)} = \alpha^{(k)}\mathbf{s}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \text{ (1.20)}$$

$$\mathbf{y}^{(k)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} \text{ (1.21)}$$

Υπάρχουν διάφοροι τρόποι ικανοποίησης της συνθήκης (1.19). Ένας απλός τρόπος είναι να θεωρήσουμε

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + a\mathbf{u}\mathbf{u}^{T} + b\mathbf{v}\mathbf{v}^{T} \text{ (1.22)}$$

Οπότε με τη συνθήκη σχεδόν –Newton (1.19) έχουμε

$$\boldsymbol{\delta}^{(\kappa)} = \mathbf{H}^{(k)}\mathbf{y}^{(k)} + a\mathbf{u}\mathbf{u}^{T}\mathbf{y}^{(k)} + b\mathbf{v}\mathbf{v}^{T}\mathbf{y}^{(k)} \text{ (1.23)}$$

Η ισότητα όμως αυτή επιδέχεται πολλές λύσεις $\mathbf{u}, \mathbf{v}$, όπως $\mathbf{u} = \boldsymbol{\delta}^{(k)}$ και $\mathbf{v} = \mathbf{H}^{(k)}\mathbf{y}^{(k)}$ με $a = 1/(\mathbf{u}^{T}\mathbf{y})$ και $b = -1/(\mathbf{v}^{T}\mathbf{y})$ εξού ο τύπος

$$\mathbf{H}_{\mathrm{DFP}}^{(k+1)} = \mathbf{H} + \frac{\boldsymbol{\delta}\boldsymbol{\delta}^{T}}{\boldsymbol{\delta}^{T}\mathbf{y}} - \frac{\mathbf{H}\mathbf{y}\mathbf{y}^{T}\mathbf{H}}{\mathbf{y}^{T}\mathbf{H}\mathbf{y}} \text{ (2.24)}$$

που είναι γνωστός ως **τύπος DFP**. Η μέθοδος δουλεύει πολύ καλά σε πρακτικά προβλήματα και τυγχάνει ευρείας εφαρμογής. Είναι ασύγκριτα αποτελεσματικότερη από τη μέθοδο μέγιστης κατάβασης, αλλά και λίγο αποτελεσματικότερη από τις μεθόδους συζυγών κλίσεων του υποκεφαλαίου 1.3.2.

Οι ιδιότητες της DFP συνοψίζονται ως εξής:

- Τερματισμός το πολύ σε n επαναλήψεις με $\mathbf{H}^{(n+1)} = \mathbf{G}^{-1}$ για τετραγωνικές συναρτήσεις με ακριβή αναζήτηση επί γραμμής.
- Παράγει, για τετραγωνικές συναρτήσεις, συζυγείς κατευθύνσεις, και μάλιστα, αν $\mathbf{H}^{(1)} = \mathbf{I}$, συζυγείς κλίσεις.
- Διατηρεί τους πίνακες $\mathbf{H}^{(k)} > 0$ αν $\mathbf{H}^{(1)} > \mathbf{0}$ και άρα παράγει πάντα κατευθύνσεις κατάβασης.
- Χρειάζεται $3n^2 + O(n)$ πολλαπλασιασμούς ανά επανάληψη.
- Έχει υπεργραμμικό βαθμό σύγκλισης (βλ. υποκεφάλαιο 1.7).

Ο **τύπος BFGS** δίνεται από τη σχέση

$$\mathbf{H}_{\mathrm{BFGS}}^{(k+1)} = \mathbf{H} + \left(1 + \frac{\mathbf{y}^{\mathrm{T}}\mathbf{H}\mathbf{y}}{\boldsymbol{\delta}^{T}\boldsymbol{y}}\right)\frac{\boldsymbol{\delta}\boldsymbol{\delta}^{\mathrm{T}}}{\boldsymbol{\delta}^{\mathrm{T}}\mathbf{y}} - \frac{\boldsymbol{\delta}\mathbf{y}^{\mathbf{T}}\mathbf{H}+\mathbf{H}\mathbf{y}\boldsymbol{\delta}^{\mathbf{T}}}{\boldsymbol{\delta}^{\mathbf{T}}\mathbf{y}} \quad (1.25)$$

Ο τύπος BFGS δουλεύει πολύ καλά σε πρακτικές εφαρμογές, ίσως καλύτερα από τον DFP, και χαρακτηρίζεται από τις ίδιες ως άνω ιδιότητες του DFP.

## 1.4 Αλγόριθμος αναζήτησης επί γραμμής

Στο βήμα (β) του αλγορίθμου λύσης του προβλήματος έχουμε το υποπρόβλημα αναζήτησης επί γραμμής που χρησιμοποιεί υπολογισμούς τιμών $f(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)})$ για τις αντίστοιχες τιμές α. Το ελάχιστο γραμμής ικανοποιεί την $df/d\alpha = 0$.

Ένας αλγόριθμος αναζήτησης επί γραμμής υπολογίζει μια ακολουθία $\{\alpha_j\}$ βημάτων και τερματίζει όταν μια επανάληψη προσδιορίζει ένα αποδεκτό σημείο, που ικανοποιεί τις συνθήκες

$$f(\alpha) \le f(0) + \alpha\rho f'(0) \quad (1.35)$$

$$|f'(\alpha)| \le -\sigma f'(0) \quad (1.36)$$

όπου $\rho \in [0, 1/2]$ και $\rho \le \sigma < 1$ είναι προεπιλεγμένες παράμετροι και $f(\alpha) = f(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)})$ και βάση της σχέσης (1.7) ισχύει $f'(\alpha) = df(\alpha)/d\alpha = (\mathbf{g}^{(k)}(\mathbf{x}^{(k)} + \alpha\mathbf{s}^{(k)}), \mathbf{s}^{(k)})$, οπότε $f'(0) = (\mathbf{g}^{(k)}, \mathbf{s}^{(k)})$ και από την συνθήκη κατάβασης $(\mathbf{g}^{(k)}, \mathbf{s}^{(k)}) < 0$ έχουμε $f'(0) < 0$.

Η συνθήκη (1.35) δίνει το ανώτατο όριο στην αναζήτηση βήματος επί γραμμής και η (1.36) εγγυάται μη αμελητέα βελτίωση της αντικειμενικής συνάρτησης στην αντίστοιχη επανάληψη. Η ακρίβεια της αναζήτησης επί γραμμής είναι μεγαλύτερη αν το σ είναι κοντά στο μηδέν. Μία αρκετά ακριβής αναζήτηση επί γραμμής (σ κοντά στο μηδέν) απαιτεί συνήθως πολλές επαναλήψεις και μειώνει, έτσι την αποδοτικότητα του αλγορίθμου βελτιστοποίησης. Μία ανακριβής αναζήτηση επί γραμμής (σ κοντά στο ένα) μπορεί επίσης να μειώνει την αποδοτικότητα του αλγορίθμου οδηγώντας σε μεγαλύτερο αριθμό επαναλήψεων. Μία κατάλληλη επιλογή του σ εξαρτάται από το συγκεκριμένο πρόβλημα αλλά και από την εφαρμοσμένη μέθοδο για τον καθορισμό της κατεύθυνσης αναζήτησης. Για παράδειγμα, η μέθοδος

BFGS έχει βρεθεί εμπειρικά ότι δουλεύει αποδοτικότερα με ανακριβή αναζήτηση επί γραμμής.

Στον αλγόριθμο αναζήτησης επί γραμμής έχουμε μια **φάση εντοπισμού αγκύλης** που προσδιορίζει μια αγκύλη $[a_i, b_i]$ εμπεριέχουσα το υπό αναζήτηση ελάχιστο επί γραμμής. Η αγκύλη χαρακτηρίζεται από $f'(a_i) < 0$ και $f'(b_i) > 0$. Η φάση αυτή ακολουθείται από τη **φάση υποδιαίρεσης** κατά την οποία η αγκύλη υποδιαιρείται οδηγώντας σε μια ακολουθία αγκυλών $[a_j, b_j]$ με μήκος τείνον προς το μηδέν. Η υποδιαίρεση επιτυγχάνεται με κάποια μορφή παρεμβολής.

## 1.4.1 Φάση εντοπισμού αγκύλης

Η φάση εντοπισμού αγκύλης αρχίζει με $\alpha_0 = 0$ και με ένα αρχικό βήμα $\alpha_1$. Για παράδειγμα $\alpha_1 = -2(f^{(k+1)} - f^{(k)})/f'(0)$. Αυτή η επιλογή του $\alpha_1$ προκύπτει μέσω τετραγωνικής παρεμβολής. Οι επαναλήψεις i φάσης εντοπισμού αγκύλης καταλήγουν είτε με καθορισμό του ελάχιστου επί γραμμής (οπότε ο αλγόριθμος δεν προχωρά στην φάση υποδιαίρεσης), είτε με καθορισμό μίας αγκύλης που θα χρησιμοποιηθεί στη φάση υποδιαίρεσης, είτε με ένα νέο βήμα $\alpha_{i+1}$ που θα χρησιμοποιηθεί για την επόμενη επανάληψη της φάσης εντοπισμού αγκύλης. Έτσι σε κάθε επανάληψη της φάσης εντοπισμού αγκύλης, για την περίπτωση που είναι δυνατός ο υπολογισμός πρώτων παραγώγων της αντικειμενικής συνάρτησης, εκτελούνται τα εξής βήματα (Fletcher, 1987 και Papageorgiou and Marinaki, 1995):

**Βήμα 1** : Υπολογισμός του $f(\alpha_i)$.

**Βήμα 2** : Αν $f(\alpha_i) > f(0) + \alpha_i \rho f'(0)$ ή $f(\alpha_i) \geq f(\alpha_{i-1})$ τότε $a_i = \alpha_{i-1}, b_i = \alpha_i$ δίνει αγκύλη η οποία περιέχει το ελάχιστο επί γραμμής και η φάση εντοπισμού αγκύλης τερματίζει.

**Βήμα 3** : Υπολογισμός του $f'(\alpha_i)$.

**Βήμα 4** : Αν $|f'(\alpha_i)| \leq -\sigma f'(0)$,τελείωσε η αναζήτηση επί γραμμής.

**Βήμα 5** : Αν $f'(\alpha_i) \geq 0$ τότε $a_i = \alpha_{i-1}$, $b_i = \alpha_i$ δίνει αγκύλη η οποία περιέχει το ελάχιστο επί γραμμής και η φάση εντοπισμού αγκύλης τερματίζει. Αλλιώς θέτουμε $\alpha_{i+1} = \alpha_i + \tau_1(\alpha_i - \alpha_{i-1})$ όπου $\tau_1 \geq 1$. Για παράδειγμα , με $\tau_1 = 2$ το βήμα διπλασιάζεται σε κάθε επανάληψη της φάσης εντοπισμού αγκύλης.

## 1.4.2 Φάση υποδιαίρεσης της αγκύλης

Η φάση υποδιαίρεσης εκτελεί επίσης μια σειρά επαναλήψεων j όπου κάθε επανάληψη τελειώνει είτε με ένα βήμα $\alpha_j$ που ικανοποιεί τις συνθήκες (1.35), (1.36), είτε με μία νέα βραχύτερη αγκύλη $[a_j, \alpha_j]$ ή $[\alpha_j, b_j]$ για την επόμενη επανάληψη. Έτσι σε κάθε επανάληψη της φάσης υποδιαίρεσης εκτελούνται τα εξής βήματα:

**Βήμα 1** : Επιλογή $\alpha_j \in [a_j + \tau_2(b_j - a_j),\ b_j - \tau_3(b_j - a_j)]$ μέσω παρεμβολής (τετραγωνική ή κυβική παρεμβολή).

**Βήμα 2** : Υπολογισμός $f(\alpha_j)$.

**Βήμα 3** : Αν $f(\alpha_j) > f(0) + \rho\alpha_j f'(0)$ ή $f(\alpha) \geq f(a_j)$ τότε $a_{j+1} = a_j, b_{j+1} = \alpha_j$ είναι η βραχύτερη αγκύλη για την επόμενη επανάληψη.

**Βήμα 4** : Υπολογισμός $f'(\alpha_j)$.

**Βήμα 5** : Αν $|f'(\alpha_j)| \leq -\sigma f'(0)$ τελείωσε η αναζήτηση επί γραμμής.

**Βήμα 6** : Αν $|f'(\alpha_j)| \geq 0$ τότε $a_{j+1} = a_j, b_{j+1} = \alpha_j$ είναι η βραχύτερη αγκύλη για την επόμενη επανάληψη. Αλλιώς $a_{j+1} = \alpha_j, b_{j+1} = b_j$ είναι η βραχύτερη αγκύλη για την επόμενη επανάληψη.

Οι $\tau_2, \tau_3$ είναι προκαθορισμένες παράμετροι, $0 < \tau_2 < \tau_3 \leq 0.5$, που εγγυώνται ότι θα υπάρξει μη μηδαμινή μείωση της αγκύλης από επανάληψη και μάλιστα

$$|b_{j+1} - a_{j+1}| \leq (1 - \tau_2)|b_j - a_j| \quad (1.37)$$

Πράγμα που οδηγεί στη σύγκλιση του αλγόριθμου. Τυπικές τιμές είναι $\tau_2 = 0.1, \tau_3 = 0.5$. Το βήμα $\alpha_j$ που προσδιορίζεται με τη σύγκλιση είναι το ζητούμενο βήμα $\alpha^i$ της αντίστοιχης επανάληψης.

Η χρησιμοποιούμενη παρεμβολή στον αλγόριθμο μπορεί να είναι τετραγωνική ή κυβική όπως αναφέρθηκε. Το ερώτημα στην περίπτωση **τετραγωνικής παρεμβολής** είναι: Δεδομένης της αγκύλης $[a, b]$ και των τιμών $[f'(a), f'(b)]$ να ευρεθεί το ελάχιστο $\hat{\alpha}$ μιας τετραγωνικής συνάρτησης $\hat{f}(\alpha) = \delta\alpha^2 + \beta\alpha + \gamma$ που ανταποκρίνεται στις δεδομένες τιμές.

Ο αντίστοιχος τύπος είναι:

$$\hat{\alpha}[f'(a), f'(b)] = \frac{b f'(a) - a f'(b)}{f'(a) - f'(b)} \quad (1.36)$$

Ο τύπος κυβικής παρεμβολής δίνεται ως εξής

$$\hat{\alpha}[f(a), f(b), f'(a), f'(b)] = b - \frac{f'(b) + w - z}{f'(b) - f'(a) + 2w}(b - a) \quad (1.37)$$

Όπου

$$z = 3\frac{f(a) - f(b)}{b - a} + f'(a) + f'(b) \quad (1.38)$$

Και

$$w = \sqrt{z^2 - f'(a)f'(b)} \quad (1.39)$$

Αν οι παραπάνω τύποι παρεμβολής δώσουν ελάχιστο $\hat{\alpha}$ εκτός αγκύλης $[a, b]$ τότε επιλέγεται το αντίστοιχο οριακό σημείο της αγκύλης.

### 1.4.3. Χρυσή τομή

Σε περίπτωση που δεν είναι δυνατός ο υπολογισμός παραγώγων της $f$ στη φάση υποδιαίρεσης μπορούν να χρησιμοποιηθούν λιγότερο αποτελεσματικοί αλγόριθμοι που δεν κάνουν χρήση παραγώγων, όπως ο **αλγόριθμος χρυσής τομής**, ο οποίος μειώνει σε μια σειρά επαναλήψεων την αρχική αγκύλη $[a, b]$ κατά ένα σταθερό ποσοστό μέχρι τον εγκλωβισμό του εμπεριεχομένου ελάχιστου με ικανοποιητική ακρίβεια. Για την περιγραφή του αλγόριθμου χρυσής τομής θεωρούμε δύο εσωτερικά σημεία $\alpha_1, \alpha_2 \in [a, b]$ προσδιοριζόμενα ως εξής

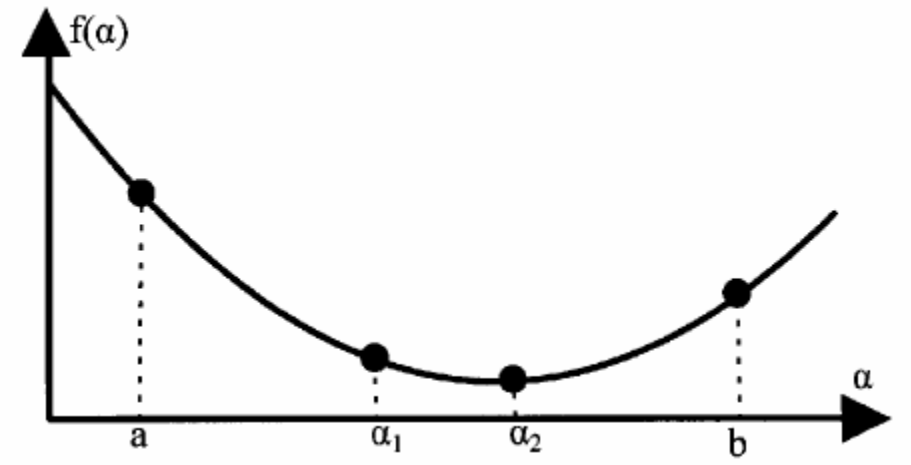
$$\alpha_1 = a + (1 - c)(b - a) \quad (1.40)$$

$$\alpha_2 = a + c(b - a) \quad (1.41)$$

Με τη σταθερά $0.5 < c < 1$. Επειδή $c > 0.5$, έχουμε $\alpha_1 < \alpha_2$, βλ. Σχ. 1.6. Μετά από υπολογισμό των $f(\alpha_1), f(\alpha_2)$ έχουμε: Αν $f(\alpha_1) < f(\alpha_2)$ τότε το ζητούμενο ελάχιστο βρίσκεται προφανώς στο διάστημα $[a, \alpha_2]$ ενώ αν $f(\alpha_1) > f(\alpha_2)$ τότε βρίσκεται στο $[\alpha_1, b]$. Οποιαδήποτε από τις δύο περιπτώσεις και αν ισχύει, έχει προσδιοριστεί μια βραχύτερη αγκύλη, με την οποία αρχίζει η επόμενη επανάληψη κ.ο.κ. έως κάποιο όριο εύρους αγκύλης ε. Η συστολή της αγκύλης σε κάθε επανάληψη δίνεται από τον παράγοντα συστολής c και ο απαιτούμενος αριθμός επαναλήψεων είναι

$$L = \text{entier}\left(\frac{\ln\frac{\varepsilon}{b-a}}{\ln c}\right) \quad (1.42)$$

Ο περιγραφείς αλγόριθμος απαιτεί δύο υπολογισμούς συνάρτησης ανά επανάληψη. Στην παρούσα εργασία θεωρούμε $c = (\sqrt{5} - 1)/2 \approx 0.618$ και έτσι έχουμε




**Σχήμα 2.6.** Υποδιαίρεση με χρυσή τομή .

$$\alpha_2 = \alpha_1 + (1 - c)(b - \alpha_1) \text{ (1.43)}$$

$$\alpha_1 = a + c(\alpha_2 - a) \text{ (1.44)}$$

Δηλαδή το σημείο $\alpha_2$ παίζει το ρόλο του εσωτερικού σημείου $\alpha_1$(1.40) για την αγκύλη $[\alpha_1, b]$ ενώ το σημείο $\alpha_1$ παίζει το ρόλο του σημείου $\alpha_2$ (1.41) για την αγκύλη $[a, \alpha_2]$ πράγμα που μειώνει τον αριθμό υπολογισμών συνάρτησης ανά επανάληψη σε έναν.

## 1.5 Επανέναρξη

Ο αλγόριθμος μπορεί να συναντήσει δυσκολίες κατά τις επαναλήψεις λόγω των μη τετραγωνικών συναρτήσεων κόστους όταν εφαρμόζονται οι μέθοδοι σχεδόν-Newton και συζυγών κλίσεων ή λόγω ανακριβούς αναζήτησης επί γραμμής. Για αυτό το λόγο η ακόλουθη **συνθήκη ικανής αρνητικότητας** πρέπει να ελέγχεται σε κάθε επανάληψη

$$\left(\mathbf{s}^{(k)}, \mathbf{g}^{(k)}\right) \leq -B[\left(\mathbf{s}^{(k)}, \mathbf{s}^{(k)}\right)\left(\mathbf{g}^{(k)}, \mathbf{g}^{(k)}\right)]^{1/2} \text{ (1.45)}$$

(όπου B είναι μια σταθερή θετική παράμετρος). Αν η (1.45) παραβιάζεται, γίνεται **επανέναρξη** (restart).

Επανέναρξη σημαίνει ότι η τρέχουσα επανάληψη διακόπτεται , και αρχίζει μία καινούρια επανάληψη με την κατεύθυνση μέγιστης κατάβασης, δηλαδή $\mathbf{s}^{(k)} = -\mathbf{g}^{(k)}$. Οι επόμενες επαναλήψεις συνεχίζονται κανονικά ανάλογα με την επιλεγμένη κατεύθυνση αναζήτησης.

Ανεξάρτητα από την (1.45), περιοδική επανέναρξη κάθε N επαναλήψεις βελτιώνει την αποτελεσματικότητα του αλγορίθμου. Στην περίπτωση των μεθόδων σχεδόν-Newton, η περιοδική επανέναρξη περιορίζει τον απαιτούμενο υπολογιστικό χώρο και τον υπολογιστικό χρόνο ανά επανάληψη.

## 1.6 Κανονικοποίηση

Η κανονικοποίηση μπορεί να χρησιμοποιηθεί για να βελτιώσει την κατάσταση ενός προβλήματος βελτιστοποίησης. Η κανονικοποίηση μπορεί να πραγματοποιηθεί υπολογίζοντας αμέσως μετά από κάθε επανέναρξη τον πίνακα κανονικοποίησης A από τη σχέση

$$\mathbf{A} = \text{diag}[\sqrt{\frac{\partial^2 f}{\partial x_i^2}}] \text{ (1.46)}$$

και χρησιμοποιώντας την κανονικοποιημένη μορφή των μεθόδων αναζήτησης κατεύθυνσης.

Η κανονικοποιημένη μορφή της μέγιστης κατάβασης δίνεται από

$$\mathbf{s}^{(k)} = -\mathbf{A}^{-2}\mathbf{g}^{(k)} \text{ (1.47)}$$

Η κανονικοποιημένη μορφή των μεθόδων συζυγών κλίσεων είναι

$$\mathbf{s}^{(1)} = -\mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k})} \text{ (1.48)}$$

$$\mathbf{s}^{(\mathrm{k})} = -\mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k})} + \beta^{(\mathrm{k})}\mathbf{s}^{(\mathrm{k}-1)}, \mathrm{k} > 1 \text{ (1.49)}$$

$$\beta^{(\mathrm{k})} = \frac{(\mathbf{g}^{(\mathrm{k})}, \mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k})})}{(\mathbf{g}^{(\mathrm{k}-1)}, \mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k}-1)})} \text{ (1.50)}$$

Για τη μέθοδο Fletcher-Reeves, και

$$\beta^{(\mathrm{k})} = \frac{(\mathbf{g}^{(\mathrm{k})} - \mathbf{g}^{(\mathrm{k}-1)}, \mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k})})}{(\mathbf{g}^{(\mathrm{k}-1)}, \mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k}-1)})} \text{(1.51)}$$

για τη μέθοδο Polak-Ribiere.

Για τις κλιμακοποιημένες σχεδόν-Newton μεθόδους για τον πρώτο τρόπο όλοι οι τύποι παραμένουν αμετάβλητοι και απαιτείται απλώς στην πρώτη επανάληψη η χρήση $\mathbf{H}^{(\mathrm{k})} = \mathbf{A}^{-2}$ αντί $\mathbf{H}^{(\mathrm{k})} = \mathbf{I}$, ενώ στο δεύτερο τρόπο οι αλλαγές είναι οι ακόλουθες

- Για $\mathrm{k} = 1$,η (1.48) αντικαθιστά την (1.26) ή την (1.31).
- Ο πρώτος όρος της (1.27) και της (1.32), ο $\mathbf{y}^{(\mathrm{k}-1)}$, αντικαθίσταται από $\mathbf{A}^{-2}\mathbf{y}^{(\mathrm{k}-1)}$.
- Ο πρώτος όρος της (1.30) και της (1.34), ο $-\mathbf{g}^{(\mathrm{k})}$, αντικαθίσταται από $-\mathbf{A}^{-2}\mathbf{g}^{(\mathrm{k})}$.

## 1.7 Συνθήκη σύγκλισης

Ο επαναληπτικός αλγόριθμος διακόπτεται όταν ικανοποιείται ένα κριτήριο σύγκλισης, όπως αναφέρεται στο υποκεφάλαιο 1.2. Αλγόριθμοι που συγκλίνουν στο $\mathbf{x}^*$ από οποιοδήποτε σημείο εκκίνησης $\mathbf{x}^{(1)}$ έχουν την ιδιότητα **καθολικής σύγκλισης**. Ιδιότητες **τοπικής σύγκλισης** αναφέρονται στο ρυθμό σύγκλισης του λάθους

$$\mathbf{h}^{(\mathrm{k})} = \mathbf{x}^{(\mathrm{k})} - \mathbf{x}^* \text{ (1.52)}$$

προς το **0** σε μια περιοχή γύρω από το $\mathbf{x}^*$.Αν

$$\left\|\mathbf{h}^{(\mathrm{k}+1)}\right\| / \left\|\mathbf{h}^{(\mathrm{k})}\right\|^{\mathrm{P}} \to \mathrm{a}, \mathrm{a} > 0 \text{ (1.53)}$$

τότε έχουμε **βαθμό σύγκλισης** $\mathrm{p}$.Συνήθως έχουμε $\mathrm{p} = 1$ (**σύγκλιση πρώτου βαθμού** ή **γραμμική σύγκλιση**) ή $\mathrm{p} = 2$ (**σύγκλιση δευτέρου βαθμού** ή **τετραγωνική σύγκλιση**). Γραμμική σύγκλιση είναι ικανοποιητική αν $\mathrm{a} \leq 1/4$. Αν όμως $\mathrm{a} \to 0$ έχουμε **υπεργραμμική σύγκλιση**, όπως για παράδειγμα ισχύει για την μέθοδο DFP. Η πρακτική σημασία των ιδιοτήτων τοπικής σύγκλισης περιορίζεται από το γεγονός ότι ισχύουν σε μια περισσότερο ή λιγότερο μικρή περιοχή περί το ζητούμενο ελάχιστο $\mathbf{x}^*$. Για το λόγο αυτό ,πέραν των θεωρητικών αποτελεσμάτων ρυθμού σύγκλισης κάθε μεθόδου, ιδιαίτερα σημαντική ή και καθοριστική είναι η αποτελεσματικότητά της σε μια σειρά δοκιμαστικών προβλημάτων που συνήθως χρησιμοποιούνται για την πειραματική σύγκριση εναλλακτικών μεθόδων.

Το κριτήριο σύγκλισης του επαναληπτικού αλγορίθμου (βήμα δ του αλγορίθμου του υποκεφαλαίου 1.2) μπορεί να έχει διάφορες μορφές (Παπαγεωργίου ,1998). Στην παρούσα εργασία οι επαναλήψεις σταματούν αν η ακόλουθη συνθήκη σύγκλισης ικανοποιηθεί

$$(\mathbf{g}^{(k)}, \mathbf{g}^{(k)}) < \varepsilon \quad (1.54)$$

Όπου $\varepsilon > 0$ δείχνει την ακρίβεια της αριθμητικής λύσης.

## 1.8 Αλγόριθμος RPROP

Για την λύση των προβλήματος (1.1) χρησιμοποιούμε και τον αλγόριθμο RPROP (Riedmiller and Braun, 1993), ο οποίος θα συγκριθεί με όλους τους παραπάνω αλγορίθμους. Ο αλγόριθμος RPROP είναι μία προσαρμοστική τεχνική, η οποία δεν στηρίζεται στην τιμή της κλίσης $\mathbf{g}$ αλλά εξαρτάται μόνο από το πρόσημο της τιμής της. Ο αλγόριθμος RPROP δεν προσδιορίζει κατεύθυνση αναζήτησης ούτε λύνει το υποπρόβλημα αναζήτησης επί γραμμής (βήματα α και β αντίστοιχα, του αλγορίθμου του υποκεφαλαίου 1.2) αλλά αλλάζει άμεσα το μέγεθος της διόρθωσης $\mathbf{d}^{(k)}$ η οποία χρησιμοποιείται για τον υπολογισμό του νέου επαναληπτικού σημείου

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)} \quad (1.55)$$

Για την επίτευξη της άμεσης αλλαγής της διόρθωσης, ορίζεται για κάθε συνιστώσα $x_i$ μια τιμή ενημέρωσης $c_i$, η οποία μεμονωμένα καθορίζει το μέγεθος της αντίστοιχης διόρθωσης $d_i$.

Πιο συγκριμένα, ο επαναληπτικός αλγόριθμος έχει την ακόλουθη δομή, εκκινώντας από ένα δεδομένο από τον χρήστη σημείο εκκίνησης $\mathbf{x}^{(1)}$ και την τιμή ενημέρωσης $\mathbf{c}^{(1)}$:

1. Προσδιορισμός της τιμής ενημέρωσης $\mathbf{c}^{(k)}$.
2. Προσδιορισμός της διόρθωσης $\mathbf{d}^{(k)}$**.**
3. Υπολογισμός του νέου επαναληπτικού $\mathbf{x}^{(k+1)}$ βάση της(1.55).
4. Αν ικανοποιείται το κριτήριο σύγκλισης, ο αλγόριθμος σταματά με $\mathbf{x}^{(k)} \approx \mathbf{x}^{*}$.

Τα βήματα 1, 2, 3, και 4. εκτελούνται σε κάθε επανάληψη του αλγορίθμου.

Στο βήμα 1 ο προσδιορισμός της τιμής ενημέρωσης γίνεται ως εξής

$$c_i^{(k)} = \begin{cases} \min\left\{n^+ c_i^{(k-1)}, c_{max}\right\} \ \alpha\nu \left(g_i^{(k-1)} g_i^{(k)}\right) > 0 \\ \max\left\{n^- c_i^{(k-1)}, c_{min}\right\} \ \alpha\nu \left(g_i^{(k-1)} g_i^{(k)}\right) < 0 \\ c_i^{(k-1)} \ \alpha\nu \left(g_i^{(k-1)} g_i^{(k)}\right) = 0 \end{cases} \quad (1.56)$$

όπου $0 < n^- < 1 < n^+$ και τυπικές τιμές τους $n^- = 0.5$ και $n^+ = 1.2$. Κάθε φορά που το πρόσημο μιας συνιστώσας της κλίσης αλλάζει, η τιμή μειώνεται κατά τον παράγοντα $n^-$. Αυτό γίνεται διότι η τελευταία τιμή ενημέρωσης που

υπολογίστηκε ήταν μεγάλη και ο αλγόριθμος υπερπήδησε ένα τοπικό ελάχιστο. Αν η τιμή της κλίσης διατηρεί το πρόσημό της, η τιμή ενημέρωσης αυξάνει κατά τον παράγοντα $n^+$ για να επιταχυνθεί η σύγκλιση.

Για την αποφυγή της υπερχείλισης όταν ο αλγόριθμος εφαρμοστεί σε έναν ηλεκτρονικό υπολογιστή χρησιμοποιούνται τα πάνω και κάτω όρια $(c_{max}, c_{min})$.

Στο βήμα 2. ο προσδιορισμός της διόρθωσης γίνεται ως εξής

$$d_i^{(k)} = \begin{cases} -c_i^{(k)}, \alpha\nu\ g_i^{(k)} > 0 \\ +c_i^{(k)}, \alpha\nu\ g_i^{(k)} < 0 \\ 0, \alpha\nu\ g_i^{(k)} = 0 \end{cases} \quad (1.57)$$

Όπως φαίνεται από την σχέση 1.57 όταν η κλίση είναι θετική τότε η διόρθωση θα πρέπει να αφαιρεθεί από την τιμή της μεταβλητής, ενώ όταν η κλίση είναι αρνητική η διόρθωση θα πρέπει να προστίθεται στην τιμή της μεταβλητής. Αν η κλίση είναι μηδέν τότε δεν υπάρχει διόρθωση.

# Κεφάλαιο 2: Υπολογιστικός Κώδικας

## 2.1 Γενικά

Όπως έγινε κατανοητό στο προηγούμενο κεφάλαιο οι αλγόριθμοι επίλυσης είναι επαναληπτικοί και επιλύουν προβλήματα με πολλές μεταβλητές. Απαιτείται λοιπόν η χρήση Η/Υ για τη γρήγορη και σωστή επίλυση των προβλημάτων. Με βάση λοιπόν το θεωρητικό κομμάτι που αναπτύχτηκε στο κεφάλαιο 1, δημιουργούμε έναν υπολογιστικό κώδικα για την επίλυση μη γραμμικών προβλημάτων και τη σύγκριση των μεθόδων επίλυσης τους. Για τη σύγκριση τους θα παίρνουμε σαν έξοδο από το πρόγραμμα την τιμή της αντικειμενικής συνάρτησης σε κάθε επανάληψη και το χρόνο εκτέλεσης κάθε επανάληψης.

Ο υπολογιστικός κώδικας δημιουργήθηκε από τον Ιωάννη Μαρινάκη το 2000 στα πλαίσια της διπλωματικής του εργασίας « Διερεύνηση Ιδιοτήτων Αλγορίθμων Μη-γραμμικής Βελτιστοποίησης». Επαναπρογραμματίζοντας και εξελίσσοντας αυτόν τον κώδικα, έγινε δυνατή η λύση περισσοτέρων προβλημάτων με διάφορες διαστάσεις. Ο προγραμματισμός έγινε στο Visual studio της Microsoft σε λειτουργικό Windows XP.

## 2.2 Εισαγωγή και εκτύπωση δεδομένων

Η εισαγωγή δεδομένων όπως η επιλογή συνάρτησης, η επιλογή μεθόδου, μεγέθους του προβλήματος κ.α., γίνεται μέσω αρχείου. Έτσι επιτυγχάνεται μεγαλύτερη ευχέρεια στο χρήστη για την αλλαγή των δεδομένων.

Η εκτύπωση των αποτελεσμάτων γίνεται στην οθόνη κατά τη διάρκεια της ελαχιστοποίησης, ώστε να παρατηρούνται τυχόν ανωμαλίες, αλλά και σε αρχεία για περαιτέρω επεξεργασία.

Στο παράρτημα αναφέρεται αναλυτικό εγχειρίδιο για τη χρήση του προγράμματος, τον ακριβή τρόπο εισαγωγής δεδομένων καθώς και της εκτύπωσής τους. Επίσης υπάρχει και ο κώδικας γραμμένος σε γλώσσα προγραμματισμού C.

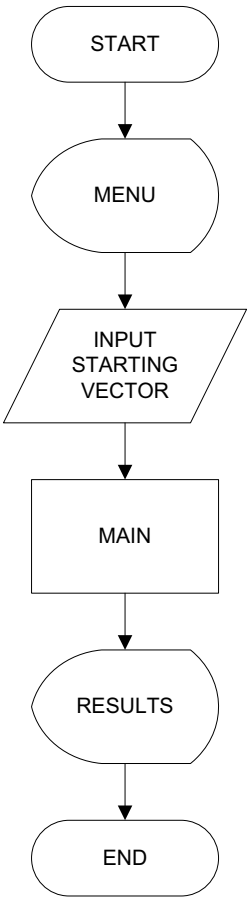
## 2.3 Διαγράμματα ροής υπορουτινών

Ο σχεδιασμός και η ανάπτυξη του υπολογιστικού κώδικα στηρίχτηκε στις αρχές του δομημένου προγραμματισμού, δηλαδή στη χρήση αυτόνομων κομματιών και υπορουτινών και αποτελείται από τέσσερεις κύριες υπορουτίνες. Κάθε κύρια υπορουτίνα εκτελεί μια αυτόνομη λειτουργία , ενώ όλες μαζί συνεργάζονται για το κτίσιμο του κυρίως προγράμματος.

Στη συνέχεια , δίνονται αναλυτικά τα διαγράμματα ροής του προγράμματος. Κάθε ένα από τα διαγράμματα συνοδεύεται από ανάλυση των στοιχείων του.

### 2.3.1 Διάγραμμα ροής του προγράμματος

Το πρόγραμμα ξεκινάει εμφανίζοντας το κυρίως menu επιλογών (Σχ. 2.1). Το menou αυτό περιλαμβάνει την επιλογή της μεθόδου βελτιστοποίησης, καθώς και ένα πλήθος επιλογών που σχετίζονται με εσωτερικές διαδικασίες.




**Σχήμα 2.1:** Διάγραμμα ροής προγράμματος

Αφού ο χρήστης έχει επιλέξει τις μεθόδους και λειτουργίες που θέλει, γίνεται η εισαγωγή του σημείου εκκίνησης από την κατάλληλη υποροutίνα. Στη συνέχεια περνάμε στην εκτέλεση του κύριου μέρους του προγράμματος (main), ενώ στο τέλος εμφανίζονται τα αποτελέσματα στον χρήστη. Ο χρήστης έχει τη δυνατότητα να λάβει τα αποτελέσματα της εκτέλεσης είτε στην οθόνη είτε σε αρχείο.

### 2.3.2 Διάγραμμα ροής του κυρίου σώματος (main) του προγράμματος

Το κύριο σώμα του προγράμματος αποτελεί το βασικό κομμάτι του αλγορίθμου αφού μέσω αυτού συνδέονται οι τέσσερεις κύριες υποροutίνες του προγράμματος δηλαδή οι υποροutίνες που αφορούν την επιλογή κατεύθυνσης αναζήτησης, τη φάση εντοπισμού αγκύλης, τη φάση υποδιαίρεσης, και τον επαναπροσδιορισμό των μεταβλητών.

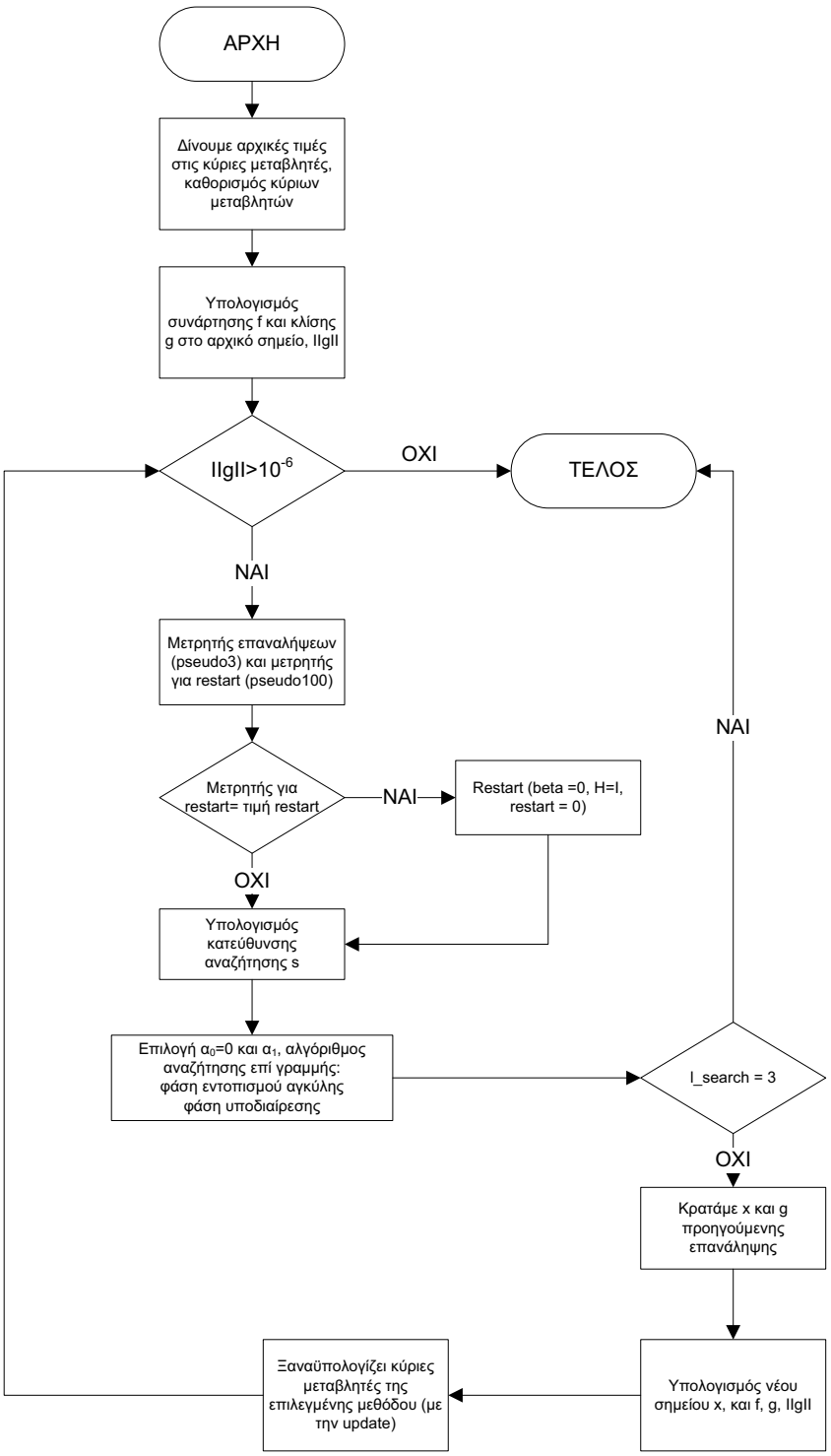
Το βασικό κομμάτι του κυρίου σώματος αποτελείται από έναν επαναλαμβανόμενο βρόγχο (Σχήμα 3.2). Το κριτήριο λήξης της επαναλη-

πτικής διαδικασίας είναι το εσωτερικό γινόμενο του διανύσματος κλίσης (g_norm) να γίνει μικρότερο από $10^{-6}$ (στις περισσότερες περιπτώσεις).

Μέσα στον κύριο βρόγχο εκτελούνται σειριακά οι υπορουτίνες:

- Επιλογή της κατεύθυνσης αναζήτησης.
- Αναζήτηση επί γραμμής που περιλαμβάνει τις φάσεις εντοπισμού αγκύλης και υποδιαίρεσης της αγκύλης.
- Επαναπροσδιορισμός των μεταβλητών.

| **Σύμβολα** | **Περιγραφή** |
|---|---|
| f | Η τιμή της συνάρτησης για δεδομένο x |
| g | Η τιμή του διανύσματος κλίσης για δεδομένο x |
| ||g|| | Το εσωτερικό γινόμενο του διανύσματος κλίσης |
| Restart | Συχνότητα εκτέλεσης της διαδικασίας επανέναρξης |
| $\alpha_0$ | Το αριστερό άκρο της πρώτης αγκύλης |
| $\alpha_1$ | Το δεξί άκρο της πρώτης αγκύλης |
| l_search | Βοηθητική μεταβλητή η οποία αν γίνει 0 μπαίνουμε στην ρουτίνα εντοπισμού αγκύλης, αν γίνει 1 μπαίνουμε στην ρουτίνα υποδιαίρεσης, αν είναι 2 τελειώνει η αναζήτηση επί γραμμής και αν είναι 3 τελειώνει το κύριο σώμα του προγράμματος |
| pseudo3 | Μετρητής για τον αριθμό των επαναλήψεων |
| pseudo100 | Μετρητής για τη συνθήκη επανέναρξης |

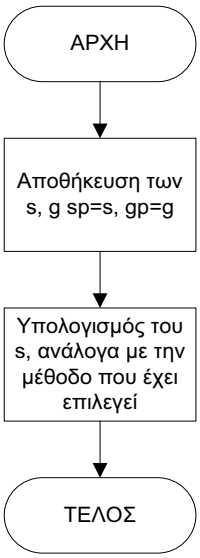



**Σχήμα 2.2:** Διάγραμμα ροής κυρίου σώματος (main) του προγράμματος

### 2.3.3 Διάγραμμα ροής κατεύθυνσης αναζήτησης

Στην υπορουτίνα αυτή γίνεται η επιλογή της κατεύθυνσης αναζήτησης (void searching_direction()).

Ανάλογα με την προεπιλεγμένη μέθοδο βελτιστοποίησης, επιλέγεται η αντίστοιχη κατεύθυνση αναζήτησης (Σχήμα 2.3 ).



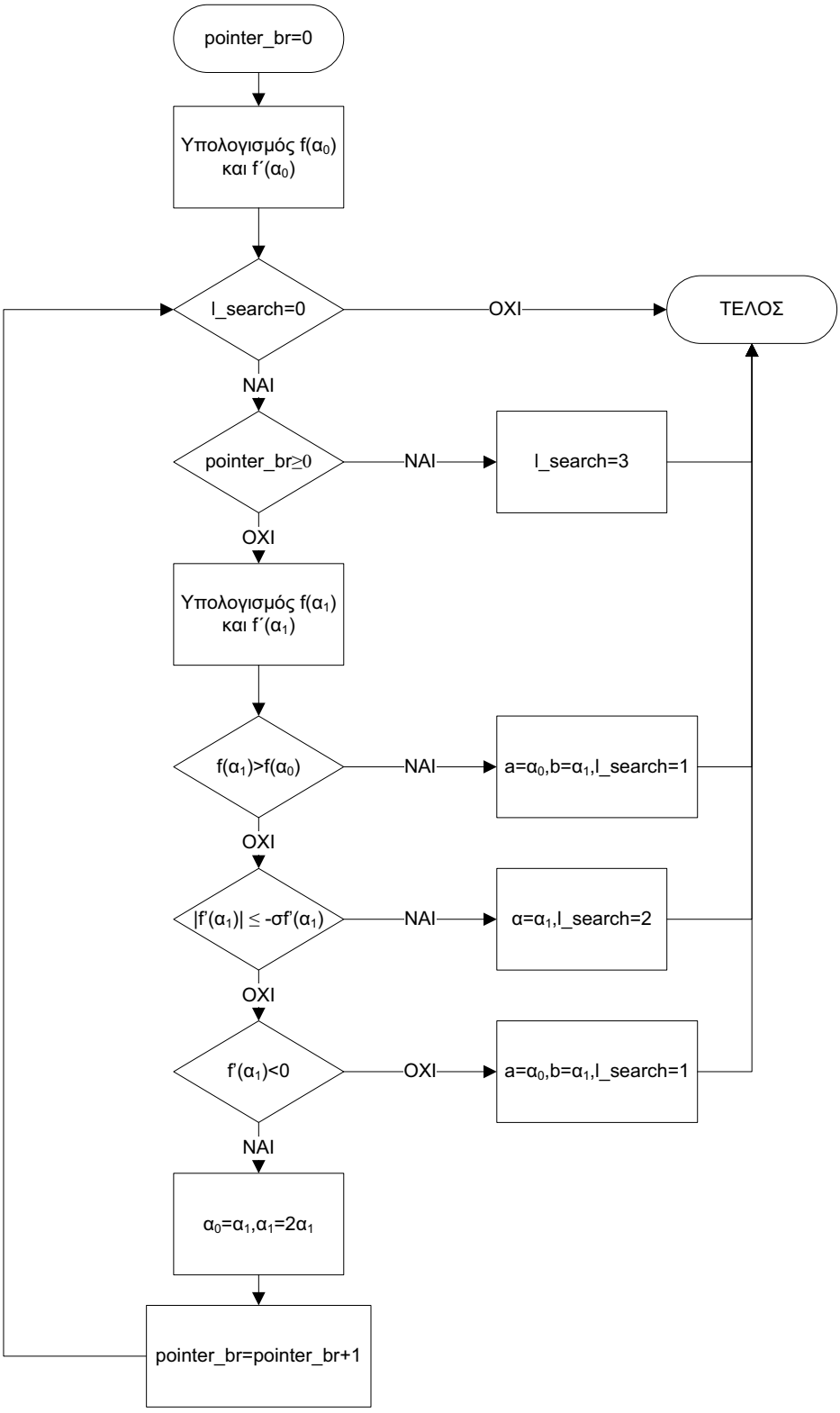

**Σχήμα 2.3:** Διάγραμμα ροής της επιλογής της κατεύθυνσης αναζήτησης.

## 2.3.4 Διάγραμμα ροής της υπορουτίνας εντοπισμού της αγκύλης

Η υπορουτίνα αυτή αποτελεί το πρώτο από τα δύο κομμάτια της αναζήτησης επί γραμμής και οδηγεί στον εντοπισμό της αγκύλης μέσα στην οποία περιέχεται το ελάχιστο επί γραμμής (Σχ.3.4).

Το κύριο σώμα της υπορουτίνας περικλείεται μέσα σε ένα επαναληπτικό βρόγχο με κριτήριο τερματισμού η τιμή του δείκτη l_search να γίνει διαφορετικός από το μηδέν. Η υπορουτίνα δέχεται ως εισαγωγικές τιμές τα αρχικά της αγκύλης [$\alpha_0$ $\alpha_1$] και ακολουθεί τα βήματα του αλγορίθμου του υποκεφαλαίου 2.4.1.

| **Σύμβολα** | **Περιγραφή** |
|---|---|
| $\alpha_0$ | Το αριστερό άκρο της αγκύλης |
| $\alpha_1$ | Αρχικό βήμα |
| $f(\alpha_0)$ | Η τιμή της συνάρτησης για δεδομένο $\mathbf{x}=\mathbf{x}+\alpha_0*\mathbf{s}$ |
| $f(\alpha_1)$ | Η τιμή της συνάρτησης για δεδομένο $\mathbf{x}=\mathbf{x}+\alpha_1*\mathbf{s}$ |
| $f'(\alpha_0)$ | Η τιμή της πρώτης παραγώγου για $\mathbf{x}=\mathbf{x}+\alpha_0*\mathbf{s}$ |
| $f'(\alpha_1)$ | Η τιμή της πρώτης παραγώγου για $\mathbf{x}=\mathbf{x}+\alpha_1*\mathbf{s}$ |
| f'(0) | Η τιμή της πρώτης παραγώγου για α=0 |
| pointer_br | Βοηθητική μεταβλητή η οποία όταν είναι μικρότερη από το 0 έχουμε εντοπίσει αγκύλη και αρχίζει η διαδικασία εντοπισμού |




**Σχήμα 2.4:** Διάγραμμα ροής της υπορουτίνας εντοπισμού της αγκύλης

## 2.3.5 Διάγραμμα ροής της υπορουτίνας υποδιαίρεσης

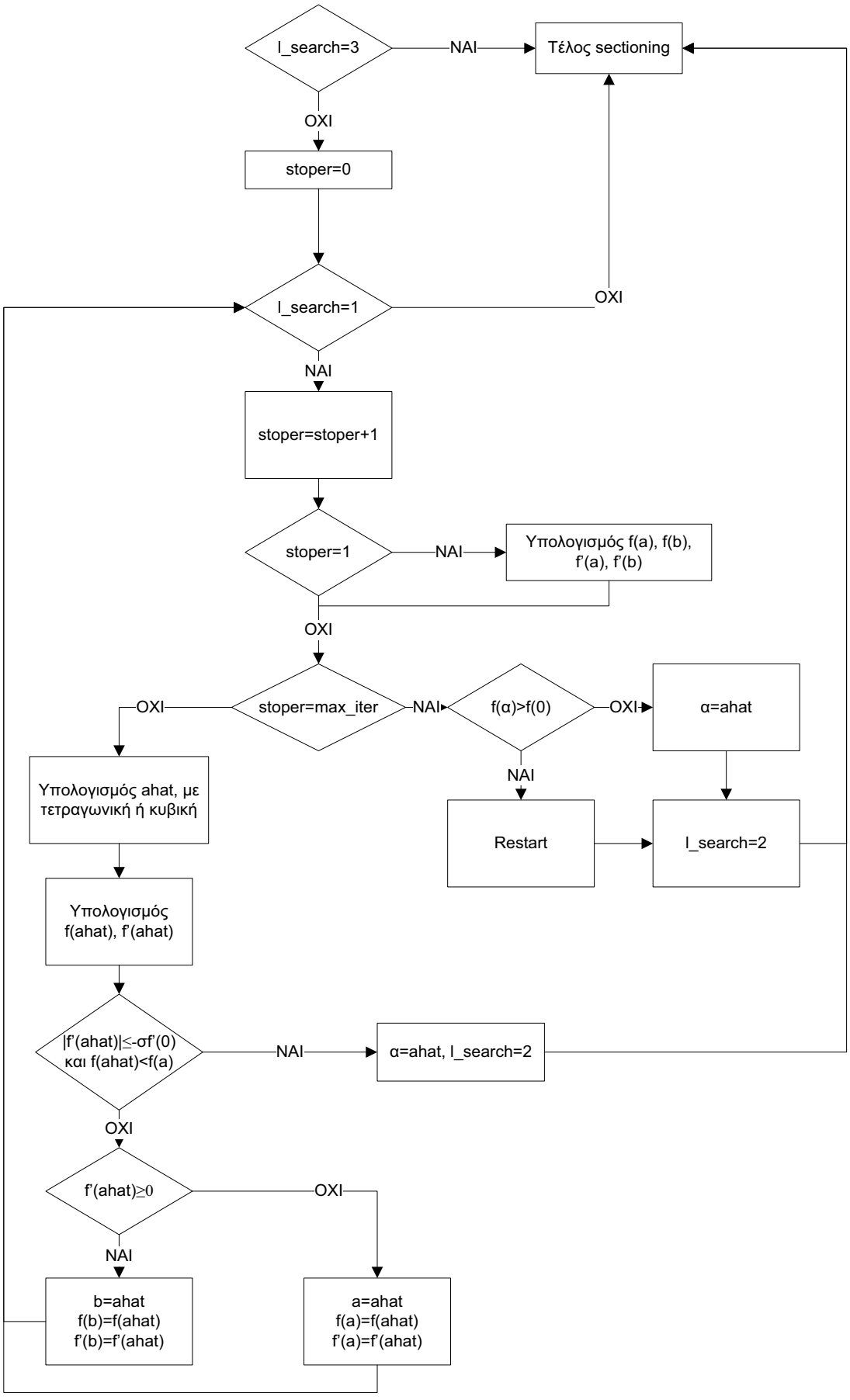
Η υπορουτίνα αυτή αποτελεί το δεύτερο μέρος της διαδικασίας αναζήτησης επί γραμμής και οδηγεί στην εύρεση του βέλτιστου σημείου επί της γραμμής (Σχ.2.5). Η δομή της είναι παρόμοια με αυτή της υπορουτίνας

καθορισμού αγκύλης. Αποτελείται από ένα επαναλαμβανόμενο βρόγχο με κριτήριο τερματισμού ο δείκτης l_search να γίνει διαφορετικός του 1.

Ως είσοδο παίρνει την αγκύλη που εμπεριέχει το ελάχιστο επί γραμμής και για την εύρεση του βέλτιστου σημείου επί γραμμής ακολουθεί τα βήματα του αλγόριθμου του υποκεφαλαίου 1.4.2.

Η φάση υποδιαίρεσης της αγκύλης και εντοπισμού του βέλτιστου είναι δυνατό σε ορισμένες περιπτώσεις να οδηγήσει σε δυσκολίες και υψηλό αριθμό επαναλήψεων. Για αυτό έχει τοποθετηθεί στην αρχή της υπορουτίνας ένας μετρητής, ο οποίος μας επιτρέπει να τερματίζεται η φάση της υποδιαίρεσης μετά από ένα προεπιλεγμένο αριθμό επαναλήψεων.

| **Σύμβολα** | **Περιγραφή** |
|---|---|
| f(a) | Η τιμή της συνάρτησης για δεδομένο **x**=**x**+a*s |
| f(b) | Η τιμή της συνάρτησης για δεδομένο **x**=**x**+b*s |
| f'(a) | Η τιμή της πρώτης παραγώγου για **x**=**x**+a*s |
| f'(b) | Η τιμή της πρώτης παραγώγου για **x**=**x**+b*s |
| ahat | Το σημείο που προκύπτει από τετραγωνική ή κυβική παρεμβολή |
| f(ahat) | Η τιμή της συνάρτησης για το σημείο που προκύπτει από την παρεμβολή |
| f'(ahat) | Η τιμή της πρώτης παραγώγου για το σημείο που προκύπτει από την παρεμβολή |
| stoper | Μετρητής των επαναλήψεων στη φάση υποδιαίρεσης |

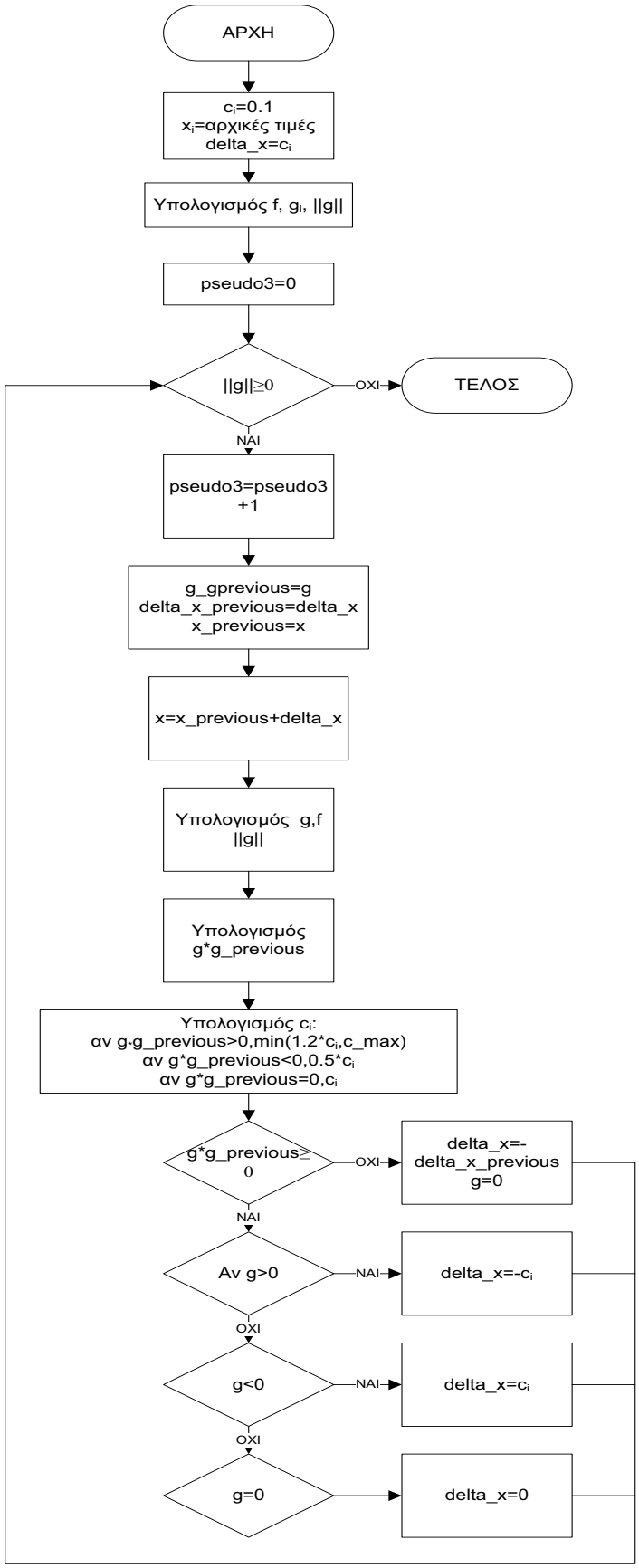



**Σχήμα 2.5:** Διάγραμμα ροής υπορουτίνας υποδιαίρεσης της αγκύλης

## 2.3.6 Διάγραμμα ροής του αλγόριθμου RPROP

Ο αλγόριθμος RPROP έχει προγραμματιστεί ακολουθώντας τα βήματα που δίνονται στο υποκεφάλαιο 1.8. Ο RPROP είναι ένας ξεχωριστός κώδικας. Αυτά τα τέσσερα βήματα φαίνονται αναλυτικά στο διάγραμμα ροής του Σχήματος 2.6.

| **Σύμβολα** | **Περιγραφή** |
|---|---|
| C | Τιμή ενημέρωσης |
| delta_x | Τιμή διόρθωσης |
| c_max | Άνω όριο των τιμών ενημέρωσης |
| F | Η τιμή της συνάρτησης για δεδομένο σημείο x |




**Σχήμα 2.6:** Διάγραμμα αλγορίθμου RPROP

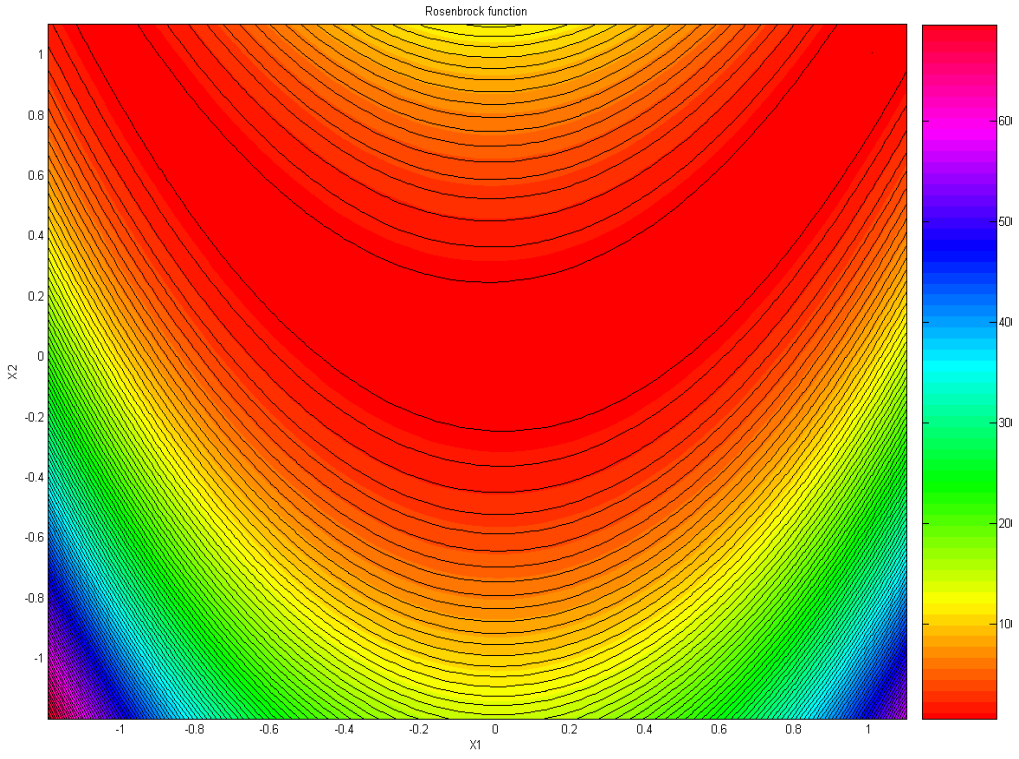
# Κεφάλαιο 3: Παρουσίαση Συναρτήσεων

Στο κεφάλαιο αυτό θα παρουσιάσουμε τις συναρτήσεις που επιλύθηκαν. Οι συναρτήσεις παρουσιάζονται με την ονομασία τους, τους τύπους τους, τα αρχικά σημεία που προτείνονται από τη βιβλιογραφία και ένα διάγραμμα για n=2. Όμως μερικές φορές είναι αποπροσανατολιστικό, όσον αφορά την αναπαράσταση της συνάρτησης, επειδή κάποιες συναρτήσεις για n>2 εμφανίζουν νέες μεταβλητές στον τύπο τους (βλ. π.χ. Broyden tridiagonal), οπότε και αλλάζει όλη η μορφή της συνάρτησης. Στο παράρτημα που ακολουθεί ο κώδικας προγραμματισμού τους είναι σε γλώσσα C. Για την δημιουργία των διαγραμμάτων για δύο μεταβλητές χρησιμοποιήθηκε το λογισμικό Matlab.

## 3.1 Rosenbrock function

$$f(x) = \sum_{i=2}^{n} [100(x_{i-1}^2 - x_i)^2 + (x_{i-1} - 1)^2]$$

**Αρχικά σημεία $x_i$=0.**

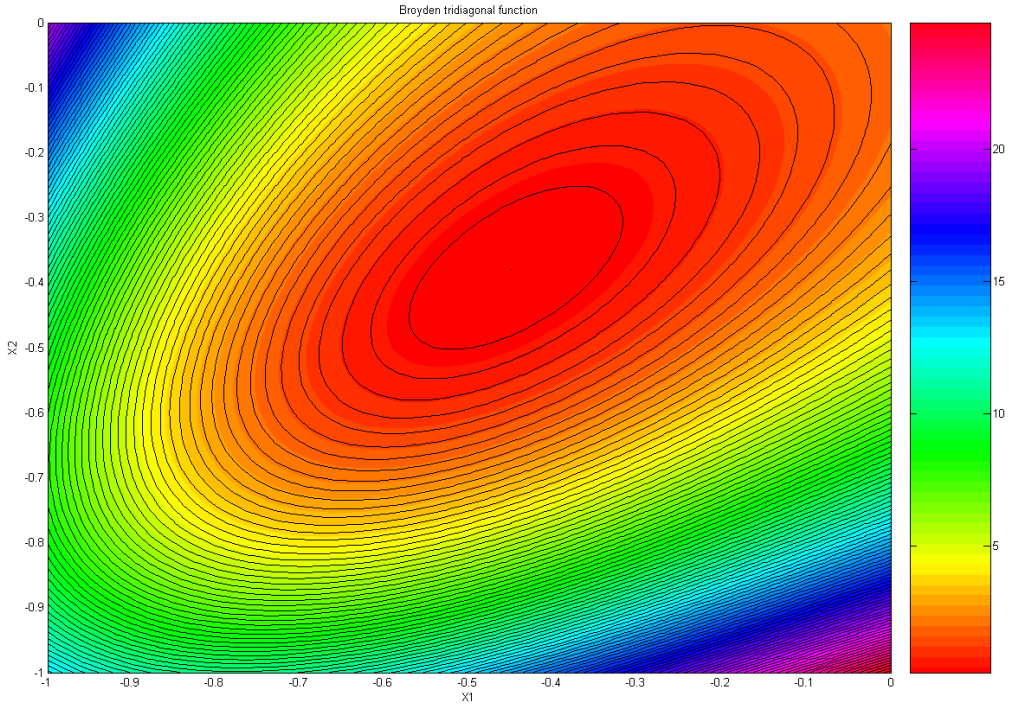



Η συνάρτηση Rosenbrock είναι γνωστή και ως συνάρτηση μπανάνα, λόγω της ιδιομορφίας που έχουν οι ισοϋψείς καμπύλες της. Η ελάχιστη τιμή της συνάρτησης είναι $\boldsymbol{f}(\mathbf{x}^*) = \mathbf{0}$ και οι τιμές των μεταβλητών $\mathbf{x}^* = \mathbf{1}$.

## 3.2 Broyden tridiagonal function

$$f(x)=\sum_{i=1}^{n}[(3-2x_i)x_i-x_{i-1}-2x_{i+1}+1]^2$$

**Αρχικά σημεία $x_i$=-1 και $x_0=x_{n+1}$=0.**

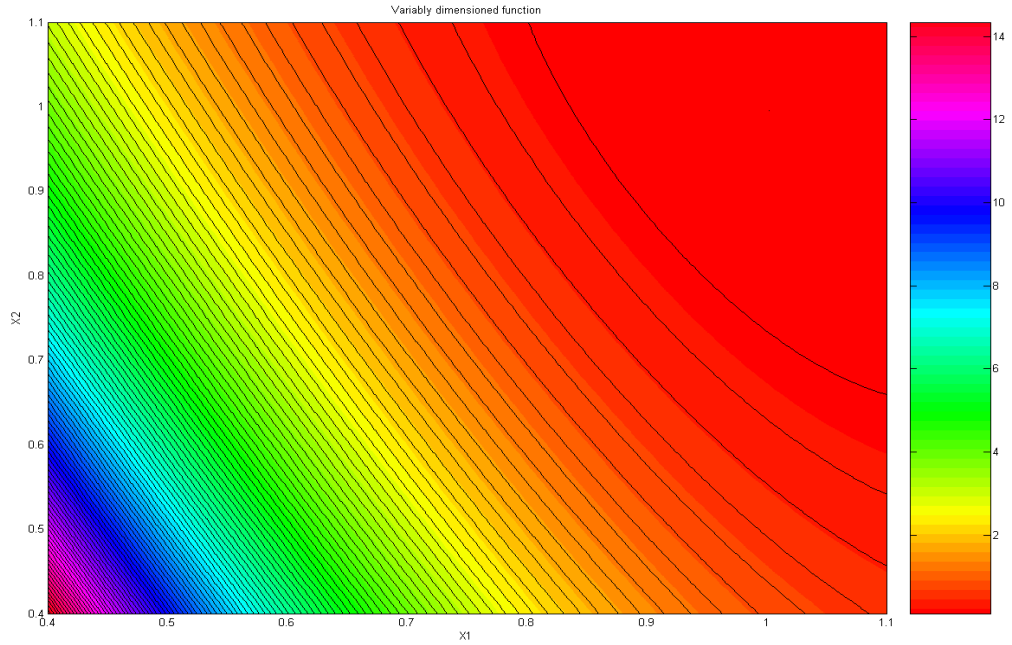



Η συνάρτηση Broyden tridiagonal θεωρείται εύκολη ως προς την λύση της αφού το gradient της συνάρτησης δείχνει σχεδόν την κατεύθυνση προς το ελάχιστο και έτσι οι διορθώσεις είναι μικρές από επανάληψη σε επανάληψη. Η ελάχιστη τιμή της συνάρτησης είναι $\boldsymbol{f}(\mathbf{x}^*)=\mathbf{0}$ και οι τιμές των μεταβλητών $\mathbf{x}^*$ διαφέρουν ανάλογα με το μέγεθος του προβλήματος. Πρέπει να σημειωθεί πως το παραπάνω διάγραμμα δείχνει την συνάρτηση με δύο μεταβλητές, ενώ στον τύπο της εμφανίζει τρείς.

## 3.3 Variably dimensioned function

$$f(x) = \sum_{i=1}^{n} (x_i - 1)^2 + [\sum_{i=1}^{n} i(x_i - 1)]^2 + [\sum_{i=1}^{n} i(x_i - 1)]^4$$

**Αρχικά σημεία $x_i$=1-i/n.**




Στην συνάρτηση Variably dimensioned η ελάχιστη τιμή της συνάρτησης είναι $\boldsymbol{f}(\mathbf{x}^*) = \mathbf{0}$ και οι τιμές των μεταβλητών $\mathbf{x}^* = \mathbf{1}$.

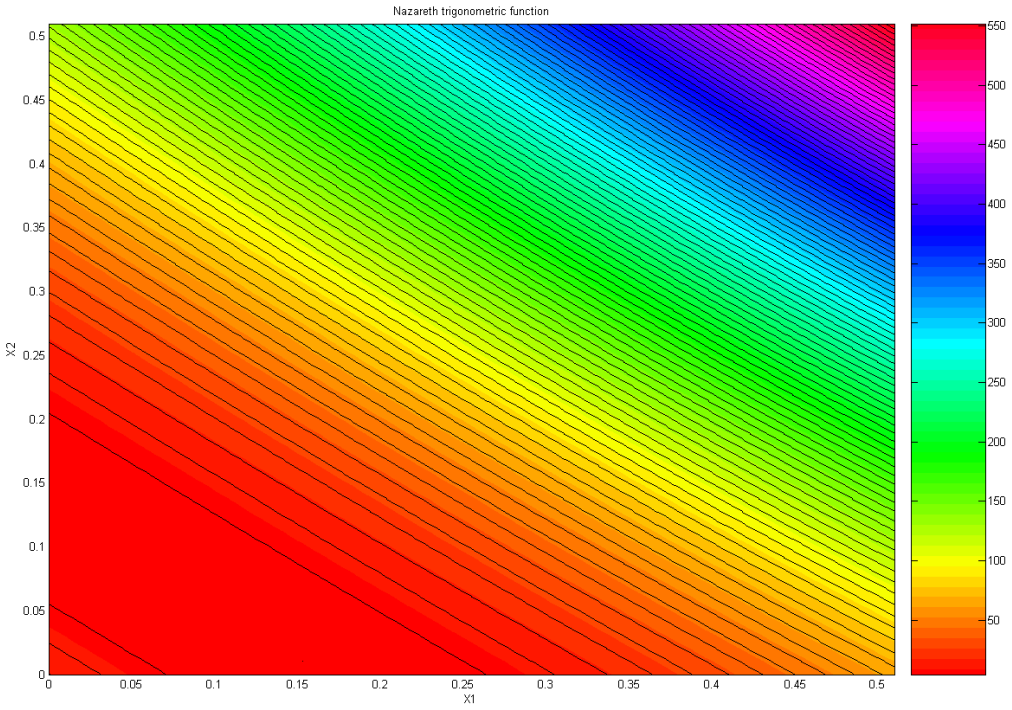
## 3.4 Nazareth trigonometric function

$$f(x) = \sum_{i=1}^{n}[n + i - \sum_{j=1}^{n}(a_{ij}\sin x_j + b_{ij}\cos x_j)]^2$$

$$a_{ij} = 5[1 + \mathrm{mod}(i,5) + \mathrm{mod}(j,5)]$$

$$b_{ij} = (i + j)/10$$

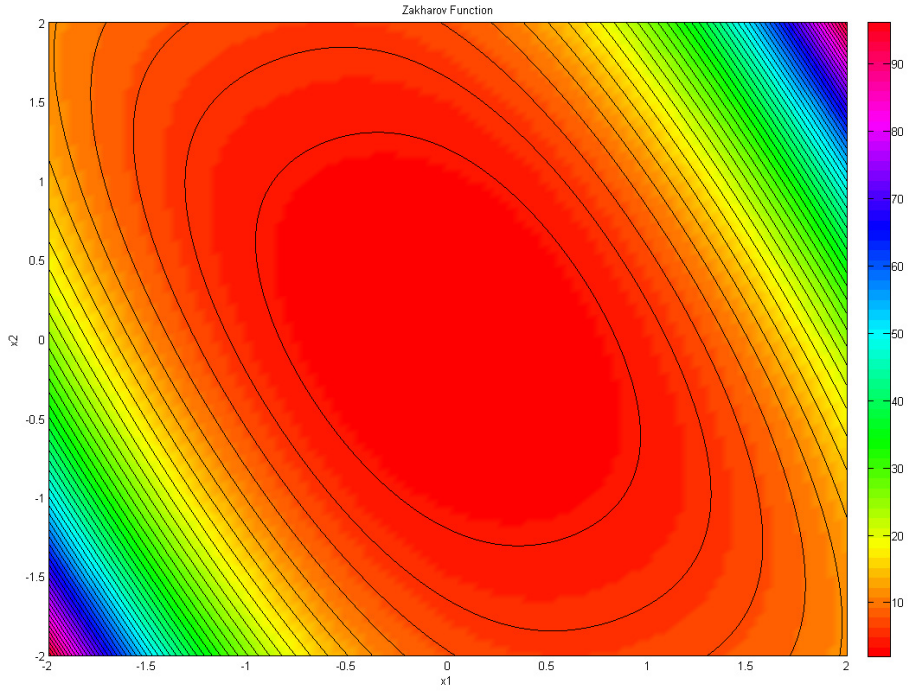
**Αρχικά σημεία $x_i$=1/n.**




Η συνάρτηση Nazareth trigonometric είναι δύσκολη υπολογιστικά αφού έχει στον τύπο της συναρτήσεις ημιτόνων και συνημίτονων. Η ελάχιστη τιμή της συνάρτησης είναι $f(\mathrm{x}^*) = 0$ και οι τιμές των μεταβλητών $\mathrm{x}^*$ διαφέρουν ανάλογα με το μέγεθος του προβλήματος.

## 3.5 Zakharov function

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left( \sum_{i=1}^{n} 0.5 i x_i \right)^2 + \left( \sum_{i=1}^{n} 0.5 i x_i \right)^4$$

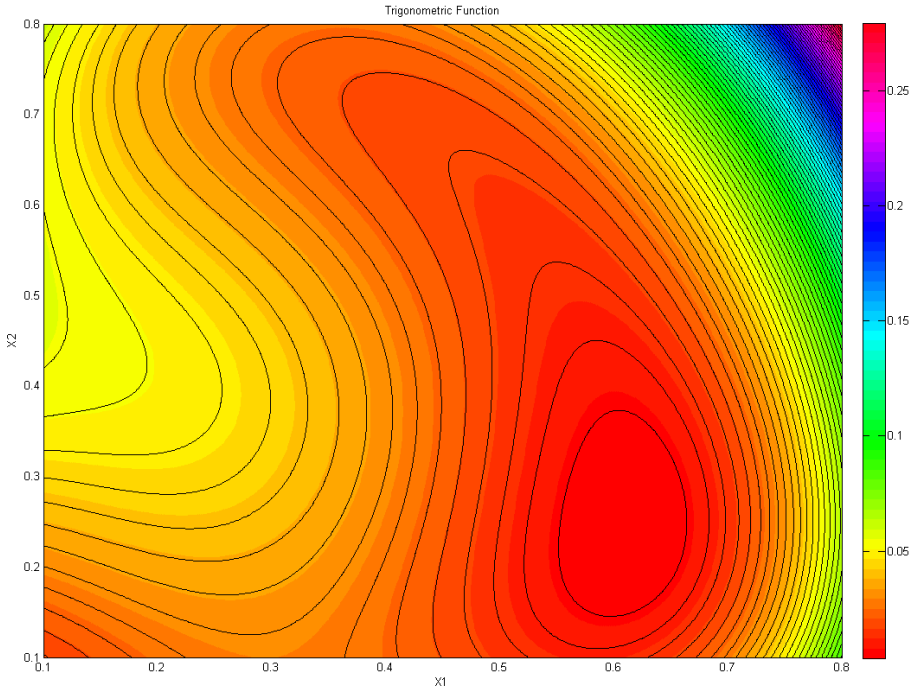



Η συνάρτηση Zakharov δημιουργεί ομόκεντρες ισοϋψείς καμπύλες σε ελλειψοειδές σχήμα. Αρχικά σημεία δεν προτείνονται από τη βιβλιογραφία, οπότε έγινε δοκιμή με διάφορες τιμές. Η ελάχιστη τιμή της συνάρτησης είναι $\boldsymbol{f}(\mathbf{x}^*) = \mathbf{0}$ και οι τιμές των μεταβλητών $\mathbf{x}^* = \mathbf{0}$.

## 3.6 Trigonometric function

$$f(x) = n - \sum_{j=1}^{n} \cos(x_j) + i(1 - \cos x_i) - \sin x_i$$

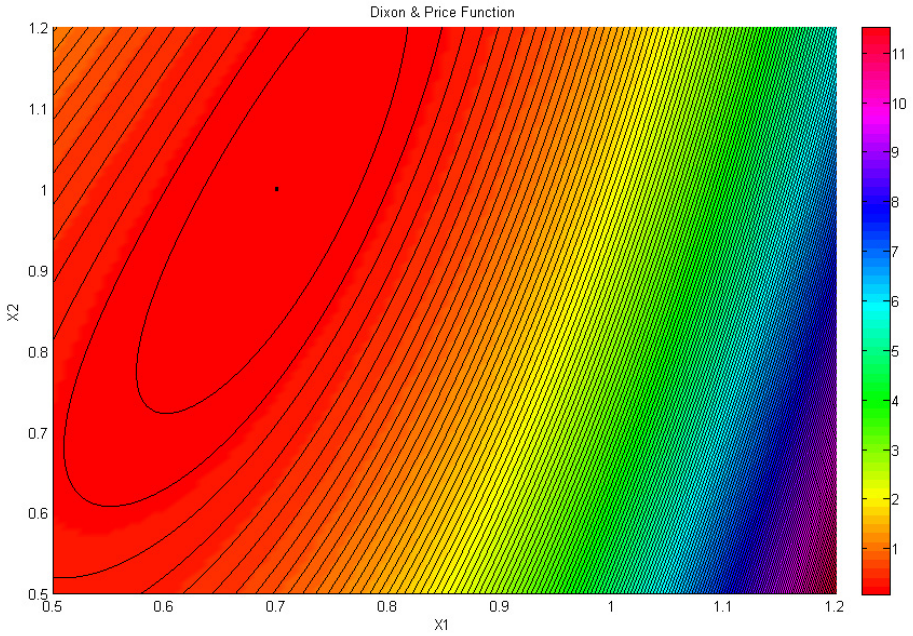
**Αρχικά σημεία $x_i$=1/n.**




Η Trigonometric είναι ακόμα μία συνάρτηση που είχε στον τύπο της τριγωνομετρικές συναρτήσεις. Με βάση το σχήμα που δίνει για δύο μεταβλητές παρατηρούμε πως έχει σύνθετη γεωμετρία. Η ελάχιστη τιμή της συνάρτησης είναι $\boldsymbol{f}(\mathbf{x}^*) = \mathbf{0}$ και οι τιμές των μεταβλητών $\mathbf{x}^*$ διαφέρουν ανάλογα με το μέγεθος του προβλήματος.

## 3.7 Dixon & Price function

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$$




Η συνάρτηση Dixon & Price έχει ελάχιστη τιμή $\boldsymbol{f}(\mathbf{x}^*) = \mathbf{0}$ και οι τιμές των μεταβλητών $\mathbf{x}^*$ διαφέρουν ανάλογα με το μέγεθος του προβλήματος.

# Κεφάλαιο 4: Παρουσίαση αποτελεσμάτων

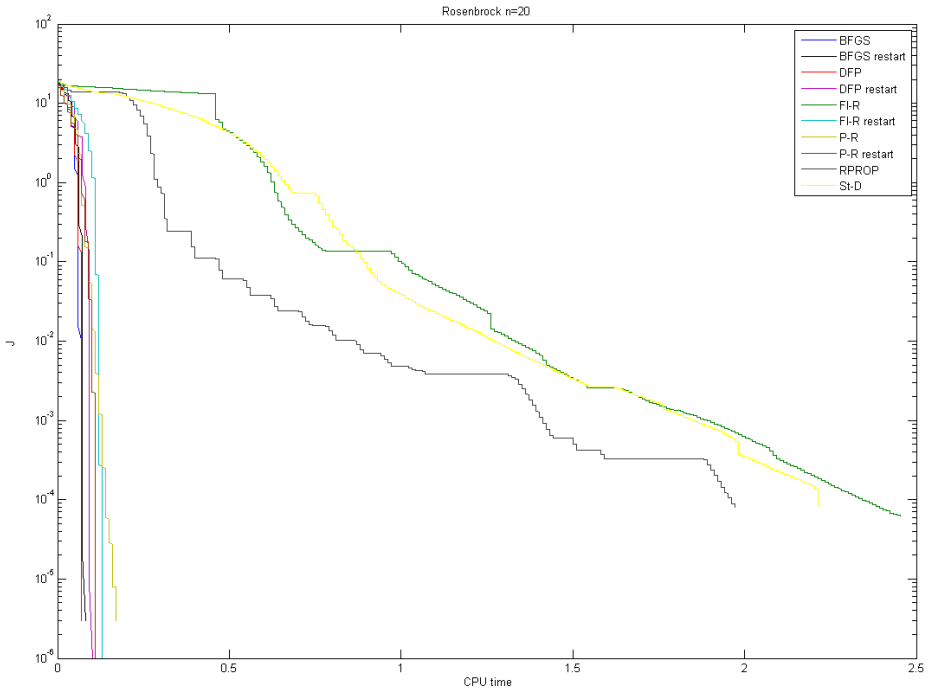
Οι συναρτήσεις που παρουσιάστηκαν στο προηγούμενο κεφάλαιο επιλέχθηκαν με κριτήριο τη δυνατότητα να επιλυθούν προβλήματα με διάφορα μεγέθη. Έτσι στις δόκιμες που έγιναν μεταβάλαμε τον αριθμό των μεταβλητών σε κάθε πρόβλημα. Σε μερικές συναρτήσεις, όπου είχαμε την δυνατότητα, μεγαλώσαμε πολύ τα προβλήματα και δοκιμάσαμε άλλα αρχικά σημεία. Ο σκοπός μας είναι να μελετήσουμε τη συμπεριφορά της μεθόδου RPROP για διάφορα προβλήματα και μεγέθη. Τα αποτελέσματα παρουσιάζονται σε συγκριτικούς αριθμητικούς πίνακες και σε διαγράμματα του χρόνου ελαχιστοποίησης της συνάρτησης ως προς την τιμή της συνάρτησης.

## 4.1 Rosenbrock function

Για την συνάρτηση Rosenbrock τρέξαμε δοκιμές με προβλήματα με n=20, 80, 200. Παρακάτω παρουσιάζονται αναλυτικά τα αποτελέσματα.

| Μέθοδος με n=20 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,070 | 105 |
| DFP restart | 0,101 | 127 |
| BFGS | 0,070 | 92 |
| BFGS restart | 0,081 | 97 |
| Fletcher-Reeves | 2,453 | 7.449 |
| Fletcher-Reeves restart | 0,130 | 212 |
| Polak Ribiere | 0,170 | 287 |
| Polak Ribiere restart | 0,120 | 193 |
| Steepest Descent | 2,213 | 6.744 |
| RPROP | 1,973 | 8.381 |

**Πίνακας 4.1.1**

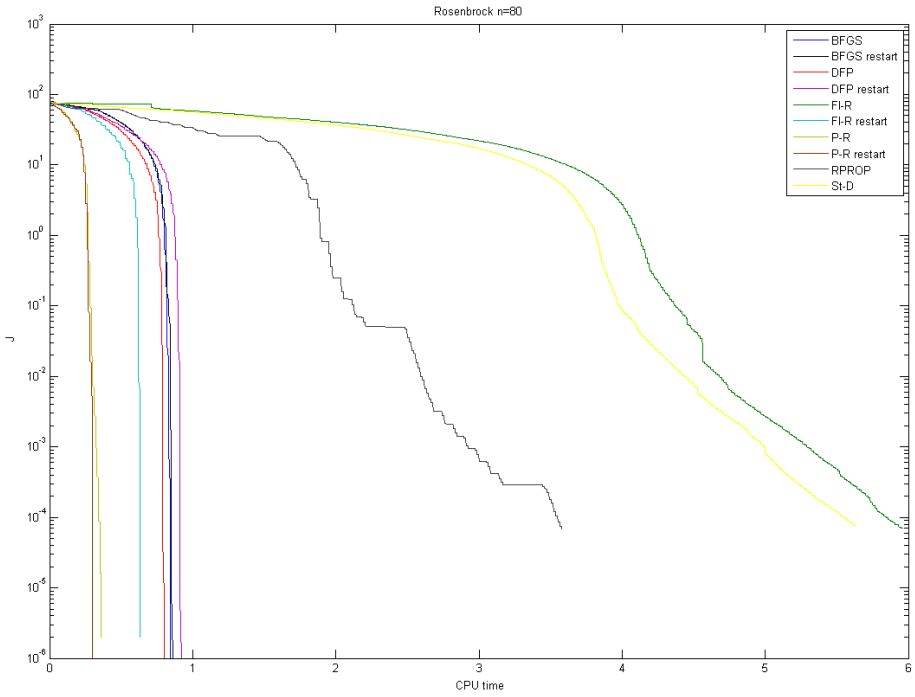



**Σχήμα 4.1.1**

Παρατηρούμε πως η RPROP έρχεται $8^{η}$ σε σύνολο 10 μεθόδων. Επόμενες είναι η Fletcher-Reeves, που φαίνεται να αντιμετωπίζει πρόβλημα χωρίς επανεκκίνηση, και η μεγίστη κατάβαση.

| **Μέθοδος με n=80** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| DFP | 0,801 | 440 |
| DFP restart | 0,921 | 502 |
| BFGS | 0,851 | 343 |
| BFGS restart | 0,862 | 347 |
| Fletcher-Reeves | 5,848 | 16.023 |
| Fletcher-Reeves restart | 0,631 | 1.300 |
| Polak Ribiere | 0,361 | 562 |
| Polak Ribiere restart | 0,301 | 507 |
| Steepest Descent | 5,628 | 15.935 |
| RPROP | 3,575 | 14.348 |

**Πίνακας 4.1.2**

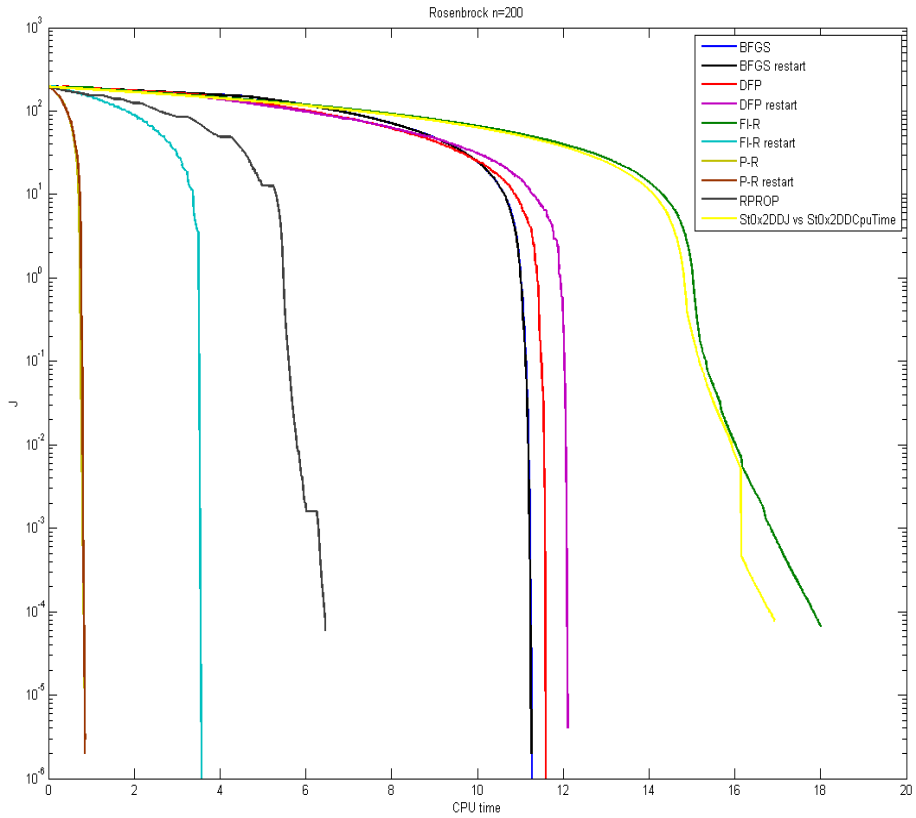



**Σχήμα 4.1.2**

Στη λύση του προβλήματος για n=80, η RPROP εξακολουθεί να βρίσκεται στην 8$^{η}$ θέση.

| **Μέθοδος με n=200** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| DFP | 11,727 | 1.280 |
| DFP restart | 12,548 | 1.385 |
| BFGS | 11,276 | 847 |
| BFGS restart | 11,266 | 851 |
| Fletcher-Reeves | 18,006 | 34.117 |
| Fletcher-Reeves restart | 3,566 | 5.294 |
| Polak Ribiere | 0,851 | 1.096 |
| Polak Ribiere restart | 0,852 | 1.089 |
| Steepest Descent | 16,934 | 32.138 |
| RPROP | 6,469 | 24.832 |

**Πίνακας 4.1.3**




**Σχήμα 4.1.3**

Στο πρόβλημα για n=200, η RPROP παίρνει την $4^{η}$ θέση μετά την Polak-Ribiere, την Polak-Ribiere restart και την Fletcher-Reeves restart. Οι μέθοδοι σχεδόν Newton λόγω των πράξεων με πίνακες που έχουν μεγάλο υπολογιστικό φόρτο, αργούν να βελτιστοποιήσουν τη συνάρτηση. Η Fletcher-Reeves και η μεγίστη κατάβαση εξακολουθούν να έχουν πρόβλημα.

Γνωρίζουμε πως η RPROP είναι απλή υπολογιστικά αλλά χρειάζεται περισσότερες επαναλήψεις για τη λύση των προβλημάτων. Λαμβάνοντας υπόψη αυτό το πλεονέκτημα, προχωρούμε σε δοκιμές για μεγάλα προβλήματα και για αρχικά σημεία μακριά από το ελάχιστο. Σε αυτές τις ακραίες περιπτώσεις η σύγκριση της RPROP γίνεται με την Polak Ribiere, αφού είναι η μέθοδος που ελαχιστοποιεί τη συνάρτηση σε λιγότερο χρόνο. Τα αποτελέσματα παρουσιάζονται στον πίνακα της επόμενης σελίδας.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **$x_i$=0** | **n=200** | | | | | **$x_i$=0** | **n=5000** | | **$x_i$=0** | **n=50000** | |
| | | | | RPROP | P-R | | | | | RPROP | P-R | | RPROP | P-R |
| | | | **iter** | 24.832 | 1.096 | | | | **iter** | 539.789 | 22.708 | **iter** | >2.000.000 | 224.694 |
| | | | **cpu time** | 6,469 | 0,851 | | | | **cpu time** | 1.448,734 | 149,219 | **cpu time** | >46.800 | 20.001,786 |
| | | | | | | | | | | | | | | |
| | | | | | | | | | **$x_i$=0,1*i** | **n=5000** | | | | |
| | | | | | | | | | | RPROP | P-R | | | |
| | | | | | | | | | **iter** | 11.079 | 2.449 | | | |
| | | | | | | | | | **cpu time** | 43,202 | 24,054 | | | |
| | | | | | | | | | | | | | | |
| **$x_i$=100** | **n=20** | | | | | **$x_i$=100** | **n=250** | | **$x_i$=100** | **n=5000** | | | | |
| | RPROP | P-R | | | | | RPROP | P-R | | RPROP | P-R | | | |
| **iter** | 72.986 | 315 | | | | **iter** | 72.930 | 505 | **iter** | 537.505 | 226 | | | |
| **cpu time** | 19,207 | 0,150 | | | | **cpu time** | 19,088 | 0,470 | **cpu time** | 1.764,417 | 2,664 | | | |
| | | | | | | | | | | | | | | |
| **$x_i$=200** | **n=20** | | **$x_i$=200** | **n=200** | | | | | | | | | | |
| | RPROP | P-R | | RPROP | P-R | | | | | | | | | |
| **iter** | 7.365 | 619 | **iter** | 28.153 | 1.157 | | | | | | | | | |
| **cpu time** | 1,973 | 0,250 | **cpu time** | 7,320 | 0,801 | | | | | | | | | |
| | | | | | | | | | | | | | | |
| **$x_i$=1000** | **n=20** | | | | | | | | **$x_i$=1000** | **n=5000** | | | | |
| | RPROP | P-R | | | | | | | | RPROP | P-R | | | |
| **iter** | 984.163 | 2.419 | | | | | | | **iter** | 983.827 | 2.847 | | | |
| **cpu time** | 123,908 | 0,781 | | | | | | | **cpu time** | 2.735,875 | 30,835 | | | |

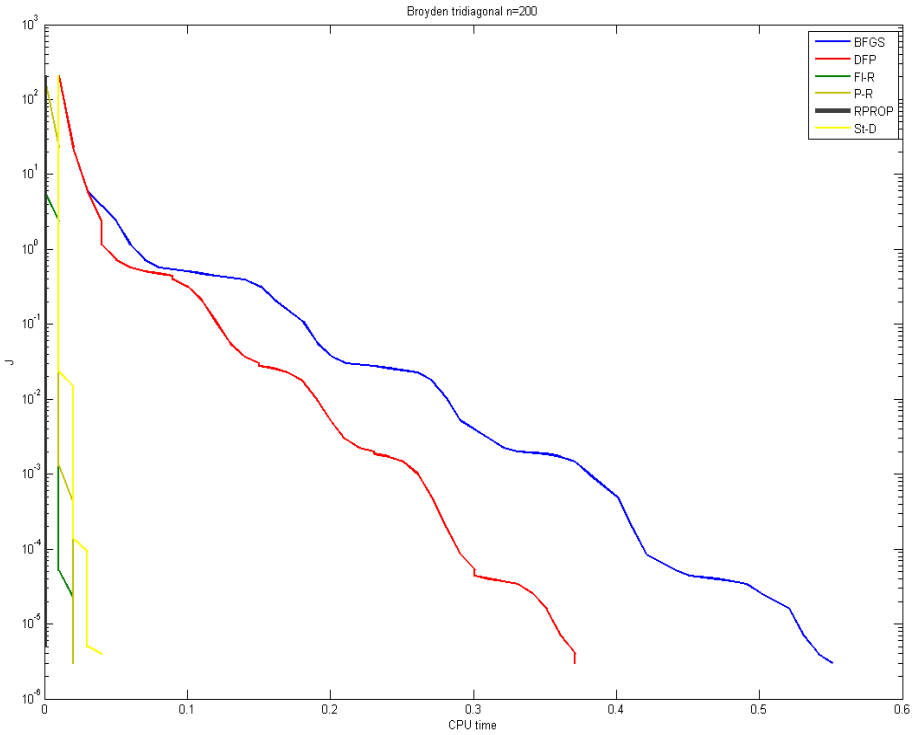
**Πίνακας 4.1.4:** Τιμές της RPROP για μεγάλα προβλήματα και για σημεία μακριά από το βέλτιστο.

## 4.2 Broyden tridiagonal function

Η Broyden tridiagonal είναι εύκολη συνάρτηση ως προς την λύση της, εφόσον σε όλες τις μεθόδους η λύση έρχεται με λίγες επαναλήψεις. Έτσι δοκιμάζοντας μεγάλου μεγέθους προβλήματα για n=200, 500, 1000 βλέπουμε πως η RPROP έχει πολύ καλή επίδοση και λύνει πρώτη το πρόβλημα. Μάλιστα όσο μεγαλώνει το πρόβλημα τόσο γίνεται και αισθητή η διαφορά της.

| Μέθοδος με n=200 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,351 | 41 |
| BFGS | 0,521 | 41 |
| Fletcher-Reeves | 0,030 | 20 |
| Polak Ribiere | 0,030 | 17 |
| Steepest Descent | 0,040 | 34 |
| RPROP | 0,020 | 54 |

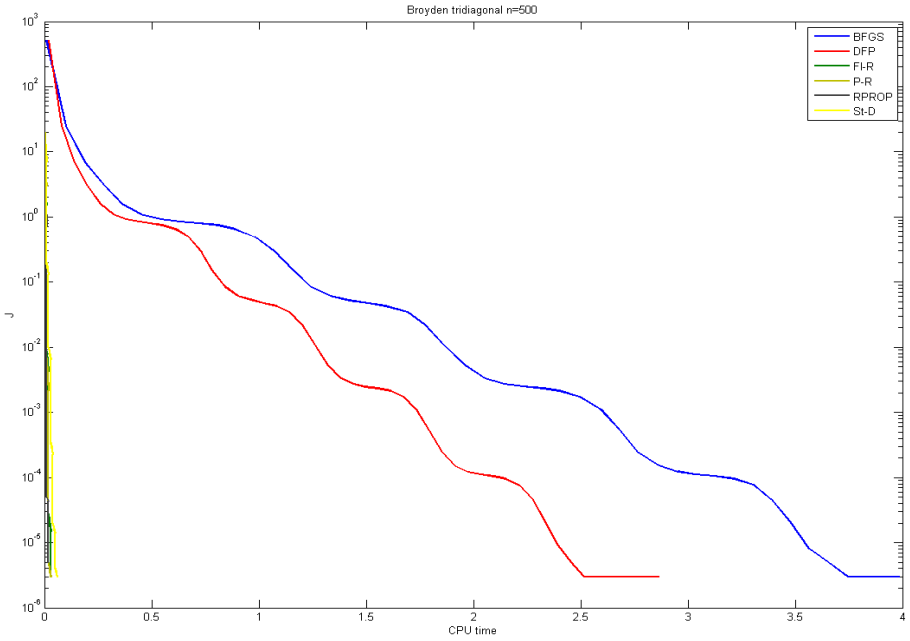
**Πίνακας 4.2.1**



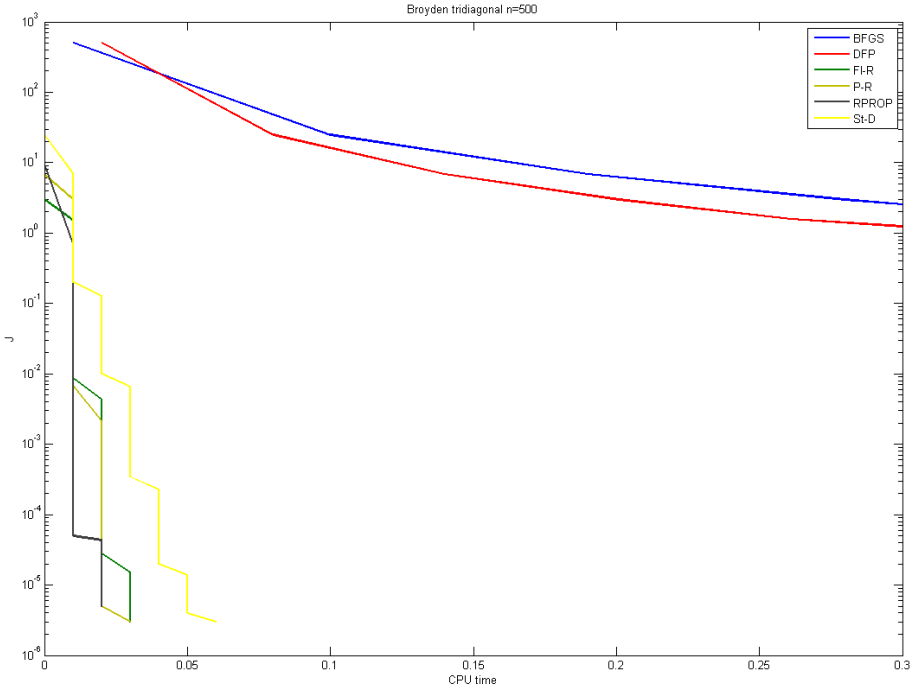

**Σχήμα 4.2.1**

| Μέθοδος με n=500 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 2,744 | 47 |
| BFGS | 3,766 | 44 |
| Fletcher-Reeves | 0,030 | 23 |
| Polak Ribiere | 0,030 | 17 |
| Steepest Descent | 0,060 | 36 |
| RPROP | 0,020 | 54 |

**Πίνακας 4.2.2**



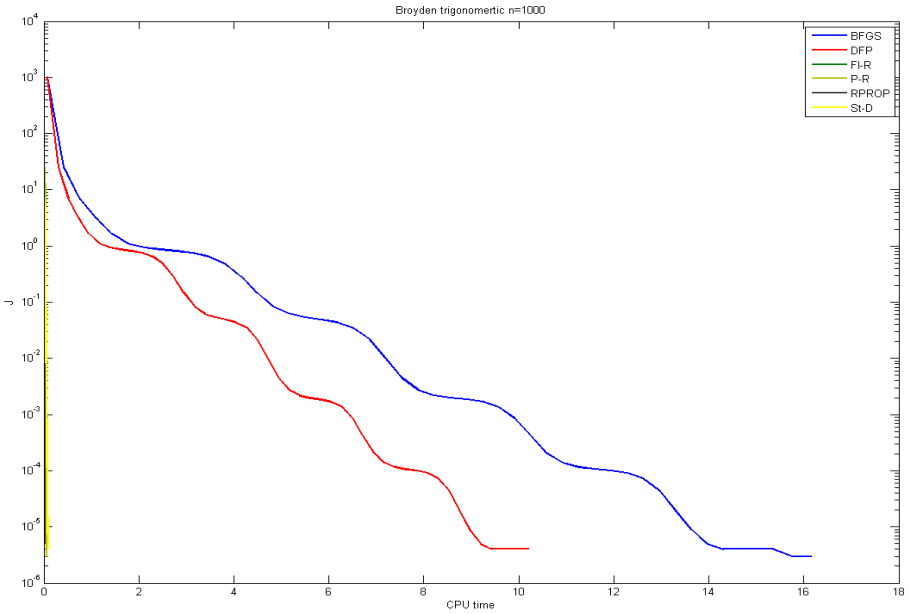

**Σχήμα 4.2.2α**



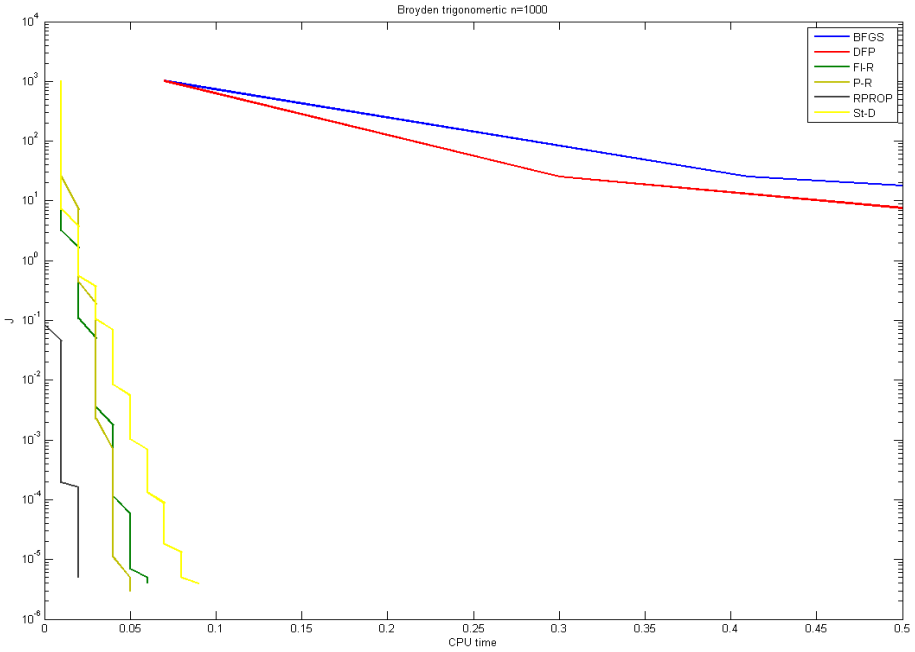

**Σχήμα 4.2.2β**

| Μέθοδος με n=1.000 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 9,794 | 45 |
| BFGS | 15,453 | 47 |
| Fletcher-Reeves | 0,060 | 24 |
| Polak Ribiere | 0,050 | 17 |
| Steepest Descent | 0,080 | 36 |
| RPROP | 0,040 | 54 |

**Πίνακας 4.2.3**




**Σχήμα 4.2.3α**




**Σχήμα 4.2.3β**

Η RPROP έρχεται πρώτη σε όλα τα μεγέθη των προβλημάτων με τα οποία πειραματιστήκαμε, παρά το γεγονός ότι κάνει τις πιο πολλές

επαναλήψεις. Αυτό εξηγείται λόγω του μικρού υπολογιστικού φόρτου που έχει η RPROP ανά επανάληψη. Ένα άλλο στοιχείο άξιο παρατήρησης είναι πως η συγκεκριμένη συνάρτηση λύνεται από όλες τις μεθόδους με σχεδόν ίδιες επαναλήψεις. Οπότε ένα συμπέρασμα που μπορούμε να βγάλουμε για την RPROP είναι ότι για απλά προβλήματα, όπως αυτό της Broyden tridiagonal function η RPROP θα δώσει γρηγορότερα αποτελέσματα για πολύ μεγάλα προβλήματα.

Για να εξακριβώσουμε την τελευταία υπόθεση θα τρέξουμε προβλήματα πολύ μεγαλύτερου μεγέθους για την RPROP, τη μεγίστη κατάβαση και τις μεθόδους των συζυγών κλίσεων. Παρακάτω φαίνονται οι πίνακες αποτελεσμάτων για n=1.000 μέχρι n=500.000, καθώς και ένα διάγραμμα με την cpu time ως προς το μέγεθος του προβλήματος.

| **Μέθοδος με n=10.000** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| Fletcher-Reeves | 0,600 | 25 |
| Polak Ribiere | 0,411 | 17 |
| Steepest Descent | 1,643 | 36 |
| RPROP | 0,601 | 54 |

**Πίνακας 4.2.4**

| **Μέθοδος με n=50.000** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| Fletcher-Reeves | 2,885 | 25 |
| Polak Ribiere | 2,134 | 17 |
| Steepest Descent | 3,666 | 36 |
| RPROP | 1,552 | 54 |

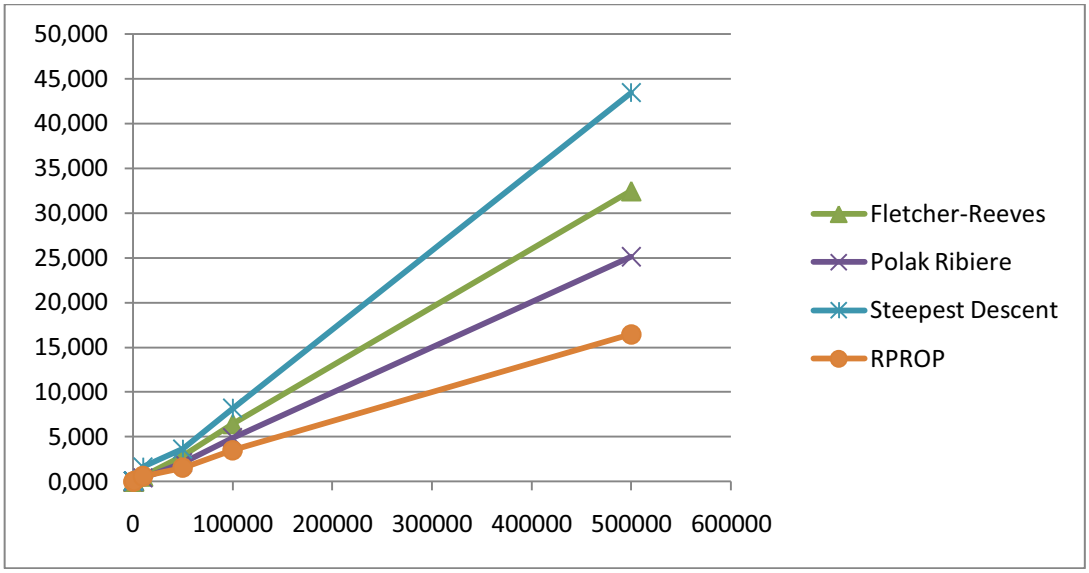
**Πίνακας 4.2.5**

| **Μέθοδος με n=100.000** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| Fletcher-Reeves | 6,460 | 25 |
| Polak Ribiere | 4,907 | 17 |
| Steepest Descent | 8,201 | 36 |
| RPROP | 3,515 | 54 |

**Πίνακας 4.2.6**

| **Μέθοδος με n=500.000** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| Fletcher-Reeves | 32,446 | 25 |
| Polak Ribiere | 25,116 | 17 |
| Steepest Descent | 43,442 | 36 |
| RPROP | 16,444 | 54 |

**Πίνακας 4.2.7**




**Σχήμα 4.2.4**

## 4.3 Variably dimensioned function

Στην συνάρτηση Variably dimensioned η RPROP παρουσιάζει πρόβλημα στην ελαχιστοποίηση της συνάρτησης. Σε αντίθεση οι κλασικές μέθοδοι λύνουν το πρόβλημα πολύ γρήγορα και με λίγες επαναλήψεις. Στη συγκεκριμένη συνάρτηση δεν παρουσιάζονται συγκριτικά διαγράμματα, αφού η διαφορά είναι ξεκάθαρη και η απόσταση των δύο καμπυλών θα ήταν πολύ μεγάλη.

| Μέθοδος με n=20 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,020 | 9 |
| BFGS | 0,010 | 8 |
| Fletcher-Reeves | 0,010 | 9 |
| Polak Ribiere | 0,010 | 10 |
| Steepest Descent | 0,201 | 371 |
| RPROP | 0,801 | 3.976 |

**Πίνακας 4.3.1**

| Μέθοδος με n=50 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,020 | 12 |
| BFGS | 0,010 | 12 |
| Fletcher-Reeves | 0,010 | 13 |
| Polak Ribiere | 0,010 | 11 |
| Steepest Descent | 13,199 | 39.329 |
| RPROP | 15,312 | 61.300 |

**Πίνακας 4.3.2**

| Μέθοδος με n=100 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,020 | 14 |
| BFGS | 0,030 | 14 |
| Fletcher-Reeves | 0,010 | 17 |
| Polak Ribiere | 0,020 | 13 |
| Steepest Descent | 47,168 | 139.232 |
| RPROP | 62,119 | 245.203 |

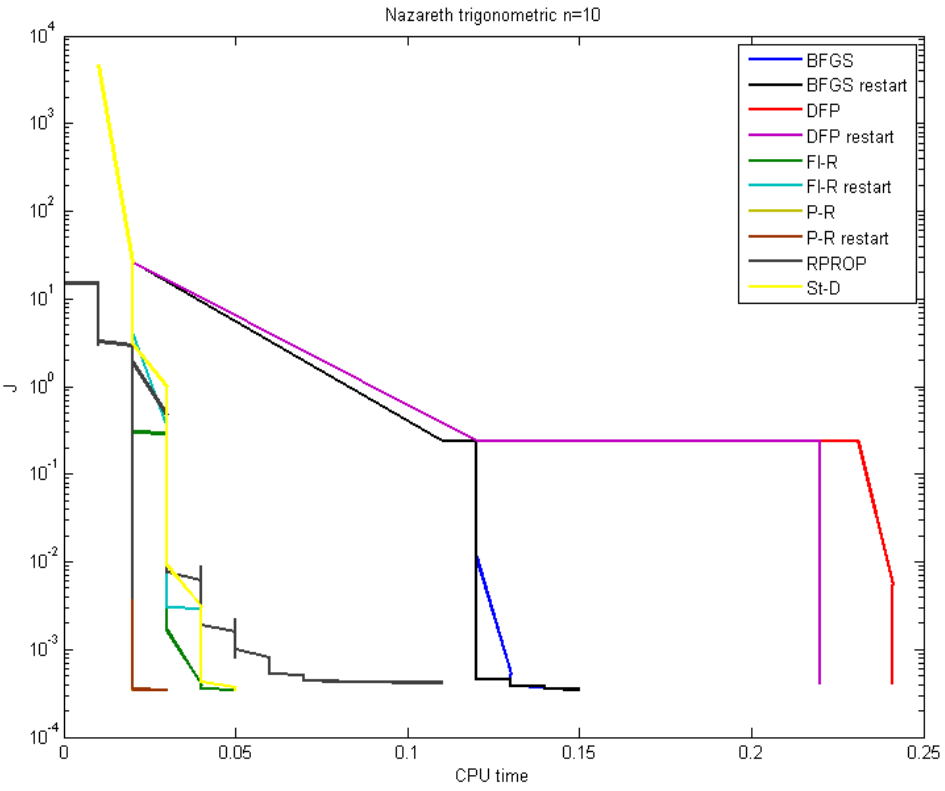
**Πίνακας 4.3.3**

## 4.4 Nazareth trigonometric function

Στη συνάρτηση Nazareth trigonometric η RPROP παρουσιάζει καλή επίδοση στο μικρό πρόβλημα με n=10, αλλά όταν μεγαλώνουμε το n τότε αργεί να δώσει λύση. Αξίζει να αναφέρουμε πως στα μεγαλύτερα προβλήματα, n=30, n=50 η RPROP έχει γρήγορη κατάβαση τα πρώτα δευτερόλεπτα (στις πρώτες επαναλήψεις) και μετά καθυστερεί.

| Μέθοδος με n=10 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,251 | 8 |
| DFP restart | 0,230 | 8 |
| BFGS | 0,150 | 17 |
| BFGS restart | 0,160 | 19 |
| Fletcher-Reeves | 0,050 | 16 |
| Fletcher-Reeves restart | 0,050 | 10 |
| Polak Ribiere | 0,030 | 6 |
| Polak Ribiere restart | 0,030 | 6 |
| Steepest Descent | 0,050 | 16 |
| RPROP | 0,111 | 269 |

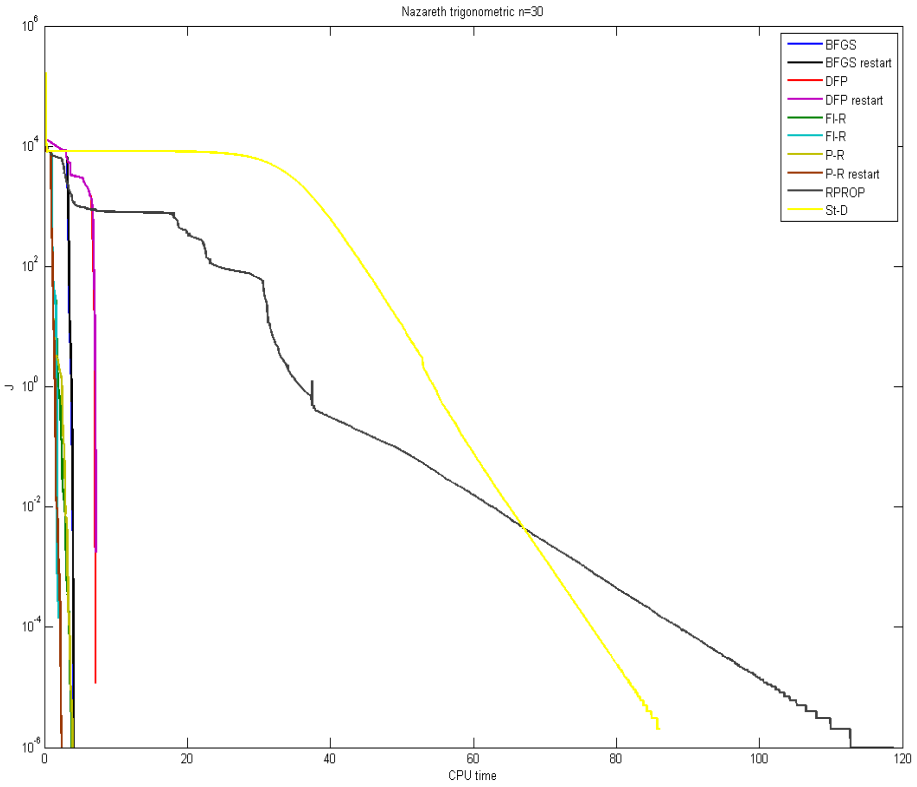
**Πίνακας 4.4.1**




**Σχήμα 4.4.1**

| Μέθοδος με n=30 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 7,191 | 48 |
| DFP restart | 7,290 | 50 |
| BFGS | 4,310 | 30 |
| BFGS restart | 4,136 | 30 |
| Fletcher-Reeves | 3,955 | 91 |
| Fletcher-Reeves restart | 2,033 | 39 |
| Polak Ribiere | 4,026 | 99 |
| Polak Ribiere restart | 2,494 | 56 |
| Steepest Descent | 86,695 | 7.409 |
| RPROP | 121,265 | 15.098 |

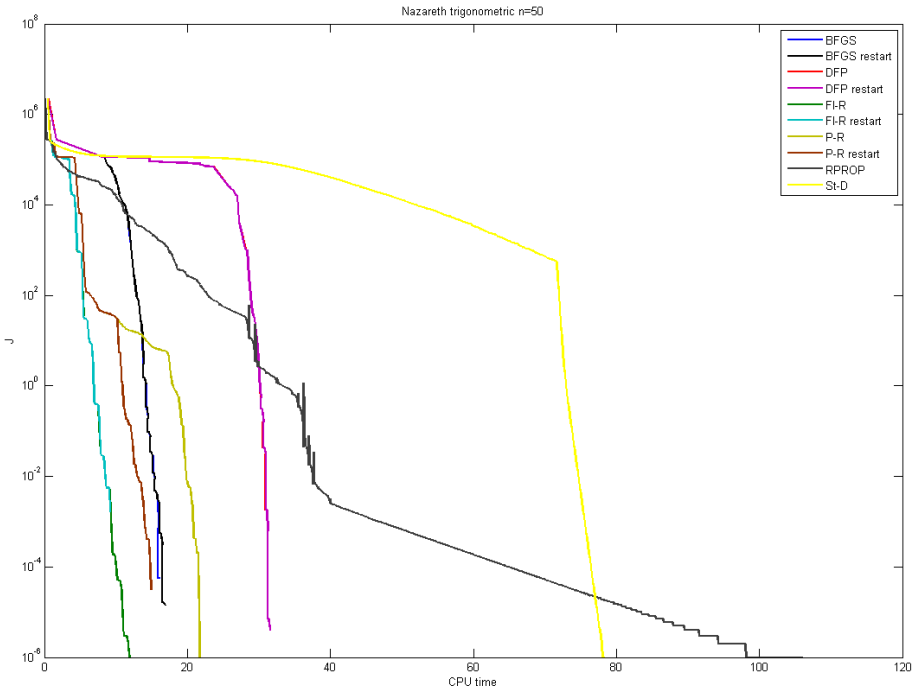
**Πίνακας 4.4.2**




**Σχήμα 4.4.2**

| Μέθοδος με n=50 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 31,685 | 48 |
| DFP restart | 31,726 | 48 |
| BFGS | 16,524 | 55 |
| BFGS restart | 17,165 | 61 |
| Fletcher-Reeves | 12,738 | 73 |
| Fletcher-Reeves restart | 9,343 | 52 |
| Polak Ribiere | 21,851 | 123 |
| Polak Ribiere restart | 15,442 | 79 |
| Steepest Descent | 85,693 | 1.560 |
| RPROP | 128,775 | 3.656 |

**Πίνακας 4.4.3**

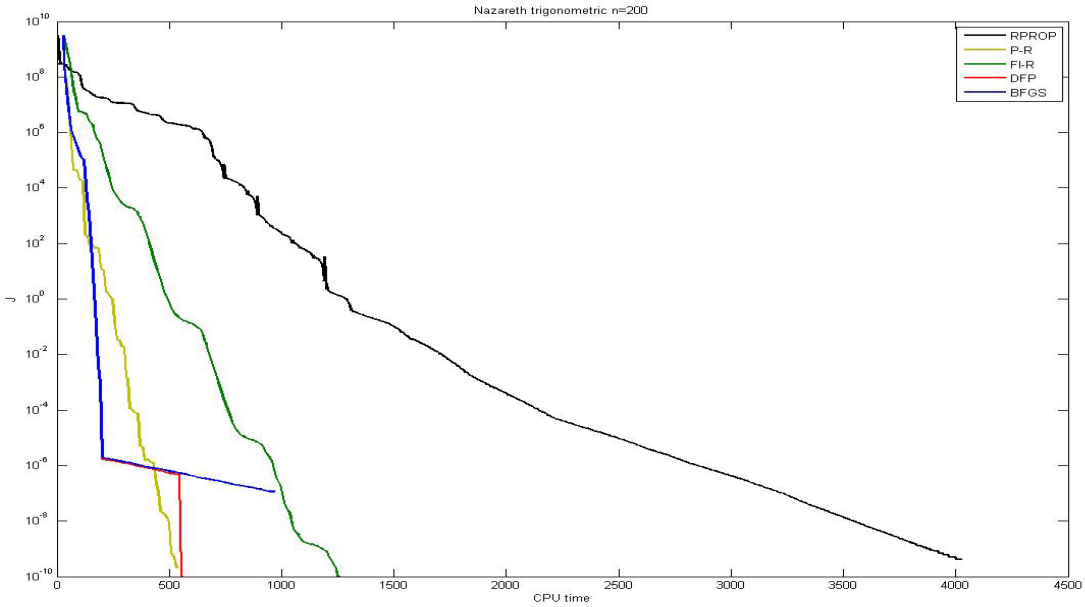



**Σχήμα 4.4.3**

Η RPROP δεν εμφανίζει καλή επίδοση σε αυτή τη συνάρτηση. Έτσι θα μεγαλώσουμε το πρόβλημα, ώστε να εξετάσουμε την πιθανότητα να το λύσει γρηγορότερα από τις άλλες μεθόδους, αφού θα έχει περισσότερους συντελεστές. Οι διαστάσεις των προβλημάτων είναι n=200 και 500. Τα αποτελέσματα φαίνονται πιο κάτω χωρίς η RPROP να βελτιώνει την ταχύτητα επίλυσης σε σχέση με τις άλλες μεθόδους. Επίσης παρουσιάζεται και ένα συγκριτικό διάγραμμα με όλες τις μεθόδους που δείχνει αναλογική αύξηση του χρόνου επίλυσης σε σχέση με το μέγεθος του προβλήματος.

| **Μέθοδος με n=200** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---:|---:|
| DFP | 578,502 | 102 |
| BFGS | 966,580 | 202 |
| Fletcher-Reeves | 1.295,683 | 144 |
| Polak Ribiere | 532,045 | 48 |
| RPROP | 4.025,889 | 1.902 |

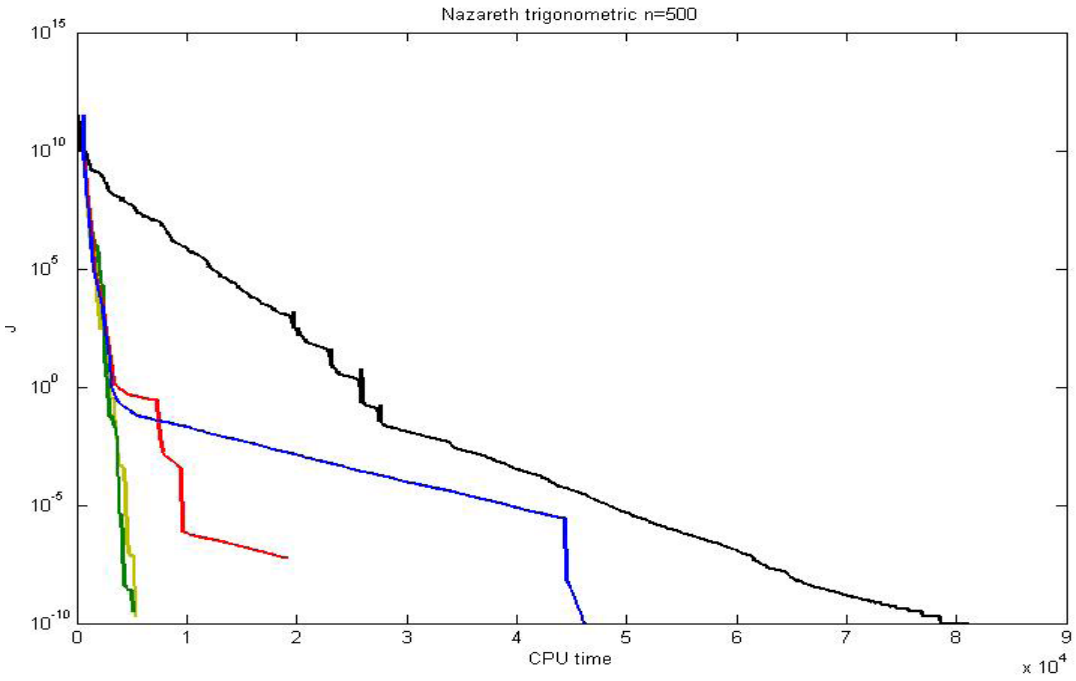
**Πίνακας 4.4.4**



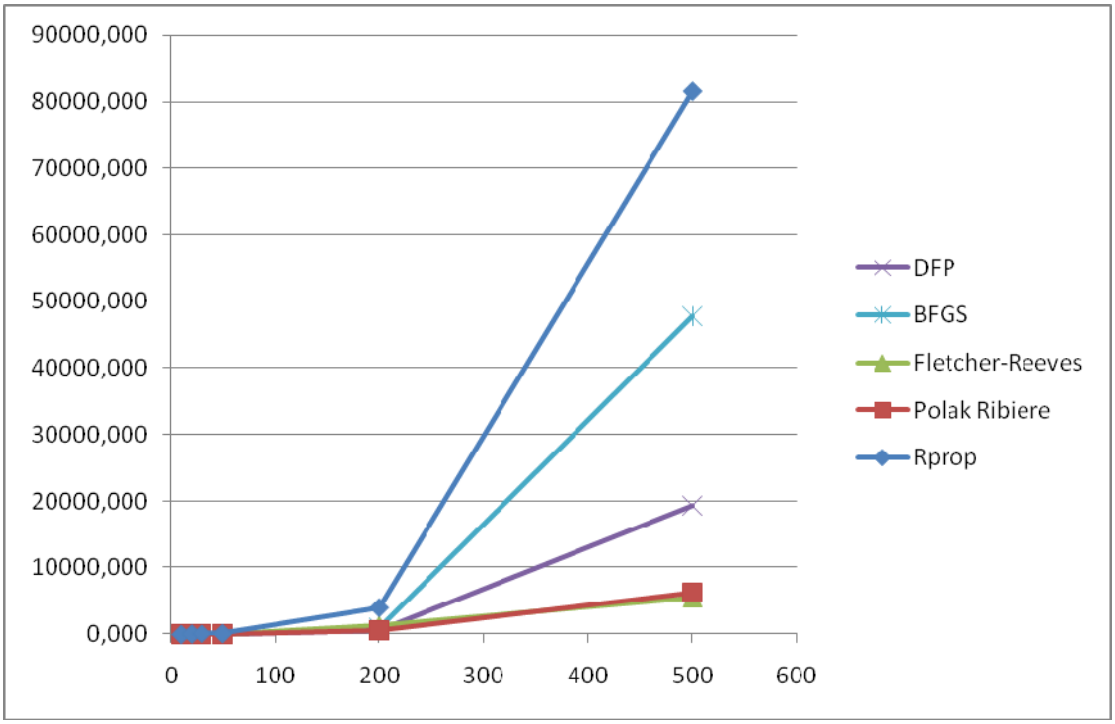

**Σχήμα 4.4.4**

| Μέθοδος με n=500 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 19.392,235 | 274 |
| BFGS | 47.735,520 | 827 |
| Fletcher-Reeves | 5.452,590 | 73 |
| Polak Ribiere | 6.194,260 | 123 |
| RPROP | 81.540,887 | 2.336 |

**Πίνακας 4.4.5**




**Σχήμα 4.4.5**




**Σχήμα 4.4.6**

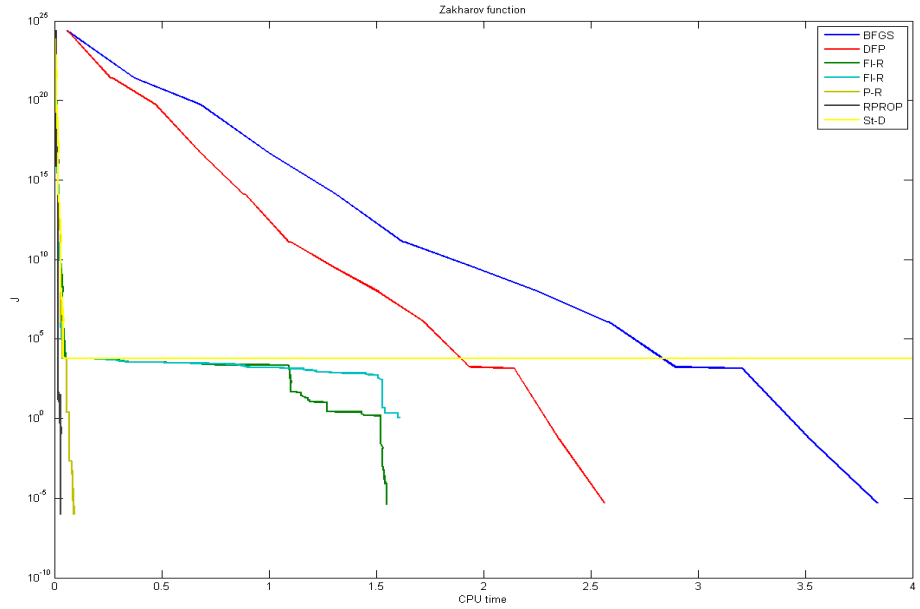
## 4.5 Zakharov function

Η πέμπτη συνάρτηση που μελετήθηκε είναι η Zakharov. Όπως παρατηρήσαμε στο σχήμα της παραγράφου 4.5, για n=2 δημιουργεί ισοϋψείς καμπύλες σε ελλειψοειδές σχήμα. Ο αλγόριθμος έτρεξε για διαφορετικά αρχικά σημεία, τα οποία έδωσαν και πολύ διαφορετική απόδοση στην μέθοδο RPROP. Αναλυτικότερα όταν βάζουμε αρχικές τιμές στα $x_i$=-5 τότε έχουμε πολύ γρήγορη σύγκλιση για πολύ μεγάλο πρόβλημα (n=1000). Αυτό οφείλεται στο ότι τα αρχικά σημεία δίνουν στην RPROP κατεύθυνση κάθετη ως προς όλες τις ισοϋψείς καμπύλες και έτσι θα διαγράψει μια ευθεία μέχρι να βρεθεί στο ελάχιστο. Στον ακριβώς αντίθετο λόγο οφείλεται η κακή επίδοση της RPROP στην δεύτερη περίπτωση, ‘όπου εκεί δίνουμε αρχικά σημεία $x_{2i-1}$=10 και $x_{2i}$=-5, με αποτέλεσμα η RPROP να δυσκολεύεται στην λύση αρκετά μικρότερων προβλημάτων (n=80). Ο λόγος βρίσκεται στον αλγόριθμο της, όπου το βήμα της μεθόδου μικραίνει πάρα πολύ επειδή η κλίση της συνάρτησης αλλάζει συνεχώς πρόσημο.

**Αρχικοί κόμβοι $x_i$=-5**

| **Μέθοδος με n=1000** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| DFP | 2,564 | 20 |
| BFGS | 3,835 | 20 |
| Fletcher-Reeves | 1,552 | 584 |
| Fletcher-Reeves restart | 1,622 | 601 |
| Polak Ribiere | 0,090 | 27 |
| Steepest Descent | 3,632 | 75 |
| RPROP | 0,040 | 70 |

**Πίνακας 4.5.1**

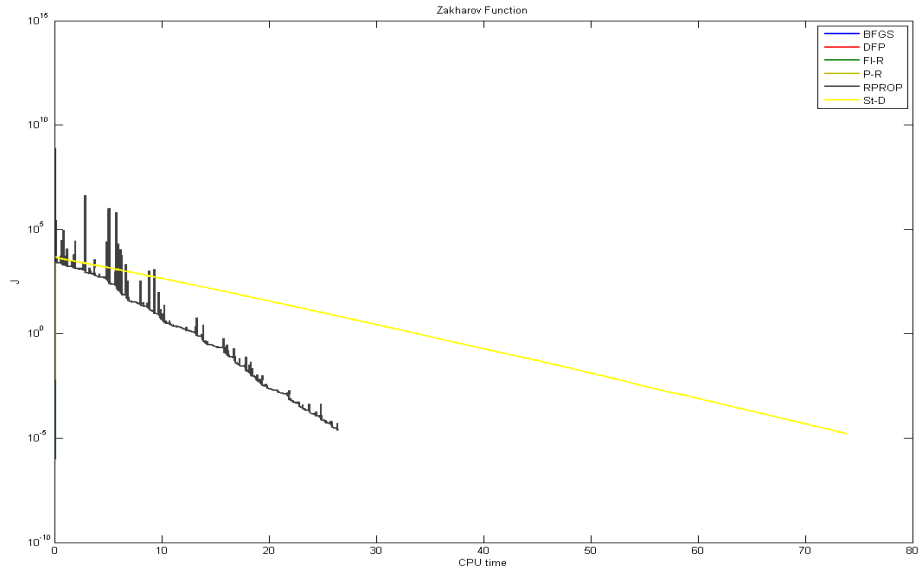



**Σχήμα 4.5.1**

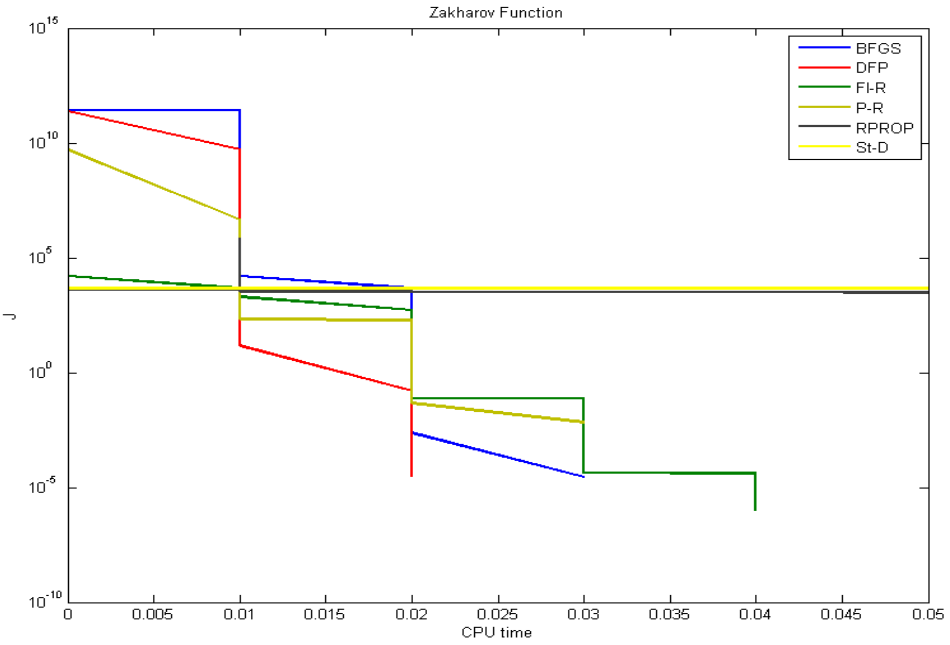
**Αρχικοί κόμβοι $x_{2*i-1}$=10 $x_{2*i}$=-5**

| Μέθοδος με n=80 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,020 | 14 |
| BFGS | 0,030 | 14 |
| Fletcher-Reeves | 0,040 | 57 |
| Polak Ribiere | 0,030 | 35 |
| Steepest Descent | 73,813 | 208.026 |
| RPROP | 26,428 | 102.970 |

**Πίνακας 4.5.2**




**Σχήμα 4.5.2α**



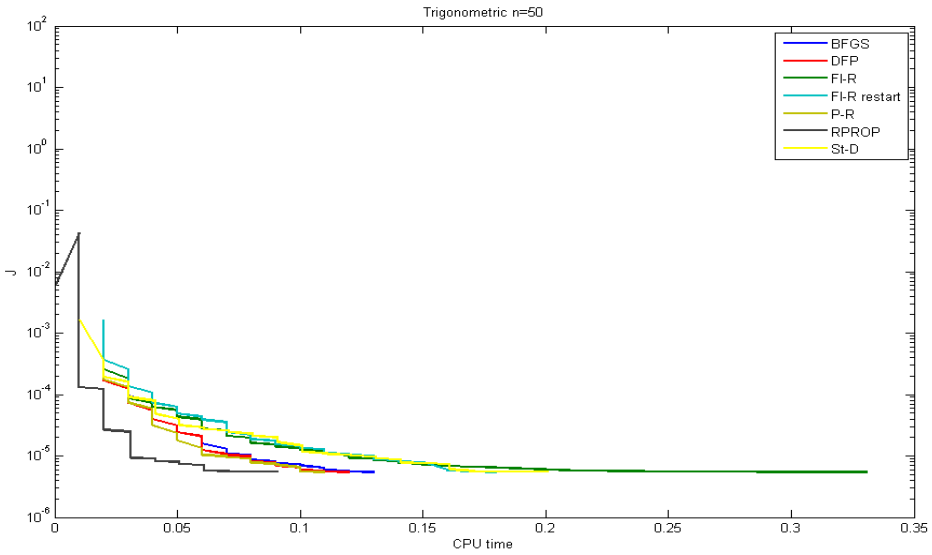

**Σχήμα 4.5.2β**

## 4.6 Trigonometric function

Η έκτη συνάρτηση που μελετήθηκε είναι η Trigonometric Function. Με βάση το σχήμα που δίνει για n=2 (βλ. 4.6) παρατηρούμε πως έχει σύνθετη γεωμετρία. Αρχικά υπήρξε ένα πρόβλημα στη χρήση της συνάρτησης. Επειδή τα σημεία εκκίνησης είναι 1/n, η συνάρτηση κόστους βρίσκονταν κοντά στο ελάχιστο με αποτέλεσμα το κριτήριο σύγκλισης να ικανοποιείται σε λίγες επαναλήψεις. Το πρόβλημα αυτό λύθηκε μεταβάλλοντας το κριτήριο σύγκλισης από $g^2=10^{-4}$ σε $g^2=10^{-8}$. Στη συγκεκριμένη συνάρτηση η RPROP έχει πολύ καλή επίδοση. Έρχεται πρώτη στη βελτιστοποίηση της συνάρτησης για n=50, n=100 και για n=200.

| **Μέθοδος με n=50** | **Χρόνος Σύγκλισης (sec)** | **Επαναλήψεις** |
|---|---|---|
| DFP | 0,120 | 28 |
| BFGS | 0,130 | 28 |
| Fletcher-Reeves | 0,320 | 134 |
| Fletcher-Reeves restart | 0,170 | 58 |
| Polak Ribiere | 0,120 | 29 |
| Steepest Descent | 0,201 | 60 |
| RPROP | 0,100 | 108 |

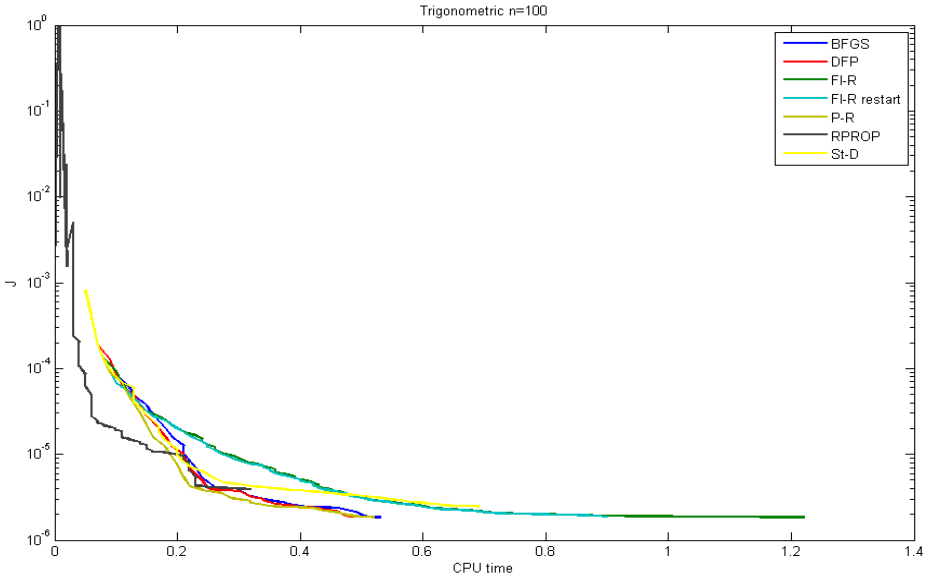
**Πίνακας 4.6.1**




**Σχήμα 4.6.1**

| Μέθοδος με n=100 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 0,510 | 33 |
| BFGS | 0,531 | 33 |
| Fletcher-Reeves | 1,222 | 141 |
| Fletcher-Reeves restart | 0,901 | 100 |
| Polak Ribiere | 0,520 | 38 |
| Steepest Descent | 0,681 | 92 |
| RPROP | 0,340 | 110 |

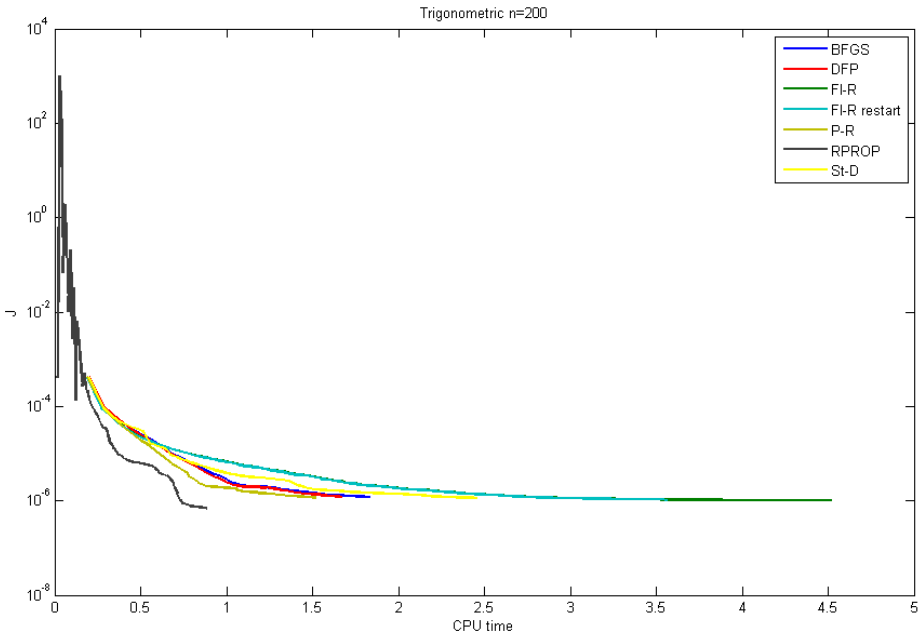
**Πίνακας 4.6.2**




**Σχήμα 4.6.2**

| Μέθοδος με n=200 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 1,672 | 28 |
| BFGS | 1,832 | 28 |
| Fletcher-Reeves | 4,517 | 138 |
| Fletcher-Reeves restart | 3,565 | 100 |
| Polak Ribiere | 1,522 | 30 |
| Steepest Descent | 2,454 | 66 |
| RPROP | 0,912 | 78 |

**Πίνακας 4.6.3**




**Σχήμα 4.6.3**

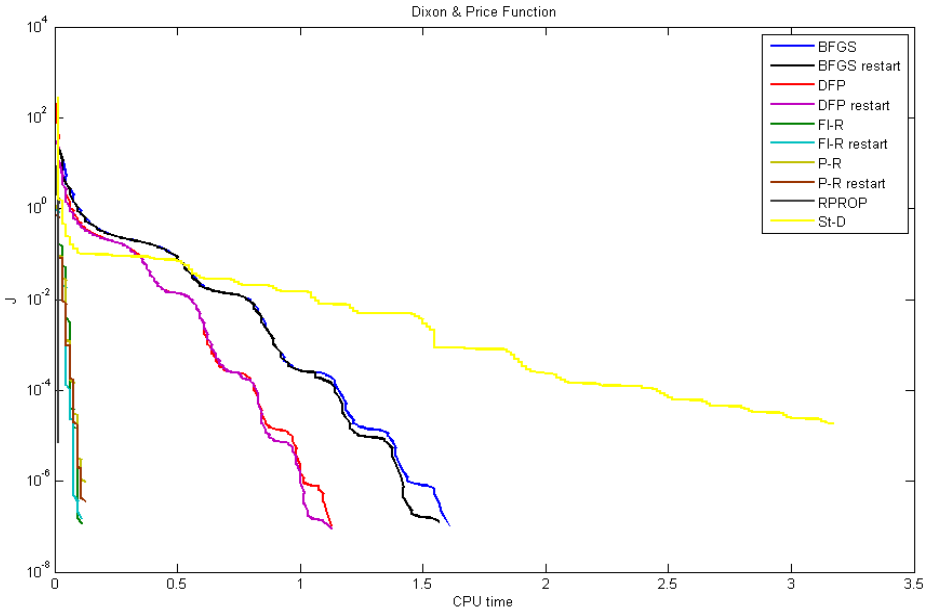
## 4.7 Dixon & Price function

Η έβδομη συνάρτηση που μελετήθηκε είναι η Dixon & Price Function. Η Dixon & Price εμφανίζει πολλά τοπικά ελάχιστα. Για να μας οδηγήσουν όλες οι μέθοδοι στο ίδιο σημείο όπου βρίσκεται και το ολικό ελάχιστο ξεκινούμε με αρχικά σημεία κοντά σε αυτό. Η επίδοση της RPROP είναι πολύ καλή, και μάλιστα αξιοσημείωτο είναι πως για όλα τα μεγέθη των προβλημάτων η RPROP κάνει τις ίδιες επαναλήψεις (62).

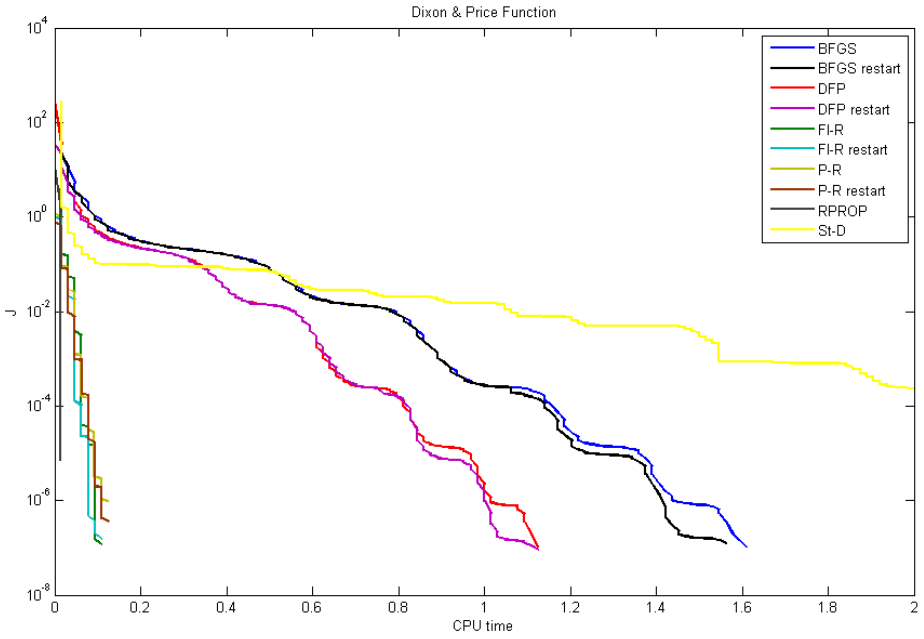
**Αρχικοί κόμβοι $x_i$=0,6**

| Μέθοδος με n=200 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 1,125 | 145 |
| DFP restart | 1,125 | 145 |
| BFGS | 1,609 | 145 |
| BFGS restart | 1,563 | 144 |
| Fletcher-Reeves | 0,109 | 144 |
| Fletcher-Reeves restart | 0,109 | 137 |
| Polak Ribiere | 0,125 | 169 |
| Polak Ribiere restart | 0,125 | 179 |
| Steepest Descent | 3,172 | 2.886 |
| RPROP | 0,047 | 62 |

**Πίνακας 4.7.1**



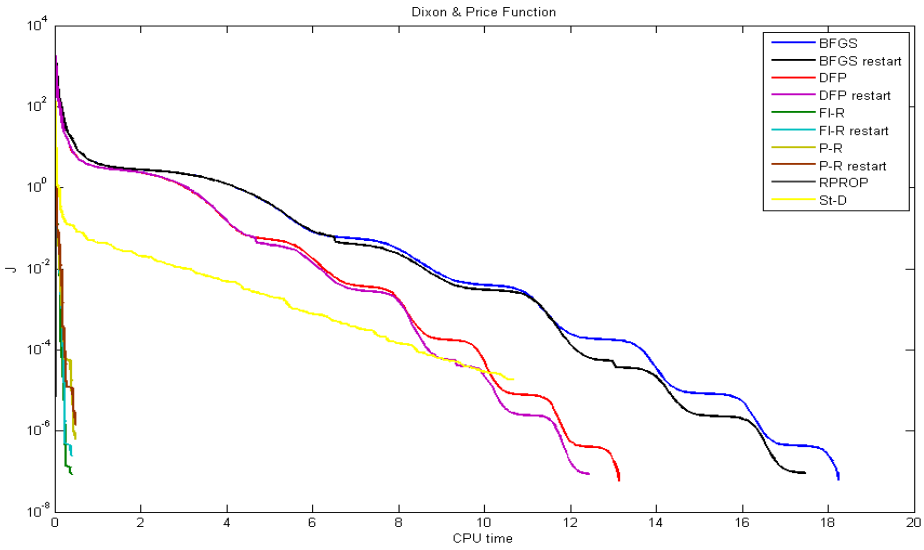

**Σχήμα 4.7.1α**



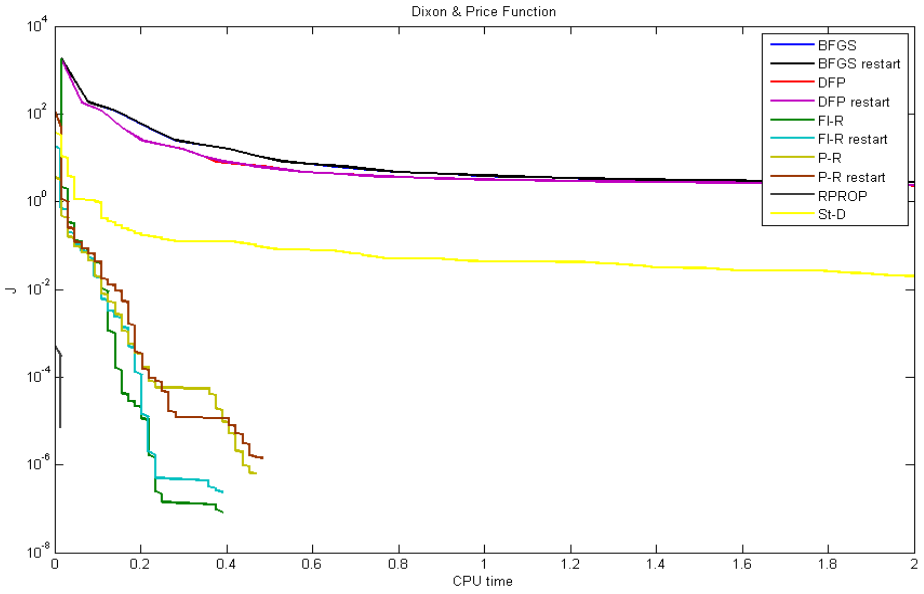

**Σχήμα 4.7.1β**

| Μέθοδος με n=500 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 13,125 | 278 |
| DFP restart | 12,437 | 265 |
| BFGS | 18,234 | 278 |
| BFGS restart | 18,469 | 267 |
| Fletcher-Reeves | 0,391 | 230 |
| Fletcher-Reeves restart | 0,390 | 227 |
| Polak Ribiere | 0,469 | 326 |
| Polak Ribiere restart | 0,485 | 300 |
| Steepest Descent | 10,672 | 9.352 |
| RPROP | 0,063 | 62 |

**Πίνακας 4.7.2**



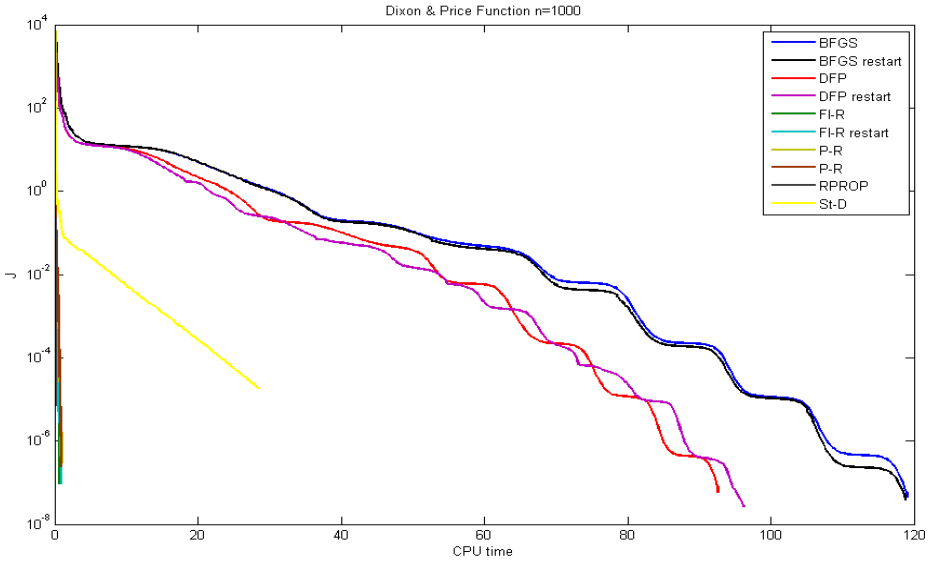

**Σχήμα 4.7.2α**



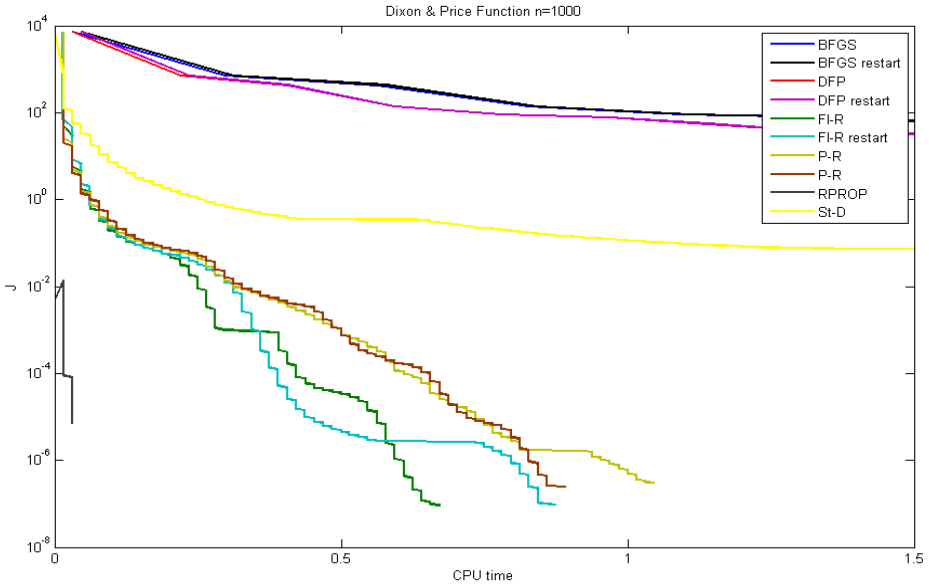

**Σχήμα 4.7.2β**

| Μέθοδος με n=1.000 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| DFP | 92,563 | 448 |
| DFP restart | 96,172 | 526 |
| BFGS | 119,000 | 449 |
| BFGS restart | 118,718 | 450 |
| Fletcher-Reeves | 0,672 | 315 |
| Fletcher-Reeves restart | 0,704 | 388 |
| Polak Ribiere | 1,047 | 507 |
| Polak Ribiere restart | 0,891 | 465 |
| Steepest Descent | 28,625 | 21.294 |
| RPROP | 0,079 | 62 |

**Πίνακας 4.7.3**



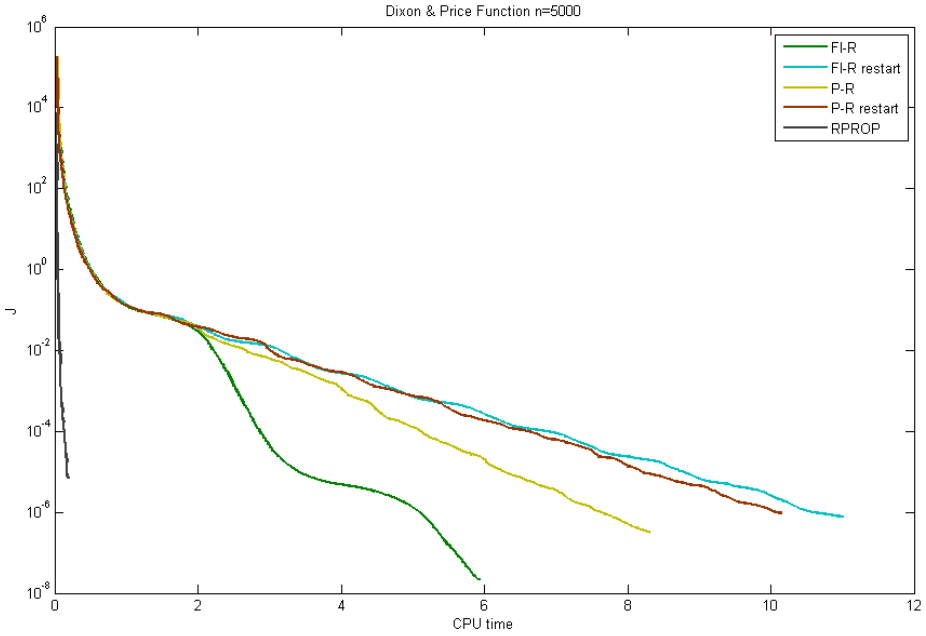

**Σχήμα 4.7.3α**




**Σχήμα 4.7.3β**

| Μέθοδος με n=5.000 | Χρόνος Σύγκλισης (sec) | Επαναλήψεις |
|---|---|---|
| Fletcher-Reeves | 5,938 | 980 |
| Fletcher-Reeves restart | 11,016 | 1.625 |
| Polak Ribiere | 8,313 | 1.351 |
| Polak Ribiere restart | 10,157 | 1.599 |
| RPROP | 0,265 | 62 |

**Πίνακας 4.7.4**




**Σχήμα 4.7.4**

## 4.8 Συνολικά αποτελέσματα

Δημιουργούμε ένα σύστημα αξιολόγησης για να κατατάξουμε την RPROP για κάθε συνάρτηση, αλλά και συνολικά. Ο τύπος που χρησιμοποιήθηκε για την κατάταξη της εκάστοτε μεθόδου για κάθε συνάρτηση είναι:

$$S_i = \sum_j \frac{r_{ji}}{\frac{m}{k}} \qquad (5.1)$$

Όπου το $S_i$ είναι το σκορ που παίρνει η μέθοδος i, θεωρώντας ως καλύτερο το μικρότερο, το $r_i$ είναι η θέση που παίρνει στη δοκιμή μας η μέθοδος i, το m είναι ο αριθμός των μεθόδων που έτρεξαν για τη συγκριμένη συνάρτηση με το συγκεκριμένο μέγεθος και το k είναι ο αριθμός των διαφορετικών μεγεθών που έτρεξαν.

Στον επόμενο πίνακα παρουσιάζεται το σκορ που έχει κάθε μέθοδος σε κάθε συνάρτηση. Το ελάχιστο σκορ αναλογεί στην καλύτερη επίδοση. Με έντονους χαρακτήρες εμφανίζεται η γρηγορότερη μέθοδος.

| | Rosenbrock | Broyden | Variably | Nazareth | Zakharov | Trigonometric | Dixon & Price |
|---|---|---|---|---|---|---|---|
| DFP | 0,53 | 0,83 | 0,56 | 0,80 | 0,42 | 0,33 | 0,70 |
| DFP restart | 0,73 | - | - | 0,83 | - | - | 0,70 |
| BFGS | 0,48 | 1,00 | 0,22 | 0,57 | 0,58 | 0,57 | 0,93 |
| BFGS restart | 0,55 | - | - | 0,60 | - | - | 0,90 |
| Fletcher-Reeves | 0,88 | 0,39 | **0,17** | 0,27 | 0,58 | 1,00 | 0,30 |
| Fletcher-Reeves restart | 0,38 | - | - | **0,17** | - | 0,81 | 0,45 |
| Polak Ribiere | **0,28** | 0,33 | 0,22 | 0,37 | **0,33** | 0,33 | 0,50 |
| Polak Ribiere restart | 0,25 | - | - | 0,20 | - | - | 0,55 |
| Steepest Descent | 0,83 | 0,67 | 0,83 | 0,70 | 1,00 | 0,76 | 0,73 |
| RPROP | 0,60 | **0,17** | 1,00 | 0,87 | 0,50 | **0,14** | **0,13** |

**Πίνακας 4.8.1**

Για τη συνολική κατάταξη των μεθόδων και την τελική κατάταξη της RPROP υπολογίζουμε το μέσο όρο του σκορ από κάθε συνάρτηση. Στον επόμενο πίνακα έχουμε την τελική κατάταξη.

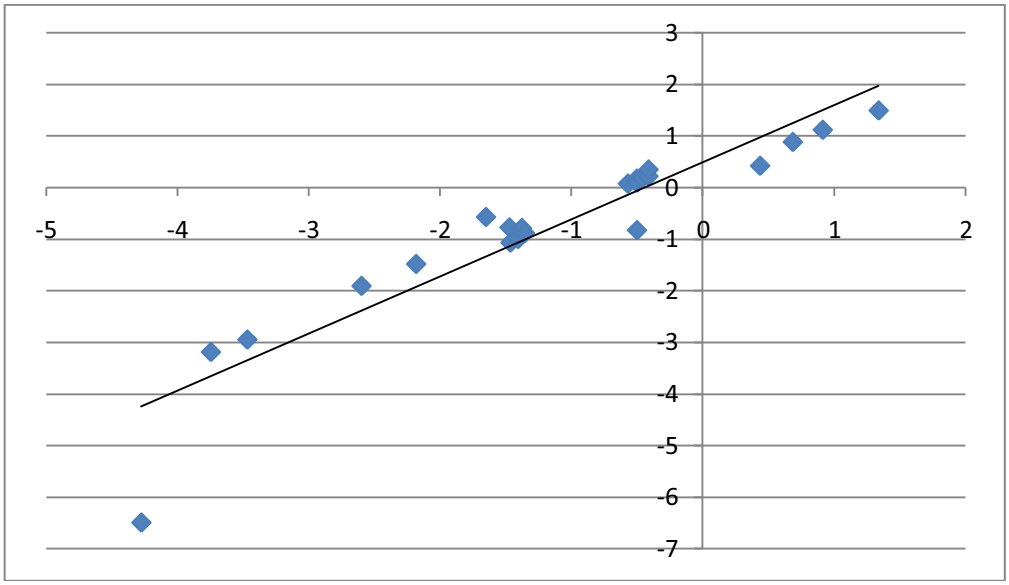
| | | |
|---|---|---|
| 1 | Polak Ribiere restart | 0,33 |
| 2 | Polak Ribiere | 0,34 |
| 3 | Fletcher-Reeves restart | 0,45 |
| 4 | RPROP | 0,49 |
| 5 | Fletcher-Reeves | 0,51 |
| 6 | DFP | 0,59 |
| 7 | BFGS | 0,62 |
| 8 | BFGS restart | 0,68 |
| 9 | DFP restart | 0,75 |
| 10 | Steepest Descent | 0,79 |

**Πίνακας 4.8.2**

Όπως παρατηρούμε η RPROP έρχεται στην τέταρτη θέση μετά την Polak Ribiere restart, την Polak Ribiere και Fletcher-Reeves restart. Ακολουθούν η Fletcher-Reeves, η DFP, η BFGS, η BFGS restart, η DFP restart και Steepest Descent, αφού είδαμε πως είχε πρόβλημα η Fletcher-Reeves σε πολλά προβλήματα αν δεν γινόταν επανεκκίνηση και οι μέθοδοι σχεδόν Newton εμφάνιζαν χρονοβόρες επαναλήψεις για τα προβλήματα μεγάλου μεγέθους.

## 4.9 Σύγκριση RPROP – Polack Ribiere

Σε αυτή την παράγραφο γίνεται η προσπάθεια να βρεθεί μια αναλογία της RPROP και της Polak Ribiere, με τη χρήση του χρόνου και των επαναλήψεων που εκτελούν για την επίλυση κάθε προβλήματος. Φτιάχνουμε το διάγραμμα του λόγου $\log(\frac{\text{iter}_{\text{P-R}}}{\text{iter}_{\text{RPROP}}})$ ως προς το λόγο $\log(\frac{\text{cputime}_{\text{P-R}}}{\text{cputime}_{\text{RPROP}}})$. Στο παρακάτω διάγραμμα διακρίνεται μια γραμμική σχέση των δύο μεταβλητών η όποια είναι και πιο ξεκάθαρη αν αφαιρέσουμε τις ακραίες τιμές.




**Σχήμα 4.9.1**

Βρίσκουμε την ευθεία γραμμικής παλινδρόμησης, η οποία έχει τύπο $y = 1{,}11x + 0{,}49$. Στη συνέχεια βρίσκουμε τη ρίζα της που στην ουσία είναι η αναλογία των επαναλήψεων της RPROP με την Polak Ribiere, στην οποία ο χρόνος επίλυσης ενός προβλήματος θα είναι ίδιος και για τις δύο μεθόδους. Έτσι βρίσκουμε πως όταν η RPROP κάνει 2,78 επαναλήψεις για κάθε μια από τις επαναλήψεις της Polak Ribiere, ο χρόνος επίλυσης θα είναι ίδιος για τις δύο επαναλήψεις.

# Κεφάλαιο 5: Συμπεράσματα

Ο σκοπός της εργασίας ήταν η κατασκευή ενός υπολογιστικού κώδικα για τη βελτιστοποίηση μη γραμμικών προβλημάτων και τη σύγκριση των αποτελεσμάτων των κλασσικών μεθόδων με τη μέθοδο RPROP, για επτά διαφορετικές συναρτήσεις.

Ο υπολογιστικός κώδικας αναπτύχθηκε σε γλώσσα προγραμματισμού C σε περιβάλλον Windows XP. Η αρχιτεκτονική του στηρίζεται στις αρχές του δομημένου προγραμματισμού και εφαρμόζεται δυναμική δέσμευση μνήμης, η οποία δίνει τη δυνατότητα στο χρήστη να μεταβάλει το πλήθος των μεταβλητών της συνάρτησης προς βελτιστοποίηση χωρίς να επεμβαίνει στο πρόγραμμα.

Ο υπολογιστικός κώδικας εφαρμόστηκε στις συναρτήσεις Rosenbrock, Broyden tridiagonal, Variably dimensioned, Nazareth trigonometric, Zakharov, Trigonometric και Dixon & Price. Οι επτά αυτές συναρτήσεις βελτιστοποιήθηκαν με της μεθόδους μεγίστης κατάβασης, Fletcher-Reeves, Polak Ribiere, DFP, BFGS, RPROP και στη συνέχεια συγκρίθηκαν ώστε να υπάρχει η δυνατότητα να βγουν συμπεράσματα για την αποδοτικότητα της RPROP. Έγιναν δόκιμες με πολύ μεγάλα προβλήματα για να εξετάσουμε τη συμπεριφορά της RPROP σε αυτά και με διάφορα αρχικά σημεία.

Μέσω της μελέτης που έγινε καταλήξαμε στα παρακάτω συμπεράσματα:

- Η μέθοδος RPROP σε τρείς συναρτήσεις (Broyden tridiagonal, Trigonometric, Dixon & Price) λαμβάνει την $1^{η}$ θέση ως προς το κριτήριο της ταχύτητας.
- Συνολικά κατατάσσεται στην $4^{η}$ θέση μετά την Polak Ribiere με επανεκκίνηση, την Polak Ribiere και την Fletcher-Reeves με επανεκκίνηση.
- Η μέθοδος RPROP κερδίζει τις μεθόδους σχεδόν Newton στα μεγάλα προβλήματα λόγω της υπολογιστικής απλότητας της και των πράξεων των πινάκων που απαιτούν οι μέθοδοι των σχεδόν Newton.
- Η μέθοδος της επανεκκίνησης αποδίδει στις μεθόδους των συζυγών κλίσεων μειώνοντας το χρόνο επίλυσης, ενώ στις μεθόδους σχεδόν Newton δεν φαίνεται να έχει μεγάλη επίδραση.
- Ακόμα και στα προβλήματα πολλών μεταβλητών (πολύ μεγάλα προβλήματα) η μέθοδος RPROP δεν αλλάζει τη σχετική της επίδοση σε σχέση με την Polak Ribiere, που είναι η γρηγορότερη μέθοδος.
- Όταν η μέθοδος RPROP εκτελεί 2,78 επαναλήψεις για κάθε επανάληψη της Polak Ribiere τότε ο χρόνος βελτιστοποίησης της συνάρτησης είναι σχεδόν ίσος. Για λόγο μεγαλύτερο θα προηγηθεί η Polak Ribiere ενώ για μικρότερο η RPROP.
- Στη συνάρτηση Broyden tridiagonal, η οποία είναι μια σχετικά απλή συνάρτηση αθροίσματος τετραγώνων, όλες οι μέθοδοι βελτιστοποιούν τα διάφορα μεγέθη του προβλήματος σε σχεδόν ίδιες επαναλήψεις.
- Στις συναρτήσεις Broyden tridiagonal και Dixon & Price η μέθοδος RPROP, όπου είναι η γρηγορότερη, βελτιστοποιεί τις συναρτήσεις σε ίδιες επαναλήψεις για όλα τα μεγέθη των προβλημάτων.

- Σε κάποιες συναρτήσεις, όπως η Nazareth trigonometric, όπου η μέθοδος RPROP δεν έχει καλή επίδοση, γίνεται μία απότομη πτώση της τιμής της αντικειμενικής συνάρτησης, στις πρώτες επαναλήψεις. Έτσι, προτείνεται νέα έρευνα για την κατασκευή υβριδικών αλγορίθμων και τη διερεύνηση του κριτηρίου επιλογής μεθόδου κατά τη διαδικασία της επίλυσης μέσα σε ένα ενιαίο αλγόριθμο. Οπότε θα δημιουργηθεί ένας αλγόριθμος που θα συμπεριφέρεται καλύτερα στην κάθε περιοχή της συνάρτησης.
- Σαν κριτήριο επιλογής της μεθόδου μπορεί να επιλεχτεί ο λόγος επαναλήψεων που έχει βρεθεί και συσχετίζει τις μεθόδους RPROP και Polak Ribiere.

# Βιβλιογραφία


Abe,S. (1997) “Neural Networks and Fuzzy Systems. Theory and Applications”, Kluwer Academic Publishers, Massachusetts,U.S.A.

Broyden, C.G. (1967) “Quasi-Newton methods and their application to function minimization”, Math. Comp. 21 ,pp. 368-381.

Fletcher, R. (1987) “Practical Methods of Optimization”, Second Edition, John Wiley & Sons, Chichester, U.K.

Forsythe, G.E. (1968) “On the asymptotic directions of the s-dimensional optimum gradient method” ,Numer. Math. 11, pp.57-76.

Foulds, G.E. (1981) “Optimization Techniques”, Springer-Verlag.

Gill, P.E. , Murray, W. ,Wright,M.H. (1981) “Practical Optimization”, Academic Press.

Hestenes, M. (1980) “Conjugate Direction Methods in Optimization”,Springer-Verlag.

Καραμπιπέρης ,Π. (2002) “Έρευνα απόδοσης προσαρμοστικών αλγορίθμων σε προβλήματα μη γραμμικής βελτιστοποίησης χωρίς περιορισμούς”, Μεταπτυχιακή Εργασία, Πολυτεχνείο Κρήτης, Τμήμα Μηχανικών Παραγωγής και Διοίκησης, Χανιά.

Luksan L. (1992) “Computational Experience with Known Variable Metric Updates”, Technical Report No. V-534, Institute of Computer Science, Prague.

Μαρινάκης, Ι. (1999) “Διερεύνηση ιδιοτήτων αλγορίθμων μη γραμμικής Βελτιστοποίησης”, Διπλωματική Εργασία, Πολυτεχνείο Κρήτης,Τμήμα Μηχανικών Παραγωγής και Διοίκησης, Χανιά.

Ντεληδήμος, Γ. (1997) “Αλγόριθμος Βελτιστοποίησης μη γραμμικών συναρτήσεων χωρίς περιορισμούς”,Διπλωματική Εργασία, Πολυτεχνείο Κρήτης,Τμήμα Μηχανικών Παραγωγής και Διοίκησης, Χανιά.

Papageorgiou, M. (1996) “Optimierung”, Second Edition, OldenbourgVerlag, Munich, Germany.

Papageorgiou, M.,and Marinaki,M. (1995) “A Feasible Direction Algorithm for the Numerical Solution of Optimal Control Problems”,Internal Report No.1995-4,Dynamic System and Simulation Laboratory,Technical University of Crete,Chania, Greece.

Παπαγεωργίου, Μ. (1998) “Μη γραμμικός Προγραμματισμός”,Πολυτεχνείο Κρήτης,Τμήμα Μηχανικών Παραγωγής και Διοίκησης,Χανιά.

Poulimenos, A., and Papageorgiou, M. (1999) “Methodological Design of the Prediction Module”,WP3-Deliverable D3.1.2.-DELPHI prject No. 26965, Technical University of Crete,Chania, Greece.

Powell,M.J.D. (January 1970) “A Survey of Numerical Methods for Unconstrained Optimization”,SIAM Review, **12**,1, 79-97.

Powell,M.J.D. (September 1979) “Optimation Algorithms”,Proc. of the 9 * IFIP Conference on optimization Techniques, Warsaw.

Riedmiller, M.,and Braun, H. (1993) “A Direct AdaptiveMethod for Faster Backpropagation Learning: The RPROP Algorithm”, Proceedings of the IEEE Interational Conference on Neural Networks,San Francisco,U.S.A.

Scales,L.E, (1985) “Introduction to Non-linear Optimization”,MacMillan.

http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm

# Παράρτημα

## Εγχειρίδιο χρήσης προγράμματος

Όπως αναφέρεται και στη παράγραφο 2.2, τα δεδομένα του κάθε προβλήματος εισάγονται στο πρόγραμμα μέσω του αρχείου dedomena.txt που θα πρέπει να βρίσκεται στον ίδιο φάκελο με το αρχείο *.exe που έχει δημιουργηθεί. Με τον τρόπο αυτό επιτυγχάνεται μεγαλύτερη ταχύτητα στις διάφορες δοκιμές που χρειάζεται να κάνει ο χρήστης. Τα αποτελέσματα εκτυπώνονται σε δυο αρχεία με την ονομασία της μεθόδου που τρέχουμε, με το αν έχουμε επιλέξει επανεκκίνηση και με τα διακριτικά J για το αρχείο τιμών της αντικειμενικής συνάρτησης και cpu time για τους αντίστοιχους χρόνους. Έτσι για παράδειγμα αν έχουμε τρέξει την μέθοδο Polak Ribiere με επανεκκίνηση θα έχουμε τα εξής δύο αρχεία:

- P-R re cpu time
- P-R re J

Όσον αφορά την εισαγωγή δεδομένων, θα πρέπει να ανοίγουμε ή να δημιουργούμε το αρχείο dedomena.txt και να εισάγουμε να δεδομένα που θέλουμε. Η κωδικοποίηση των δεδομένων για τη μέθοδο RPROP παρουσιάζεται στον πίνακα 1 και για τις υπόλοιπες μεθόδους στον πίνακα 2.

| **Επιλογές εισαγωγής δεδομένων για την μέθοδο RPROP** | | |
|---|---|---|
| Περιγραφή | Σύμβολο | Τιμές |
| Συνάρτηση | - | r → Rosenbrock<br>b → Broyden tridiagonal<br>v → Variably dimensioned<br>n → Nazareth trigonometric<br>z → Zackharov<br>t → Trigonometric<br>d → Dixon & Price |
| Μέθοδος | - | r |
| Μέγεθος | N | αριθμητική |
| Ακρίβεια | ||g|| | αριθμητική |
| Άνω όριο διόρθωσης | $c_{max}$ | αριθμητική |
| Κάτω όριο διόρθωσης | $c_{min}$ | αριθμητική |

**Πίνακας 1**

Έτσι, για παράδειγμα, αν θέλουμε να τρέξουμε δοκιμή με τη συνάρτηση Zackharov, με τη μέθοδο RPROP, για μέγεθος προβλήματος 100 συντελεστών, με ακρίβεια 0,0001, πάνω όριο διόρθωσης 1000 και κάτω όριο διόρθωσης 0, θα πρέπει να γράψουμε στο αρχείο dedomena.txt τα παρακάτω:

```
z
r
100
0.0001
```

1000
0

| Επιλογές εισαγωγής δεδομένων για τις υπόλοιπες μεθόδους | | |
|---|---|---|
| Περιγραφή | Σύμβολο | Τιμές |
| Συνάρτηση | - | r → Rosenbrock<br>b → Broyden tridiagonal<br>v → Variably dimensioned<br>n → Nazareth trigonometric<br>z → Zackharov<br>t → Trigonometric<br>d → Dixon & Price |
| Μέθοδος | - | s → Steepest Descent<br>d → DFP<br>b → BFGS<br>f → Fletcher Reeves<br>p → Polak Ribiere |
| Μέγεθος | n | αριθμητική |
| Ακρίβεια | ||g|| | αριθμητική |
| Κανονικοποίηση | - | y ή n |
| Μέθοδος αναζήτησης επί γραμμής | - | q → τετραγωνική παρεμβολή<br>c → κυβική παρεμβολή<br>g → χρυσή τομή |
| Επανεκκίνηση | $c_{min}$ | Αν όχι, τότε 0<br>Αν ναι, την τιμή της επανάληψης που θα γίνεται επανεκκίνηση |
| Μέγιστος αριθμός επαναλήψεων αλγορίθμου αναζήτησης επί γραμμής | max iter | max iter>0 |
| Παράμετρος αλγορίθμου αναζήτησης επί γραμμής σ (σχέση 1.36) | σ | $0<\sigma<1$ |
| Παράμετρος φάσης εντοπισμού της αγκύλης, $\tau_1$ (σχέση στο **Βήμα 5** υπ. 1.4.1) | $\tau_1$ | $\tau_1 \geq 1$ |
| Παράμετρος χρήσης τομής, c (χρησιμοποιείται μόνο αν επιλεγεί η μέθοδος, σχέση) | c | $0.5<c< 1$ |
| Παράμετρος συνθήκης ικανής αρνητικότητας (σχέση 1.45) | B | B>0 |

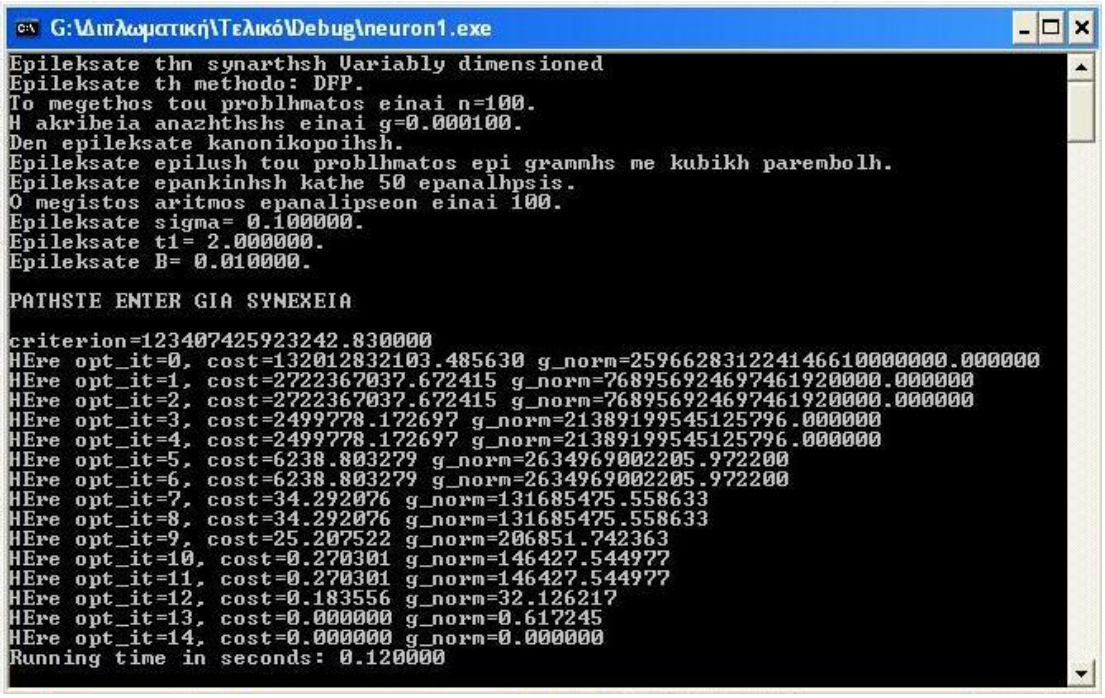
**Πίνακας 2**

Έτσι, για παράδειγμα, αν θέλουμε να τρέξουμε δοκιμή με τη συνάρτηση Variably dimensioned, με τη μέθοδο DFP, για μέγεθος προβλήματος 100 συντελεστών, με ακρίβεια 0,0001, χωρίς κανονικοποίηση, κυβική παρεμβολή για τη μέθοδο αναζήτησης επί γραμμής, επανεκκίνηση κάθε 50 επαναλήψεις, μέγιστο αριθμό επαναλήψεων της αναζήτησης επί γραμμής 100, σ ίσο με 0,1,

$\tau_1$ ίσο με 2 και Β ίσο με 0,01 θα πρέπει να γράψουμε στο αρχείο dedomena.txt τα παρακάτω:

v
d
100
0.0001
n
c
50
100
0.1
2
0.01

Έχοντας εισάγει τις επιθυμητές τιμές στο αρχείο ,τρέχουμε το πρόγραμμα επιλέγοντας το αρχείο *.exe. Μας εμφανίζεται μια οθόνη επιβεβαίωσης των επιλογών και πατώντας Enter αρχίζει η ελαχιστοποίηση της συνάρτησης. Στην παρακάτω εικόνα φαίνεται το προηγούμενο παράδειγμα.




**Εικόνα 1**

## Υπολογιστικός κώδικας

Ο κώδικας χωρίζεται σε τέσσερα βασικά αρχεία τα οποία συνδέονται μεταξύ τους με αρχεία *.h.

Το **πρώτο** αρχείο είναι το **neuron1** και περιέχει συναρτήσεις εισαγωγής και εκτυπώσεις δεδομένων, καθώς και τη main συνάρτηση του προγράμματος. Αναλυτικότερα οι συναρτήσεις που υπάρχουν στο neuron περιγράφονται στον πίνακα 3.

| **Συναρτήσεις** | **Περιγραφή** |
|---|---|
| void screen(void) | Συνάρτηση εισαγωγής δεδομένων και εκτύπωσης τους στην οθόνη |
| void allocate_memories(void) | Συνάρτηση δυναμικής δέσμευσης μνήμης |
| void read_weight_max_min(void) | Συνάρτηση δήλωσης ακραίων τιμών για τους συντελεστές $x_i$. |
| void read_initial_weights(void) | Συνάρτηση δήλωσης αρχικών τιμών των συντελεστών $x_i$. |
| void keep_optimum(void) | Συνάρτηση υπολογισμού και αποθήκευσης της τιμής της αντικειμενικής συνάρτησης για την εκάστοτε επανάληψη. |
| main() | Η κύρια συνάρτηση του κώδικα. Από εδώ θα αρχίσει να τρέχει ο κώδικας και έτσι από εδώ καλούνται όλες οι συναρτήσεις για τη βελτιστοποίηση των συναρτήσεων. |

**Πίνακας 3**

Το **δεύτερο** αρχείο είναι το **functions1** και περιέχει τον κώδικα των αντικειμενικών συναρτήσεων προς βελτιστοποίηση και των συναρτήσεων κλίσης. Πιο συγκεκριμένα περιέχει τις συναρτήσεις του πίνακα 4.

| **Συναρτήσεις** | **Περιγραφή** |
|---|---|
| double calculate_cost_criterion (double *weight1) | Συνάρτηση υπολογισμού της αντικειμενικής συνάρτησης για συγκεκριμένα σημεία. |
| void calculate_gradient (double *weight1, double *gradient1) | Συνάρτηση υπολογισμού της κλίσης της αντικειμενικής συνάρτησης για συγκεκριμένα σημεία. |

**Πίνακας 4**

Το **τρίτο** αρχείο είναι το **opt_util1** και περιέχει συναρτήσεις του κάθε βήματος βελτιστοποίησης για όλες τις μεθόδους και για όλες τις φάσεις. Πιο συγκριμένα οι συναρτήσεις περιγράφονται στον πίνακα 5.

| **Συναρτήσεις** | **Περιγραφή** |
|---|---|
| void constrain_weights() | Συνάρτηση έλεγχου των τιμών των συντελεστών της αντικειμενικής συνάρτησης. |
| void constraint_gradient() | Συνάρτηση έλεγχου των μερικών παραγώγων της αντικειμενικής συνάρτησης. |
| void initialise_search_direction (double *gr) | Βοηθητική συνάρτηση που δίνει στην κατεύθυνση αναζήτησης την τιμή της κλίσης της αντικειμενικής συνάρτησης. |
| void calculate_norms() | Συνάρτηση υπολογισμού της νόρμας κλίσης. |
| void search_direction_function() | Συνάρτηση υπολογισμού κατεύθυνσης αναζήτησης. |
| void line_optimisation() | Συνάρτηση υπολογισμού αναζήτησης επί γραμμής. |
| void rprop() | Συνάρτηση βελτιστοποίησης με τη μέθοδο RPROP. |

**Πίνακας 5**

Το **τέταρτο** και τελευταίο αρχείο είναι το **math_utils1** και περιέχει βοηθητικές μαθηματικές συναρτήσεις που χρειάζονται σε διάφορα βήματα της ελαχιστοποίησης. Πιο συγκεκριμένα οι συναρτήσεις περιγράφονται στον πινάκα 6.

| **Συναρτήσεις** | **Περιγραφή** |
|---|---|
| void copy_matrix(double *dest,double *src,int n) | Συνάρτηση αντιγραφής ενός διανύσματος σε δεύτερο διάνυσμα. |
| double scalar_product(double *mtr1,double *mtr2,int n) | Συνάρτηση πολλαπλασιασμού ανάστροφου διανύσματος με δεύτερο διάνυσμα. |
| void sub_matrix(double *dest,double *mtr1,double *mtr2,int n) | Συνάρτηση αφαίρεσης διανυσμάτων. |
| void scalar_matrix(double *dest,double *mtr,double scal,int n) | Συνάρτηση πολλαπλασιασμού διανύσματος με ένα συντελεστή. |
| void add_matrix(double *dest,double *mtr1,double *mtr2,int n) | Συνάρτηση πρόσθεσης διανυσμάτων. |
| void mult_vecsym(double *a, double *b, double *l,int n) | Συνάρτηση πολλαπλασιασμού διανύσματος με άλλο ανάστροφο διάνυσμα. |
| void mult_complex(double *a, double *b, double *c,int n) | Συνάρτηση πολλαπλασιασμού πίνακα με διάνυσμα. |
| void mult_complex1(double *k, double *a, double *l,int n) | Συνάρτηση πολλαπλασιασμού διανύσματος με πίνακα. |
| void mult_scalar(double *a,double x,double *b,int n) | Συνάρτηση πολλαπλασιασμού ενός συντελεστή με έναν πίνακα. |
| void unit_matrix(double *a,int n) | Συνάρτηση δημιουργίας μοναδιαίου πίνακα. |
| void mult_sym(double *a,double x,int n) | Συνάρτηση δημιουργίας πίνακα που περιέχει σε όλες τις θέσεις του ένα συντελεστή. |
| void mult_vector2(double *z,double *k,double *C,int n) | Συνάρτηση πολλαπλασιασμού διανύσματος με πίνακα. |
| void sigr(double *a, double *b, double *c, double d1, double d2, int nr) | Συνάρτηση 1.56 της μεθόδου επίλυσης RPROP. |
| void sigr2(double *a, double *b, double *c, int nr) | Συνάρτηση 1.57 της μεθόδου επίλυσης RPROP. |
| void input_vector1(double *a, int n) | Συνάρτηση εισαγωγής μίας τιμής σε όλες τις θέσεις ενός διανύσματος. |
| void ypol_a(double *v, double *ypol,int n) | Συνάρτηση κλιμακοποίησης (κανονικοποίησης). |
| void unit_matrix1(double *a,double *b,int n) | Συνάρτηση εισαγωγής τιμών στη διαγώνιο ενός πίνακα και μηδενισμό όλων των άλλων τιμών. |

**Πίνακας 6**

## Παράδειγμα εισαγωγής συνάρτησης προς λύση

Ένας από τους σκοπούς δημιουργίας του κώδικα και προγράμματος βελτιστοποίησης μη-γραμμικών συναρτήσεων είναι ο χρήστης στο μέλλον να δύναται να εισάγει νέες συναρτήσεις στο πρόγραμμα και να τις βελτιστοποιεί.

Για την εισαγωγή νέων συναρτήσεων χρειάζεται η εισαγωγή τριών δεδομένων.

- Εισαγωγή της αντικειμενικής συνάρτησης, προγραμματισμένη έτσι ώστε να μπορεί να τρέχει για πολλές μεταβλητές
- Εισαγωγή του διανύσματος κλίσης, προγραμματισμένο έτσι ώστε να μπορεί να τρέχει για πολλές μεταβλητές.
- Εισαγωγή αρχικών σημείων.

Πιο συγκεκριμένα παρακάτω θα δοθεί ο τρόπος εισαγωγής των στοιχείων για τη συνάρτηση Dixon & Price.

## Εισαγωγή δεδομένων

Θυμίζουμε πως η αλγεβρική μορφή της Dixon & Price function είναι:

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$$

όπου n ο αριθμός των μεταβλητών.

Έτσι το διάνυσμα μερικών παράγωγων της συνάρτησης που χρειάζονται για τον υπολογισμό της κλίσης είναι:

για $x_1$,

$$\partial f(x) / \partial x_1 = 2(x_1 - 1) - 4(2x_2^2 - x_1)$$

για $x_i$ όπου i=2…n-1,

$$\partial f(x) / \partial x_i = 2i(2x_i^2 - x_{i-1})4x_i - 2(i+1)(2x_{i+1}^2 - x_i)$$

και για $x_n$

$$\partial f(x) / \partial x_n = 2n(2x_n^2 - x_{n-1})4x_n$$

Για τον προγραμματισμό της αντικειμενικής συνάρτησής ανοίγουμε το αρχείο **functions1** και στη συνάρτηση **double calculate_cost_criterion (double *weight1)** προγραμματίζουμε με βάση την αλγεβρική της μορφή. Παρακάτω παρουσιάζεται προγραμματισμένη σε γλώσσα C.

```
double calculate_cost_criterion(double *weight1)
{
        int i;
        double c_crit;
        c_crit=0.0;
        c_crit+=sq(*weight1-1);
        weight1++;
        for (i=2;i<=n;i++)
        {
```

```
            c_crit+=i*sq(2*sq(*weight1)-*(weight1-1));
            weight1++;
        }
        return c_crit;
}
```

Για τον προγραμματισμό των μερικών παραγώγων ανοίγουμε το αρχείο **functions1** και στη συνάρτηση **void calculate_gradient (double *weight1, double *gradient1)** προγραμματίζουμε με βάση την αλγεβρική τους μορφή. Παρακάτω παρουσιάζονται προγραμματισμένες σε γλώσσα C.

```
void calculate_gradient(double *weight1,double *gradient1)
{
        int i;
        *gradient1++=2*(*weight1-1)-4*(2*sq(*(weight1+1))-*weight1);
        weight1++;
        for (i=2;i<n;i++)
        {
            *gradient1++=2*i*(2*sq(*weight1)-*(weight1-1))*4*(*weight1)-
2*(i+1)*(2*sq(*(weight1+1))-*weight1);
            weight1++;
        }
        *gradient1++=2*n*(2*sq(*weight1)-*(weight1-1))*4*(*weight1);
}
```

Για την εισαγωγή και τον προγραμματισμό των αρχικών σημείων ανοίγουμε το αρχείο **neuron1** και στη συνάρτηση void **read_initial_weights(void)** προγραμματίζουμε τα επιθυμητά αρχικά σημεία. Στο συγκριμένο παράδειγμα τα αρχικά σημεία είναι $x_i$=0,6. Παρακάτω παρουσιάζονται προγραμματισμένα σε γλώσσα C.

```
void  read_initial_weights(void)
{
         int i;
        for (i=0;i<n;i++)
        {
            weight[i] =0.6;
        }
}
```

## Κώδικας σε C

Ακολουθεί εκτυπωμένος ο υπολογιστικός κώδικας σε γλώσσα προγραμματισμού C.

### neuron1.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "neuron1.h"
#include "opt_util1.h"
#include "functions1.h"
#include "math_utils1.h"
FILE *fp,*fp1,*fp2,*fp4,*fp5,*fp3,*fp6;

int method; /* search direction method */
int method9;
int method_prev;
int method10;
int method1; /* interpolation in the sectioning phase */
int method11; /*scalling*/
int method12;
int function;
double gg; /* accuracy of the convergence */
int max_iter; /* maximum number of iterations in line optimisation */
int n; /* the number of variables*/
int mi_index; /* index for convergence test */
int opt_it;
int restart;
int indicator1;
int indicator2;
int indicator4;
int pseudo4;
int j3;
int keeper;
int pseudo100;
double sigma2;
double c_value;
double t1;
double d_max;
double d_min;
double g_norm,opt_cost;
double s_norm;
double sg_norm;
double neg;
double *weight; /* weights to be optimised */
double *opt_weight; /* optimum weights */
double *weight_0; /* auxiliary matrix for weights */
double *max_weight; /* maximum weights */
double *min_weight; /* minimum weights */
double *gradient; /* unconstrained gradient */
double *gradienta;
double *gradienta_0;
double *c_gradient; /* constrained gradient */
double *c_gradient_0; /* constrained gradient i-1 */
double *search_direction; /* search direction */
double *search_direction_0; /* previous search direction */
double criterion; /* value of the cost criterion */
double *inputs; /* inputs trent */
double *outputs; /* output data */
double *f_cost;/* final cost */
double *opt_final_cost; /* optimum final costs */
double *output_function; /* neuron output */
double *receive_function; /* input to a neuron */
double *final_cost; /* final cost vector */
```

```
double *aux_matrix; /* auxiliary matrix */
double *incr1; /* auxiliary matrix */
double deltaJ; /* for initial bracketing */
double *output_pair; /* logistic function of neurons */
double *H;       /*Matrix H*/
double *A;
double *HG;      /*The product HG*/
double *G1H;     /*The matrix G*H */
double *inter;
double *interp;
double *inter1;
double *delta;   /*The vector (x-x_previous)*/
double *gamma;   /*The vector (g-g_previous)*/
double *nominator;/*An Intermediate matrix*/
double  denominator; /*An Intermediate value*/
double  denominator1; /*An Intermediate value*/
double  multiplier; /*An Intermediate value*/
double *quotient; /*An Intermediate matrix*/
double *DG1H;
double *HGD1;
double sigma;
double beta;
double vy;
double by;
double bg;
double wy;
double *z;
double *v_h;
double *w_h;
double *z_h;
double *v_h1;
double *w_h1;
double *s_h;
double *b_h;
double *b_h1;
double ra;
double rb;
double rc;
double vg;
double wg;
double *a_step;
double *a_step_old;
double *c_gradient_1;
double *ga;
double *gpr;
double *delta_x;
double *delta_x_prev;
double *gpn;
double *delta_xp;
double *delta_x_prevp;
clock_t ticks, ticks_begin, ticks2;
double cpu_time;
int restartp;

void screen(void)
{
        int c;
        char methodos[1],function_s[1];
        fp6=fopen("dedomena.txt","r");
        fgets(function_s,2,fp6);
        if(function_s[0]=='r')
        {
                function=0;
                printf("Epileksate thn synarthsh Rosenbrock\n");
        }
        else if(function_s[0]=='b')
        {
                function=1;
```

```
			printf("Epileksate thn synarthsh Broyden tridiagonal\n");
		}
		else if(function_s[0]=='v')
		{
			function=2;
			printf("Epileksate thn synarthsh Variably dimensioned\n");
		}
		else if(function_s[0]=='n')
		{
			function=3;
			printf("Epileksate thn synarthsh Nazareth trigonometric\n");
		}
		else if(function_s[0]=='z')
		{
			function=4;
			printf("Epileksate thn synarthsh Zakharov\n");
		}
		else if(function_s[0]=='t')
		{
			function=5;
			printf("Epileksate thn synarthsh Trigonometric\n");
		}
		else if(function_s[0]=='d')
		{
			function=6;
			printf("Epileksate thn synarthsh Dixon & Price\n");
		}
		fscanf(fp6,"\n");
		fgets(methodos,2,fp6);
		if (methodos[0]=='r')
		{
			printf("Epileksate th methodo: RPROP\n.");
			method=0;
			fscanf(fp6,"%d",&n);
			printf("To megethos tou problhmatos einai n=%d.\n",n);
			fscanf(fp6,"%lf",&gg);
			printf("H akribeia anazhthshs einai g=%f.\n",gg);
			max_iter=100;
			fscanf(fp6,"%lf",&d_max);
			printf("To pano orio ths enhmeroshs einai d_max=%f.\n",d_max);
			fscanf(fp6,"%lf",&d_min);
			printf("To kato orio ths enhmeroshs einai d_mix=%f.\n",d_min);
			printf("PATHSTE ENTER GIA SYNEXEIA\n");
			getchar();
			method10=0;
}
else
{
			method10=1;
			if(methodos[0]=='s')
			{
				printf("Epileksate th methodo: Steepest Descent.\n");
				method=1;
			}
			else if(methodos[0]=='d')
			{
				printf("Epileksate th methodo: DFP.\n");
				method=2;
			}
			else if(methodos[0]=='b')
			{
				printf("Epileksate th methodo: BFGS.\n");
				method=3;
			}
			else if(methodos[0]=='f')
			{
				printf("Epileksate th methodo: Fletcher Reeves.\n");
				method=4;
```

```
                }
                else if(methodos[0]=='p')
                {
                        printf("Epileksate th methodo: Polak Ribiere.\n");
                        method=5;
                }
                fscanf(fp6,"%d",&n);
                printf("To megethos tou problhmatos einai n=%d.\n",n);
                fscanf(fp6,"%lf\n",&gg);
                printf("H akribeia anazhthshs einai g=%f.\n",gg);
                fgets(methodos,2,fp6);
                if(methodos[0]=='y')
                {
                        printf("Epileksate kanonikopoihsh.\n");
                        method11=1;
                        method12=1;
                }
                else if(methodos[0]=='n')
                {
                        printf("Den epileksate kanonikopoihsh.\n");
                        method11=0;
                        method12=0;
                }
                fscanf(fp6,"\n");
                fgets(methodos,2,fp6);
                if(methodos[0]=='q')
                {
                        printf("Epileksate epilush tou problhmatos epi grammhs me tetragonikh
parembolh.\n");
                        method1=1;
                }
                else if(methodos[0]=='c')
                {
                        printf("Epileksate epilush tou problhmatos epi grammhs me kubikh
parembolh.\n");
                        method1=2;
                }
                else if(methodos[0]=='g')
                {
                        printf("Epileksate epilush tou problhmatos epi grammhs me xrysh tomh.\n");
                        method1=3;
                }
                fscanf(fp6,"%d",&restart);
                if(restart!=0)
                {
                        printf("Epileksate epankinhsh kathe %d epanalhpsis.\n",restart);
                }
                method9=1;
                fscanf(fp6,"%d",&max_iter);
                printf("O megistos aritmos epanalipseon einai %d.\n",max_iter);
                fscanf(fp6,"%lf",&sigma);
                printf("Epileksate sigma= %f.\n",sigma);
                fscanf(fp6,"%lf",&t1);
                printf("Epileksate t1= %f.\n",t1);
                if (methodos[0]=='g')
                {
                        fscanf(fp6,"%lf",&c_value);
                        printf("Epileksate c= %f.\n",c_value);
                }
                fscanf(fp6,"%lf",&sigma2);
                printf("Epileksate B= %f.\n\n",sigma2);
                printf("PATHSTE ENTER GIA SYNEXEIA\n");
                getchar();
        }
        fclose(fp6);
}

void allocate_memories(void)
```

```
{
        weight=(double *)malloc((n)*sizeof(double));
        opt_weight=(double *)malloc((n)*sizeof(double));
        weight_0=(double *)malloc((n)*sizeof(double));
        gradient=(double *)malloc((n)*sizeof(double));
        c_gradient=(double *)malloc((n)*sizeof(double));
        c_gradient_0=(double *)malloc((n)*sizeof(double));
        c_gradient_1=(double *)malloc((n)*sizeof(double));
        gradienta=(double *)malloc((n)*sizeof(double));
        gradienta_0=(double *)malloc((n)*sizeof(double));
        search_direction=(double *)malloc((n)*sizeof(double));
        search_direction_0=(double *)malloc((n)*sizeof(double));
        max_weight=(double *)malloc((n)*sizeof(double));
        min_weight=(double *)malloc((n)*sizeof(double));
        output_function=(double *)malloc((n)*sizeof(double));
        aux_matrix=(double *)malloc((n)*sizeof(double));
        incr1=(double *)malloc((n)*sizeof(double));
        output_pair=(double *)malloc((n)*sizeof(double));
        outputs=(double *)malloc(n*sizeof(double));
        inputs=(double *)malloc(n*sizeof(double));
        f_cost=(double *)malloc(n*sizeof(double));
        opt_final_cost=(double *)malloc(n*sizeof(double));
        receive_function=(double *)malloc((n)*sizeof(double));
        final_cost=(double *)malloc(n*sizeof(double));
        A = (double *) malloc(n*n * sizeof(double));
        if((method==2)^(method==3))
        {
                H = (double *) malloc(n*n * sizeof(double));
                G1H=(double *) malloc(n * sizeof(double));
                HG=(double *) malloc(n * sizeof(double));
                inter=(double *) malloc(n * sizeof(double));
                interp=(double *) malloc(n * sizeof(double));
                delta=(double *) malloc(n * sizeof(double));
                gamma=(double *) malloc(n * sizeof(double));
                nominator=(double *) malloc(n*n * sizeof(double));
                quotient=(double *) malloc(n*n * sizeof(double));
                inter1=(double *) malloc(n*n * sizeof(double));
                HGD1=(double *) malloc(n*n * sizeof(double));
                DG1H=(double *) malloc(n*n * sizeof(double));
                z=(double *) malloc(n * sizeof(double));
                w_h=(double *) malloc(n * sizeof(double));
                v_h=(double *) malloc(n * sizeof(double));
                z_h=(double *) malloc(n * sizeof(double));
                v_h1=(double *) malloc(n * sizeof(double));
                w_h1=(double *) malloc(n * sizeof(double));
                s_h=(double *) malloc(n * sizeof(double));
                b_h=(double *) malloc(n * sizeof(double));
                b_h1=(double *) malloc(n * sizeof(double));
  }
        if(method10==0)
        {
                a_step=(double *) malloc(n * sizeof(double));
                a_step_old=(double *) malloc(n * sizeof(double));
                ga=(double *) malloc(n * sizeof(double));
                gpr=(double *) malloc(n * sizeof(double));
                delta_x=(double *) malloc(n * sizeof(double));
                delta_x_prev=(double *) malloc(n * sizeof(double));
                gpn=(double *) malloc(n * sizeof(double));
                delta_xp=(double *) malloc(n * sizeof(double));
                delta_x_prevp=(double *) malloc(n * sizeof(double));
        }

}

void  read_weight_max_min(void)
{
        int i;
        for(i=0;i<n;i++)
```

```
                max_weight[i] = 100000000.8;
        for(i=0;i<n;i++)
                min_weight[i] = -100000000.0;
}

void  read_initial_weights(void)
{
        if(function==0)
        {
                int i;
                for (i=0;i<n;i++)
                {
                        weight[i] =0;
                }
        }
        else if(function==1)
        {
                int i;
                for (i=0;i<n;i++)
                {
                        weight[i] =-1;
                }
        }
        else if(function==2)
        {
                int i;
                double m,j;
                m=n;
                j=0;
                for (i=0;i<n;i++)
                {
                        weight[i] =1-(j/m);
                        j++;
                }
        }
        else if(function==3)
        {
                int i;
                double m;
                m=n;
                for (i=0;i<n;i++)
                {
                        weight[i] =1/m;
                }
        }
        else if(function==4)
        {
                int i;
                for (i=0;i<n;i++)
                {
                        if(i%2==0)
                                weight[i] =10;
                        else
                                weight[i]=9;
                }
        }
        else if(function==5)
        {
                int i;
                double m;
                m=(double) n;
                for (i=0;i<n;i++)
                {
                        weight[i] =1/m;
                }
        }
        else if(function==6)
        {
```

```
                        int i;
                        double a;
                        a=0.6;
                        weight[0]=0.6;
                        for (i=1;i<n;i++)
                        {
                                weight[i] =a;
                        }
                }
}

void keep_optimum(void)
{
                int i;
                copy_matrix(opt_weight,weight,n);
                for(i=0;i<n;i++)
                        opt_final_cost[i]=f_cost[i];
                opt_cost=criterion;
}

/*void write_optimum(void)
{
 char name[15],name1[15];
 int i;
 for(i=0;i<15;i++)
  name[i]='\0';
 for(i=0;i<15;i++)
  name1[i]='\0';
 sprintf(name,"optimum.txt");
 fp=fopen(name,"a+");
 fprintf(fp,"Optimum cost: %.5f\topt_it=%d\n",opt_cost,opt_it);
 fprintf(fp,"Stage \t Origin \t Destination \t Optimum Weight\n");
 for(i=0;i<n;i++)
   {
    fprintf(fp,"%d \t %.5f\n",i,opt_weight[i]);
   }
 fprintf(fp,"Pair \t Output \t Data \t Difference\n");
 for(i=0;i<n;i++)
 {
  fprintf(fp,"%d \t %.5f \t %.5f \t %.5f\n", i,opt_final_cost[i],outputs[i],opt_final_cost[i]-outputs[i]);
  fprintf(fp,"-------------------------------------------------------------------------------\n");
 }
 fprintf(fp,"HEre opt_it=%d, cost=%f g_norm=%f\n",opt_it,criterion,g_norm);
  fprintf(fp,"-------------------------------------\n");
     for (i=0;i<n;i++)
   fprintf(fp,"weight=%f\n",weight[i]);
   fprintf(fp,"-------------------------------------\n");
   fprintf(fp,"gnorm=%f\n",g_norm);
                fprintf(fp,"\nthe  new function value ");
                fprintf(fp,"%.10lf \n ",criterion);

  fclose(fp);
 }*/

main()
{
                /*fp1=fopen("apotelesmata cpu time.txt","w");
                fp4=fopen("apotelesmata J.txt","w");*/
                fp3=fopen("shmeia.txt","w");
                screen();
                if (method==1)
                {
                        fp1=fopen("St-D cpu time.txt","w");
                        fp4=fopen("St-D J.txt","w");
                }
                else if(method==2)
                {
                        if (restartp=='n')
```

```
		{
			fp1=fopen("DFP cpu time.txt","w");
			fp4=fopen("DFP J.txt","w");
		}
		else
		{
			fp1=fopen("DFP re cpu time.txt","w");
			fp4=fopen("DFP re J.txt","w");
		}
}
else if (method==3)
{
		if (restartp=='n')
		{
			fp1=fopen("BFGS cpu time.txt","w");
			fp4=fopen("BFGS J.txt","w");
		}
		else
		{
			fp1=fopen("BFGS re cpu time.txt","w");
			fp4=fopen("BFGS re J.txt","w");
		}
}
else if(method==4)
{
		if (restartp=='n')
		{
			fp1=fopen("Fl-R cpu time.txt","w");
			fp4=fopen("Fl-R J.txt","w");
		}
		else
		{
			fp1=fopen("Fl-R re cpu time.txt","w");
			fp4=fopen("Fl-R re J.txt","w");
		}
}
else if(method==5)
{
		if (restartp=='n')
		{
			fp1=fopen("P-R cpu time.txt","w");
			fp4=fopen("P-R J.txt","w");
		}
		else
		{
			fp1=fopen("P-R re cpu time.txt","w");
			fp4=fopen("P-R re J.txt","w");
		}
}
else if(method==0)
{
			fp1=fopen("cpu time.txt","w");
			fp4=fopen("J.txt","w");
}
allocate_memories();
char name[15];
int i;
for(i=0;i<15;i++)
		name[i]='\0';
ticks = clock();
ticks_begin=clock();
read_weight_max_min();
/*****************************/
/* initialisation phase      */
/*****************************/
read_initial_weights();
constrain_weights();
if(method11 == 1)
```

```
{
        ypol_a(weight,incr1,n);
        unit_matrix1(A,incr1,n);
}
criterion=calculate_cost_criterion(weight);
printf("criterion=%f\n",criterion);
calculate_gradient(weight,gradient);
initialise_search_direction(gradient);
constraint_gradient();
calculate_norms();
/*calculate_m();*/
opt_it=0; /* initial value of optimisation iteration */
/*fprintf(fp3,"%f %f\n",weight[0],weight[1]);*/
fprintf(fp4,"%f\n",criterion);
if(method10==0)
{
        rprop();
}
else
{
        initialise_search_direction(c_gradient);
        copy_matrix(c_gradient_0,c_gradient,n);
        mi_index=1;
        method_prev=method;
        /*****************************/
        /* optimisation loop         */
        /*****************************/
        for (i=0;i<n;i++)
        {
                fprintf(fp3,"%f\t",gradient[i]);
        }
        fprintf(fp3,"\n ");
        do
        {
                if(indicator2!=1)
                {
                        method=method_prev;
                }
                pseudo100=pseudo100+1;
                if (pseudo100==restart)
                {
                        indicator1=1;
                        beta=0;
                        method=1;
                        pseudo100=0;
                        if(method11 == 1)
                        {
                                ypol_a(weight,incr1,n);
                                unit_matrix1(A,incr1,n);
                                if (((method==2)^(method==3))||(method9 == 1))
                                {
                                        copy_matrix(H,A,n*n);
                                }
                        }
                }
                search_direction_function();
                s_norm=scalar_product(search_direction,search_direction,n);
                sg_norm=scalar_product(search_direction,c_gradient,n);
                neg=-1*sigma2*sqrt(s_norm*g_norm);
                /*printf("%f %f \n",sg_norm,neg);*/
                if (sg_norm>=neg)
                {
                        /*printf(" neg\n");
                        getchar();
                        getchar();*/
                        keeper=method_prev;
                        indicator1=1;
                        indicator2=1;
```

```
                                        method=1;
                                        beta=0;
                                    if(method11 == 1)
                                        {
                                                ypol_a(weight,incr1,n);
                                                unit_matrix1(A,incr1,n);
                                                if (((method==2)^(method==3))||(method9 == 1))
                                                {
                                                        copy_matrix(H,A,n*n);
                                                }
                                        }
                                        calculate_norms();
                                    if(deltaJ<0.0)
                                                break;
                                    else
                                                keep_optimum();
                                        printf("HEre opt_it=%d, cost=%f
g_norm=%f\n",opt_it,criterion,g_norm);
                                        opt_it++;
                                }
                                else
                                {
                                        /*printf("%d\n",method);*/
                                        method=method_prev;
                                        /*printf("%d\n",method);
                                        getchar();
                                        getchar();*/
                                        line_optimisation();
                                        calculate_norms();
                                        if(deltaJ<0.0)
                                                break;
                                        else
                                                keep_optimum();
                                        printf("HEre opt_it=%d, cost=%f
g_norm=%f\n",opt_it,criterion,g_norm);
                                        opt_it++;
                                }
                                ticks2 = clock() - ticks_begin;
                                cpu_time = (double) ticks2/CLOCKS_PER_SEC;
                                fprintf(fp1,"%f\n",cpu_time);
                                fprintf(fp4,"%f\n",criterion);
                                for (i=0;i<n;i++)
                                {
                                        fprintf(fp3,"%f\t",gradient[i]);
                                }
                                fprintf(fp3,"\n ");
                        }
                        while(deltaJ>0.00000001 && g_norm>gg && mi_index!=0);
                        fclose(fp4);
        }
/*      write_optimum();*/
        /*for (i=0;i<n;i++)
        {
                fprintf(fp3,"%f\n ",gradient[i]);
        }*/
        fclose(fp3);
        ticks = clock() - ticks;
        cpu_time = (double) ticks/CLOCKS_PER_SEC;
        printf("Running time in seconds: ");
        printf("%f\n", cpu_time);
        fprintf(fp1,"%f\n",cpu_time);
        fclose(fp1);
        getchar();
        getchar();
} /* end of main */
```

## neuron1.h

```
#include <time.h>
```

```
#define e_value 0.00000001

void screen(void);

extern int method; /* search direction method */
extern int method9;
extern int method_prev;
extern int method10;
extern int method1; /* interpolation in the sectioning phase */
extern int method11; /*scalling*/
extern int method12;
extern int function;
extern double gg; /* accuracy of the convergence */
extern int max_iter; /* maximum number of iterations in line optimisation */
extern int n; /*number of variables */
extern int mi_index; /* index for convergence test */
extern int opt_it;
extern int keeper;
extern int restart;
extern int indicator1;
extern int indicator2;
extern int indicator4;
extern int pseudo4;
extern int j3;
extern int pseudo100;
extern double sigma2;
extern double c_value;
extern double t1;
extern double d_max;
extern double d_min;
extern double g_norm;
extern double sg_norm;
extern double s_norm;
extern double neg;
extern double *weight; /* weights to be optimised */
extern double *weight_0; /* auxiliary matrix for weights */
extern double *max_weight; /* maximum weights */
extern double *min_weight; /* minimum weights */
extern double *gradient; /* unconstrained gradient */
extern double *gradienta;
extern double *gradienta_0;
extern double *c_gradient_1;
extern double *c_gradient; /* constrained gradient */
extern double *c_gradient_0; /* constrained gradient i-1 */
extern double *search_direction; /* search direction */
extern double *search_direction_0; /* previous search direction */
extern double criterion; /* value of the cost criterion */
extern double *inputs; /* inputs data */
extern double *outputs; /* output data */
extern double *f_cost;/* final cost */
extern double *output_function; /* neuron output */
extern double *receive_function; /* input to a neuron */
extern double *final_cost; /* final cost vector */
extern double *aux_matrix; /* auxiliary matrix */
extern double *incr1; /* auxiliary matrix */
extern double deltaJ; /* for initial bracketing */
extern double *output_pair; /* logistic function of neurons */
extern double *H;       /*Matrix H*/
extern double *A;
extern double *HG;      /*The product HG*/
extern double *G1H;     /*The matrix G*H */
extern double *inter;
extern double *interp;
extern double *inter1;
extern double *delta;   /*The vector (x-x_previous)*/
extern double *gamma;   /*The vector (g-g_previous)*/
extern double *nominator;/*An Intermediate matrix*/
extern double  denominator; /*An Intermediate value*/
```

```
extern double  denominator1; /*An Intermediate value*/
extern double  multiplier; /*An Intermediate value*/
extern double *quotient; /*An Intermediate matrix*/
extern double *DG1H;
extern double *HGD1;
extern double sigma;
extern double beta;
extern double vy;
extern double by;
extern double bg;
extern double wy;
extern double *z;
extern double *v_h;
extern double *w_h;
extern double *z_h;
extern double *v_h1;
extern double *w_h1;
extern double *s_h;
extern double *b_h;
extern double *b_h1;
extern double ra;
extern double rb;
extern double rc;
extern double vg;
extern double wg;
extern double *a_step;
extern double *a_step_old;
extern double *ga;
extern double *gpr;
extern double *delta_x;
extern double *delta_x_prev;
extern double *gpn;
extern double *delta_xp;
extern double *delta_x_prevp;
```

## functions1.cpp

```
#include "functions1.h"
#include "neuron1.h"

double sq(double a)
 { return a*a; }

double sq3(double a)
{ return a*a*a; }

double calculate_cost_criterion(double *weight1)
{
        if(function==0)
        {
                int i;
                double c_crit;
                c_crit=0.0;
                for (i=n-1;i;i--)
                {
                        c_crit += sq(1-*weight1)+ 100*sq(*(weight1+1)-sq(*weight1));
                        weight1++;
                }
                return c_crit;
        }
        else if(function==1)
        {
                int j;
                double c_crit;
                c_crit=0.0;
                c_crit+=sq((3-2**weight1)**weight1-2**(weight1+1)+1);
                weight1++;
                for(j=1;j<=n-2;j++)
                {
```

```
                c_crit+=sq((3-2**weight1)**weight1-2**(weight1+1)-*(weight1-1)+1);
                weight1++;
        }
        c_crit+=sq((3-2**weight1)**weight1-*(weight1-1)+1);
        return c_crit;
}
else if(function==2)
{
        int j;
        double c_crit, c_crit_help1,c_crit_help2;
        c_crit_help1=0.0;
        c_crit_help2=0.0;
        c_crit=0.0;
        for(j=1;j<=n;j++)
        {
                c_crit_help1+=sq(*weight1-1);
                c_crit_help2+=j*(*weight1-1);
                weight1++;
        }
        c_crit=c_crit_help1+sq(c_crit_help2)+sq(sq(c_crit_help2));
        return c_crit;
}
else if(function==3)
{
        int i,j;
        double c_crit,a,b,c_crit_sum1,*re;
        c_crit=0.0;
        re=weight1;
        for(i=1;i<=n;i++)
        {
                weight1=re;
                c_crit_sum1=0.0;
                for(j=1;j<=n;j++)
                {
                        a=5*(1+(i%5)+(j%5));
                        b=((double)i+(double)j)/10;
                        c_crit_sum1+=a*sin(*weight1)+b*cos(*weight1);
                        weight1++;
                        /*printf("%f %f %f ",*(weight1-1),a,b);
                        getchar();*/
                }
                c_crit+=sq(n+i-c_crit_sum1);
        }
        return c_crit;
}
else if(function==4)
{
        int i;
        double c_crit,c_crit_1,c_crit_2,*weight_help;
        c_crit=0.0;
        c_crit_1=0.0;
        c_crit_2=0.0;
        weight_help=weight1;
        for (i=1;i<=n;i++)
        {
                c_crit_1 += sq(*weight1);
                weight1++;
        }
        weight1=weight_help;
        for (i=1;i<=n;i++)
        {
                c_crit_2 += 0.5*i*(*weight1);
                weight1++;
        }
        c_crit=c_crit_1+sq(c_crit_2)+sq(sq(c_crit_2));
        return c_crit;
}
else if(function==5)
```

```
	{
		int i;
		double c_crit, c_crit_help,*weight2,n2;
		weight2=weight1;
		c_crit=0.0;
		c_crit_help=0.0;
		n2=(double) n;
		for (i=1;i<=n;i++)
		{
			c_crit_help+=cos(*weight2);
			weight2++;
		}
		for(i=1;i<=n;i++)
		{
			c_crit+=sq(n2-c_crit_help+i*(1-cos(*weight1))-sin(*weight1));
			weight1++;
		}
		return c_crit;
	}
	else if(function==6)
	{
		int i;
		double c_crit;
		c_crit=0.0;
		c_crit+=sq(*weight1-1);
		weight1++;
		for (i=2;i<=n;i++)
		{
			c_crit+=i*sq(2*sq(*weight1)-*(weight1-1));
			weight1++;
		}
		return c_crit;
	}
}


void calculate_gradient(double *weight1,double *gradient1)
{
	if(function==0)
	{
		int i;
		*gradient1++ = -2*(1-*weight1) - 400 * (*weight1)*(*(weight1+1) - sq(*weight1));
		weight1++;
		for (i=1;i<n-1;i++)
		{
			*gradient1++ = 200 * (*weight1- sq(*(weight1-1)))-2* (1-*weight1) -400
*(*weight1) * (*(weight1+1)-sq(*weight1));
			weight1++;
		}
		*gradient1 = 200 *(*weight1-sq(*(weight1-1)));
	}
	else if(function==1)
	{
		int i;
		if (n==2)
		{
			*gradient1++=2*((3-2**weight1)**weight1-2**(weight1+1)+1)*(3-4**weight1)-
2*((3-2**(weight1+1))**(weight1+1)-*weight1+1);
			weight1++;
			*gradient1=2*((3-2**weight1)**weight1-*(weight1-1)+1)*(3-4**weight1)-4*((3-
2**(weight1-1))**(weight1-1)-2**weight1+1);
			printf("%f %f\n",*gradient1,*(gradient1-1));
		}
		else if(n==3)
		{
			*gradient1++=2*((3-2**weight1)**weight1-2**(weight1+1)+1)*(3-4**weight1)-
2*((3-2**(weight1+1))**(weight1+1)-*weight1-2**(weight1+2)+1);
			weight1++;
```

```
                        *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-2**weight1+1)+2*((3-
2**weight1)**weight1-*(weight1-1)-2**(weight1+1)+1)*(3-4**weight1)-2*((3-
2**(weight1+1))**(weight1+1)-*weight1+1);
                        weight1++;
                        *gradient1=-4*((3-2**(weight1-1))**(weight1-1)-*(weight1-2)-
2**weight1+1)+2*((3-2**weight1)**weight1-*(weight1-1)+1)*(3-4**weight1);
                }
                else if(n==4)
                {
                        *gradient1++=2*((3-2**weight1)**weight1-2**(weight1+1)+1)*(3-4**weight1)-
2*((3-2**(weight1+1))**(weight1+1)-*weight1-2**(weight1+2)+1);
                        weight1++;
                        *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-2**weight1+1)+2*((3-
2**weight1)**weight1-*(weight1-1)-2**(weight1+1)+1)*(3-4**weight1)-2*((3-
2**(weight1+1))**(weight1+1)-*weight1-2**(weight1+2)+1);
                        weight1++;
                        *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-*(weight1-2)-
2**weight1+1)+2*((3-2**weight1)**weight1-*(weight1-1)-2**(weight1+1)+1)*(3-4**weight1)-2*((3-
2**(weight1+1))**(weight1+1)-*weight1+1);
                        weight1++;
                        *gradient1=-4*((3-2**(weight1-1))**(weight1-1)-*(weight1-2)-
2**weight1+1)+2*((3-2**weight1)**weight1-*(weight1-1)+1)*(3-4**weight1);
                }
                else
                {
                        *gradient1++=2*((3-2**weight1)**weight1-2**(weight1+1)+1)*(3-4**weight1)-
2*((3-2**(weight1+1))**(weight1+1)-*weight1-2**(weight1+2)+1);
                        weight1++;
                        *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-2**weight1+1)+2*((3-
2**weight1)**weight1-*(weight1-1)-2**(weight1+1)+1)*(3-4**weight1)-2*((3-
2**(weight1+1))**(weight1+1)-*weight1-2**(weight1+2)+1);
                        weight1++;
                        for (i=1;i<n-3;i++)
                        {
                                *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-*(weight1-2)-
2**weight1+1)+2*((3-2**weight1)**weight1-*(weight1-1)-2**(weight1+1)+1)*(3-4**weight1)-2*((3-
2**(weight1+1))**(weight1+1)-*weight1-2**(weight1+2)+1);
                                weight1++;
                        }
                        *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-*(weight1-2)-
2**weight1+1)+2*((3-2**weight1)**weight1-*(weight1-1)-2**(weight1+1)+1)*(3-4**weight1)-2*((3-
2**(weight1+1))**(weight1+1)-*weight1+1);
                weight1++;
                *gradient1++=-4*((3-2**(weight1-1))**(weight1-1)-*(weight1-2)-2**weight1+1)+2*((3-
2**weight1)**weight1-*(weight1-1)+1)*(3-4**weight1);
                }
        }
        else if(function==2)
        {
                int i,j;
                double gradient_help, *re;
                gradient_help=0.0;
                re=weight1;
                for (i=1;i<=n;i++)
                {
                        gradient_help+=i*(*weight1-1);
                        weight1++;
                }
                for (j=1;j<=n;j++)
                {
                        *gradient1++=2*(*re-1)+2*j*gradient_help+4*j*sq3(gradient_help);
                        re++;
                }
        }
        else if(function==3)
        {
                int i,j,k;
                double grad_sum,grad_sum2,*re,*re2,a,a1,b,b1;
```

```
                re=weight1;
                re2=weight1;
                for(k=1;k<=n;k++)
                {
                        grad_sum2=0.0;
                        for(i=1;i<=n;i++)
                        {
                                weight1=re2;
                                grad_sum=0.0;
                                for(j=1;j<=n;j++)
                                {
                                        a=5*(1+(i%5)+(j%5));
                                        b=((double)i+(double)j)/10;
                                        grad_sum+=a*sin(*weight1)+b*cos(*weight1);
                                        weight1++;
                                        if(j==k)
                                        {
                                                a1=a;
                                                b1=b;
                                        }
                                }
                                grad_sum2+=2*(n+i-grad_sum)*(-a1*cos(*re)+b1*sin(*re));
                        }
                        *gradient1++=grad_sum2;
                        re++;
                }
        }
        else if(function==4)
        {
                int i;
                double grad_help,*weight_help;
                grad_help=0.0;
                weight_help=weight1;
                for (i=1;i<=n;i++)
                {
                        grad_help+=0.5*i*(*weight1);
                        weight1++;
                }
                weight1=weight_help;
                for(i=1;i<=n;i++)
                {
                        *gradient1++
=2*(*weight1)+2*0.5*i*grad_help+4*0.5*i*grad_help*sq(grad_help);
                        weight1++;
                }
        }
        else if(function==5)
        {
                int i,j;
                double gradient_help,gradient_help2, *re,*weight2,n2;
                gradient_help=0.0;
                gradient_help2=0.0;
                n2=(double) n;
                re=weight1;
                weight2=re;
                for (i=1;i<=n;i++)
                {
                        gradient_help+=cos(*weight1);
                        weight1++;
                }
                weight1=re;
                for (i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if(j==i)
                                {
                                        weight2++;
```

```
                                        }
                                        else
                                        {
                                                gradient_help2+=2*sin(*weight1)*(n2-gradient_help+j*(1-
cos(*weight2))-sin(*weight2));
                                                weight2++;
                                        }
                                }
                                *gradient1++=2*(n2-gradient_help+i*(1-cos(*weight1))-
sin(*weight1))*((i+1)*sin(*weight1)-cos(*weight1))+gradient_help2;
                                weight1++;
                                weight2=re;
                                gradient_help2=0.0;
                        }
                }
                else if(function==6)
                {
                        int i;
                        *gradient1++=2*(*weight1-1)-4*(2*sq(*(weight1+1))-*weight1);
                        weight1++;
                        for (i=2;i<n;i++)
                        {
                                *gradient1++=2*i*(2*sq(*weight1)-*(weight1-1))*4*(*weight1)-
2*(i+1)*(2*sq(*(weight1+1))-*weight1);
                                weight1++;
                        }
                        *gradient1++=2*n*(2*sq(*weight1)-*(weight1-1))*4*(*weight1);
                }
}
```

## functions1.h

```
#include <stdio.h>

double calculate_cost_criterion(double *);
void calculate_gradient(double *,double *);
```

## opt_util1.cpp

```
#include "opt_util1.h"
#include "neuron1.h"
#include "math_utils1.h"

void constrain_weights()
{
        int i;
        for(i=0;i<n;i++)
        {
                if(weight[i]>max_weight[i])
                        weight[i]=max_weight[i];
                if(weight[i]<min_weight[i])
                        weight[i]=min_weight[i];
        }
}

void constraint_gradient()
{
        int i;
        for(i=0;i<n;i++)
        {
                c_gradient[i]=gradient[i];
                if(weight[i]==min_weight[i] && search_direction[i]<0.0)
                        c_gradient[i]=0.0;
                if(weight[i]==max_weight[i] && search_direction[i]>0.0)
                        c_gradient[i]=0.0;
        }

}
```

```
void initialise_search_direction(double *gr)
{
	int i;
	for(i=0;i<n;i++)
		search_direction[i]=gr[i];
}

void calculate_norms()
{
	int i;
	g_norm=0.0;
	for(i=0;i<n;i++)
	{
		g_norm+=pow(c_gradient[i],2.0);
	}
}

void search_direction_function()
{
	int i;
	double v[100][100];
	double w[100][100];
	double be[100][100];
	double b1,b2;
   switch (method)
	{
		/**steepest descent**/
		case 1:
			if(method11==1)
			{
				scalar_matrix(A,A,-1.0,n*n);
				mult_complex(A,gradient,search_direction,n);
			}
			else
			{
				scalar_matrix(aux_matrix,gradient,-1.0,n);
				copy_matrix(search_direction,aux_matrix,n);
			}
			break;
		/****DFP*******/
		case 2:
		/*pseudo4=1;*/
			if(method9 == 1)
			{
				if(opt_it==0)
				{
					if(method11==1)
					{
						copy_matrix(H,A,n*n);
					}
					else
						unit_matrix(H,n);
				}
				else
				{
					sub_matrix(delta,weight,weight_0,n);
					sub_matrix(gamma,c_gradient,c_gradient_0,n);
					mult_complex(H,gamma,HG,n);
					mult_vecsym(HG,HG,quotient,n);
					denominator=scalar_product(gamma,HG,n);
					mult_sym(quotient,1/denominator,n);
					mult_vecsym(delta,delta,inter1,n);
					denominator=scalar_product(delta,gamma,n);
					mult_sym(inter1,1/denominator,n);
					sub_matrix(inter1,inter1,quotient,n*n);
					add_matrix(H,inter1,H,n*n);
				}
				mult_complex(H, gradient,inter,n);
```

```
		interp = inter;
		scalar_matrix(interp,interp,-1.0,n);
		copy_matrix(search_direction,interp,n);
		scalar_matrix(interp,interp,-1.0,n);
}
else
{
		pseudo4=1;
		if(opt_it==0)
		{
			if(method11==1)
			{
				scalar_matrix(A,A,-1.0,n*n);
				mult_complex(A,gradient,search_direction,n);
			}
			else
			{
				scalar_matrix(aux_matrix,gradient,-1.0,n);
				copy_matrix(search_direction,aux_matrix,n);
			}
		}
		else
		{
			sub_matrix(delta,weight,weight_0,n);
			sub_matrix(gamma,c_gradient,c_gradient_0,n);
			if(method11==1)
			{
				mult_complex(A,gamma,gradienta_0,n);
				mult_complex(A,c_gradient,gradienta,n);
			}
			if(method11==1)
			{
				copy_matrix(z,gradienta_0,n);
				scalar_matrix(gradienta,gradienta,-1.0,n);
			}
			else
			{
				copy_matrix(z,gamma,n);
			}
			if ( pseudo4>=2)
			{
				for (j3=0;j3<=pseudo4-2;j3++)
				{
					for(i=0;i<n;i++)
					{
						v_h[i]=v[i][j3];
						w_h[i]=w[i][j3];
					}
				vy=scalar_product(v_h,gamma,n);
					wy=scalar_product(w_h,gamma,n);
						scalar_matrix(v_h,v_h,vy,n);
						scalar_matrix(w_h,w_h,wy,n);
				sub_matrix(z_h,v_h,w_h,n);
				add_matrix(z,z,z_h,n);
					}
			}
			ra=scalar_product(delta,gamma,n);
			rb=scalar_product(z,gamma,n);
			if((ra<=0)^(rb<=0))
			{
		keeper=method_prev;
				indicator1=1;
				indicator2=1;
				method=1;printf(" DFP ");
				pseudo4=2;
				indicator4=1;
			}
			else
```

```
                                        {
                                                ra=1/sqrt(ra);
                                                rb=1/sqrt(rb);
                                                scalar_matrix(v_h1,delta,ra,n);
                                                scalar_matrix(w_h1,z,rb,n);
                                                for(i=0;i<n;i++)
                                                {
                                                        v[i][pseudo4-1]=v_h1[i];
                                                        w[i][pseudo4-1]=w_h1[i];
                                          }
                                                scalar_matrix(aux_matrix,gradient,-1.0,n);
                                                if(method11==1)
                                                {

copy_matrix(search_direction,gradienta,n);
                                                }
                                                else
                                                {

copy_matrix(search_direction,aux_matrix,n);
                                                }
                                                for (j3=0;j3<=pseudo4-1;j3++)
                                                {
                                                        for(i=0;i<n;i++)
                                                        {
                                                                v_h[i]=v[i][j3];
                                                                w_h[i]=w[i][j3];
                                                        }
                                                        vg=scalar_product(v_h,c_gradient,n);
                                                        wg=scalar_product(w_h,c_gradient,n);
                                                        scalar_matrix(v_h,v_h,vg,n);
                                                        scalar_matrix(w_h,w_h,wg,n);
                                                        sub_matrix(s_h,v_h,w_h,n);

sub_matrix(search_direction,search_direction,s_h,n);
                                                }
                                        }
                                }
                                pseudo4=pseudo4+1;
                        }
                        break;
                /********BFGS*****/
                case 3:
                        pseudo4=1;
                        if(method9 == 1)
                        {
                                if(opt_it==0)
                                {
                                        if(method11==1)
                                        {
                                                copy_matrix(H,A,n*n);
                                        }
                                        else
                                                unit_matrix(H,n);
                                }
                           else
                                {
                                        sub_matrix(delta,weight,weight_0,n);
                                        sub_matrix(gamma,c_gradient,c_gradient_0,n);
                                        mult_complex(H,gamma,HG,n);
                                        mult_complex1(gamma,H,G1H,n);
                                        denominator1=scalar_product(gamma,HG,n);
                                        denominator=scalar_product(delta,gamma,n);
                                        multiplier=denominator1/denominator;
                                        multiplier=1+multiplier;
                                        mult_vecsym(delta,delta,quotient,n);
                                        denominator=scalar_product(delta,gamma,n);
                                        mult_sym(quotient,1/denominator,n);
```

```
                    mult_sym(quotient,multiplier,n);
                        mult_vector2(delta,G1H,DG1H,n);
                        mult_vector2(HG,delta,HGD1,n);
                        add_matrix(nominator,HGD1,DG1H,n*n);
                        denominator=scalar_product(delta,gamma,n);
                        mult_scalar(nominator,1/denominator,inter1,n);
                        sub_matrix(inter1,quotient,inter1,n*n);
                        add_matrix(H,H,inter1,n*n);
                }
                mult_complex(H, gradient,inter,n);
                interp = inter;
                scalar_matrix(interp,interp,-1.0,n);
                copy_matrix(search_direction,interp,n);
                scalar_matrix(interp,interp,-1.0,n);
        }
                else
                {
                        if(opt_it==0)
                        {
                                if(method11==1)
                                {
                                        scalar_matrix(A,A,-1.0,n*n);
                                        mult_complex(A,gradient,search_direction,n);
                                }
                                else
                                {
                                        scalar_matrix(aux_matrix,gradient,-1.0,n);
                                        copy_matrix(search_direction,aux_matrix,n);
                                }
                        }
                        else
                        {
                                sub_matrix(delta,weight,weight_0,n);
                                sub_matrix(gamma,c_gradient,c_gradient_0,n);
                                if(method11==1)
                                {
                                        mult_complex(A,gamma,gradienta_0,n);
                                        mult_complex(A,c_gradient,gradienta,n);
                                        copy_matrix(z,gradienta_0,n);
                                        scalar_matrix(gradienta,gradienta,-1.0,n);
                                }
                                else
                                {
                                        copy_matrix(z,gamma,n);
                                }
if ( pseudo4>=2)
                                {
                                        for (j3=0;j3<=pseudo4-2;j3++)
                                        {
                        for(i=0;i<n;i++)
                                                {
                                                        v_h[i]=v[i][j3];
                                                        w_h[i]=w[i][j3];
                                                        b_h[i]=be[i][j3];
                                                }
                                                vy=scalar_product(v_h,gamma,n);
                                                wy=scalar_product(w_h,gamma,n);
                                                by=scalar_product(b_h,gamma,n);
                                        scalar_matrix(v_h,v_h,vy,n);
                                                scalar_matrix(w_h,w_h,wy,n);
                                                scalar_matrix(b_h,b_h,by,n);
                        sub_matrix(z_h,v_h,w_h,n);
                                                add_matrix(z_h,z_h,b_h,n);
                        add_matrix(z,z,z_h,n);
                                        }
                                }
                                ra=scalar_product(delta,gamma,n);
                                rb=scalar_product(z,gamma,n);
```

```
                                        if((ra<=0)^(rb<=0))
                                        {
                                keeper=method_prev;
                                                indicator1=1;
                                                indicator2=1;
                                                method=1;
                                                pseudo4=2;
                                                indicator4=1;
                                        }
                                        else
                                        {
                                                rc= sqrt(rb);
                                                ra=1/sqrt(ra);
                                                rb=1/sqrt(rb);
                                                rc=ra*ra*rc;
                                                scalar_matrix(v_h1,delta,ra,n);
                                                scalar_matrix(w_h1,z,rb,n);
                                                scalar_matrix(b_h1,delta,rc,n);
                                                sub_matrix(b_h1,b_h1,w_h1,n);
                                                for(i=0;i<n;i++)
                                                {
                                                        v[i][pseudo4-1]=v_h1[i];
                                                        w[i][pseudo4-1]=w_h1[i];
                                                        be[i][pseudo4-1]=b_h1[i];
                                                }
                                                scalar_matrix(aux_matrix,gradient,-1.0,n);
                                                if(method11==1)
                                                {

        copy_matrix(search_direction,gradienta,n);
                                                }
                                                else
                                                {

        copy_matrix(search_direction,aux_matrix,n);
                                                }
                                                for (j3=0;j3<=pseudo4-1;j3++)
                                                {
                                                        for(i=0;i<n;i++)
                                                        {
                                                                v_h[i]=v[i][j3];
                                                                w_h[i]=w[i][j3];
                                                                b_h[i]=be[i][j3];
                                                        }
                                                        vg=scalar_product(v_h,c_gradient,n);
                                                        wg=scalar_product(w_h,c_gradient,n);
                                                bg=scalar_product(b_h,c_gradient,n);
                                                        scalar_matrix(v_h,v_h,vg,n);
                                                        scalar_matrix(w_h,w_h,wg,n);
                                                        scalar_matrix(b_h,b_h,bg,n);
                                                        sub_matrix(s_h,v_h,w_h,n);
                                                        add_matrix(s_h,s_h,b_h,n);

        sub_matrix(search_direction,search_direction,s_h,n);
                                                }
                                        }
                                }
                                pseudo4=pseudo4+1;
                        }
                        break;
/****Fletcher Reeves***/
                case 4:
                        if(opt_it==0)
                        {
                                if(method11 == 1)
                                {
                                        scalar_matrix(A,A,-1.0,n*n);
                                        mult_complex(A,gradient,search_direction,n);
```

```
                        }
                        else
                        {
                                scalar_matrix(aux_matrix,gradient,-1.0,n);
                                copy_matrix(search_direction,aux_matrix,n);
                        }
                }
                else
                {
                        copy_matrix(search_direction_0,search_direction,n);
                        if(method11 == 1)
                        {
                                mult_complex(A,c_gradient_0,gradienta_0,n);
                                mult_complex(A,c_gradient,gradienta,n);
                                b2=scalar_product(c_gradient_0,gradienta_0,n);
                                b1=scalar_product(c_gradient,gradienta,n);
                        }
                        else
                        {
                                b2=scalar_product(c_gradient_0,c_gradient_0,n);
                                b1=scalar_product(c_gradient,c_gradient,n);
                        }
                        beta=b1/b2;
                        scalar_matrix(search_direction,search_direction_0,beta,n);
                        scalar_matrix(aux_matrix,c_gradient,-1.0,n);
                        add_matrix(search_direction,search_direction,aux_matrix,n);
                }
                break;
        /*****Polak Ribiere****/
                case 5:
                if(opt_it==0)
                {
                        if(method11 == 1)
                        {
                                scalar_matrix(A,A,-1.0,n*n);
                                mult_complex(A,gradient,search_direction,n);
                        }
                        else
                        {
                                scalar_matrix(aux_matrix,gradient,-1.0,n);
                                copy_matrix(search_direction,aux_matrix,n);
                        }
                }
                else
                {
                        copy_matrix(search_direction_0,search_direction,n);
                        if(method11 == 1)
                        {
                                mult_complex(A,c_gradient_0,gradienta_0,n);
                                mult_complex(A,c_gradient,gradienta,n);
                                b2=scalar_product(c_gradient_0,gradienta_0,n);
                                sub_matrix(aux_matrix,c_gradient,c_gradient_0,n);
                                b1=scalar_product(aux_matrix,gradienta,n);
                        }
                        else
                        {
                                b2=scalar_product(c_gradient_0,c_gradient_0,n);
                                sub_matrix(aux_matrix,c_gradient,c_gradient_0,n);
                                b1=scalar_product(aux_matrix,c_gradient,n);
                        }
                        beta=b1/b2;
                        scalar_matrix(search_direction,search_direction_0,beta,n);
                        scalar_matrix(aux_matrix,c_gradient,-1.0,n);
                        add_matrix(search_direction,search_direction,aux_matrix,n);
                }
                break;
        }
}
```

```
void line_optimisation()
{
	double a,b,z,w;
	double alpha1,alpha0;
	double J_a,J_b,J_alpha,J_a_hat,J_alpha1;
	double dJ_dalpha_0,dJ_dalpha,dJ_da_hat;
	double dJ_da,dJ_db;
	double aux_var;
	double a_hat;
	int brack=1,line_opt;
	int l_n,pointer;
	int line_iter,flag1,flag2,stoper;
	double  x1, /* The x1 value*/x2,  /*  The x2 value*/J_x1,J_x2;
	int  L; /* The value which control's the golden sectioning search*/
	line_opt=1;
	copy_matrix(weight_0,weight,n);
	copy_matrix(c_gradient_0,c_gradient,n);
	J_alpha1=criterion;
	line_iter=0;
	/********************** Bracketing phase *************************/
	dJ_dalpha_0=scalar_product(c_gradient,search_direction,n);
	if(opt_it==0 || dJ_dalpha_0==0.0)
		/*alpha1=0.02;*/alpha1=-2.0*(criterion*0.0001)/dJ_dalpha_0;
	else
		alpha1=-2.0*deltaJ/dJ_dalpha_0;
	alpha0=0.0;
	/*printf("%f %f\n",alpha1,alpha0);*/
	/*getchar();*/
	while(brack && line_opt && line_iter<=max_iter)
	{
		scalar_matrix(aux_matrix,search_direction,alpha1,n);
		add_matrix(weight,aux_matrix,weight_0,n);
		constrain_weights();
		J_alpha=calculate_cost_criterion(weight);
		calculate_gradient(weight,gradient);
		constraint_gradient();
		/*printf("%f %f\n",J_alpha,J_alpha1);*/
		if(J_alpha>=J_alpha1)
		{
			/*printf("%s\n","FEA 1o");*/
			a=alpha0;
			b=alpha1;
			brack=0;
		}
		if(brack)
		{
			dJ_dalpha=scalar_product(c_gradient,search_direction,n);
			/*printf("%f %f\n",dJ_dalpha,dJ_dalpha_0);*/
			if(fabs(dJ_dalpha)<= -1.0*sigma*dJ_dalpha_0)
			{
				/*printf("%s\n","FEA 2o");*/
				line_opt=0;
				a=alpha1;
				b=0;
				deltaJ=criterion-J_alpha;
				criterion=J_alpha;
				/*printf("%f
%f\n",a*1000000000000000000,b*1000000000000000000);*/
			}
			if(brack && line_opt)
			{
				if(dJ_dalpha>=0.0)
				{
					/*printf("%s\n","FEA 3ao");*/
					a=alpha0;
					b=alpha1;
```

```
                                        /*printf("%f
%f\n",a*1000000000000000000,b*1000000000000000000);*/
                                        brack=0;
                                }
                                else
                                {
                                        /*printf("%s\n","FEA 3bo");*/
                                        aux_var=alpha1;
                                        alpha1=alpha1+t1*(alpha1-alpha0);
                                        alpha0=aux_var;
                                }
                        }
                }/* end if(brack) */
                line_iter++;
                /*printf("%d %d %d\n",line_iter,brack,line_opt);*/
        }/* end while of bracketing */
        flag2=0;
        /*printf("%f %f\n",alpha1,alpha0);
        printf("%f %f\n",dJ_dalpha,dJ_dalpha_0);*/
        /* getchar();*/
        if(method1!=3&& line_iter<max_iter&& flag2 ==0)
        {
                stoper = 0;
                while(line_opt && line_iter<max_iter)
                {
                        if(line_opt)
                        {
                                stoper = stoper + 1;
                                scalar_matrix(aux_matrix,search_direction,a,n);
                                add_matrix(weight,aux_matrix,weight_0,n);
                                constrain_weights();
                                J_a=calculate_cost_criterion(weight);
                                calculate_gradient(weight,gradient);
                                constraint_gradient();
                                dJ_da=scalar_product(c_gradient,search_direction,n);
                                scalar_matrix(aux_matrix,search_direction,b,n);
                                add_matrix(weight,aux_matrix,weight_0,n);
                                constrain_weights();
                                J_b=calculate_cost_criterion(weight);
                                calculate_gradient(weight,gradient);
                                constraint_gradient();
                                dJ_db=scalar_product(c_gradient,search_direction,n);
                                /*printf("%f %f %f
%f",a*1000000000000000000,b*1000000000000000000,dJ_da,dJ_db);
                                getchar();*/
                                if (stoper==20)
                                {
                                        if (J_a_hat>J_a)
                                        {
                                                a=0;
                                                keeper=method_prev;
                                                indicator1=1;
                                                indicator2=1;
                                                method=1;
                                        }
                                        else
                                        {
                                                a=a_hat;
                                        }
                                        /*line_opt=0;*/
                                }
                                switch (method1)
                                {
  /****** quadratic interpolation in [a,b]   ******/
                                        case 1:
                                                if(fabs((b-a)*dJ_da+J_a-
J_b)<0.0000000000000000001)
                                                {
```

```
                                                deltaJ=0.0;
                                                break;
                                        }
                                        a_hat=0.5*(2.0*a*(J_a-J_b)+dJ_da*(pow(b,2)-
pow(a,2)))/((b-a)*dJ_da+J_a-J_b);
                                        break;
   /****** cubic interpolation in [a,b]   ******/
                                case 2:
                                if(fabs(a-b)<0.00000000000000000001)
                                        {
                                                a_hat=a;
                                                break;
                                        }
                                        z=3*((J_a-J_b)/(b-a))+dJ_da+dJ_db;
                                        w=sqrt(pow(z,2)-(dJ_da*dJ_db));
                                        a_hat=b-(((dJ_db+w-z)/(dJ_db-dJ_da+(2*w)))*(b-
a));
                                        break;
                        }
                        scalar_matrix(aux_matrix,search_direction,a_hat,n);
                        add_matrix(weight,aux_matrix,weight_0,n);
                        constrain_weights();
                        J_a_hat=calculate_cost_criterion(weight);
                        calculate_gradient(weight,gradient);
                        constraint_gradient();
                        dJ_da_hat=scalar_product(c_gradient,search_direction,n);
                        flag1=1;
                        if(J_a_hat>=J_a)
                        {
                                b=a_hat;
                                flag1=0;
                                /*printf("%d %f %f %f %f\n ",1,a_hat,z,dJ_da,dJ_db);*/
                        }
                        if(flag1)
                        {
                                if(fabs(dJ_da_hat)<= -1.0*sigma*dJ_dalpha_0)
                                {
                                        line_opt=0;
                                        deltaJ=criterion-J_a_hat;
                                        criterion=J_a_hat;
                                        /*   printf("%d\n",2);*/
                                }
                                if(line_opt)
                                {
                                        if(dJ_da_hat>0)
                                        {
                                                b=a_hat;
                                                /*printf("%d %f %f %f %f\n
",2,a_hat,z,dJ_da,dJ_db);*/
                                        }
                                        else
                                        {
                                                a=a_hat;
                                                /*printf("%d %f %f %f %f\n
",3,a_hat,z,dJ_da,dJ_db);*/
                                        }
                                }
                        }
                        }
                        line_iter++;
                }/*end of while(line_opt) */
        }
        else
        {
 /********** Golden Sectionong ***********/
                l_n=log(e_value/fabs(b-a))/log(c_value);
                L=l_n;
                pointer=0;
```

```
                x1=a+(1-c_value)*(b-a);
                x2=a+c_value*(b-a);
                scalar_matrix(aux_matrix,search_direction,x1,n);
                add_matrix(weight,aux_matrix,weight_0,n);
                constrain_weights();
                J_x1=calculate_cost_criterion(weight);
                scalar_matrix(aux_matrix,search_direction,x2,n);
                add_matrix(weight,aux_matrix,weight_0,n);
                constrain_weights();
                J_x2=calculate_cost_criterion(weight);
                while(pointer<=L)
                {
                        if(J_x1<J_x2)
                        {
                                b=x2;
                                x2=x1;
                                J_x2=J_x1;
                                x1=a+(1-c_value)*(b-a);
                                scalar_matrix(aux_matrix,search_direction,x1,n);
                                add_matrix(weight,aux_matrix,weight_0,n);
                                constrain_weights();
                                J_x1=calculate_cost_criterion(weight);
                        }
                        else
                        {
                                a=x1;
                                x1=x2;
                                J_x1=J_x2;
                                x2=a+c_value*(b-a);
                                scalar_matrix(aux_matrix,search_direction,x2,n);
                                add_matrix(weight,aux_matrix,weight_0,n);
                                constrain_weights();
                                J_x2=calculate_cost_criterion(weight);
                        }
                        pointer=pointer+1;
        }
                scalar_matrix(aux_matrix,search_direction,a,n);
                add_matrix(weight,aux_matrix,weight_0,n);
                constrain_weights();
                J_a_hat=calculate_cost_criterion(weight);
                calculate_gradient(weight,gradient);
                constraint_gradient();
                deltaJ=criterion-J_a_hat;
                criterion=J_a_hat;
        }
}

void rprop()
{
        double cpu_time;
        clock_t ticks;
        clock_t ticks_begin;
        FILE *fp6,*fp8,*fp9;
        fp6=fopen("RPROP cpu time.txt","w");
        fp8=fopen("RPROP J.txt","w");
        fp9=fopen("shmeiaR.txt","w");
        ticks_begin=clock();
        int i;
        for (i=0;i<n;i++)
                fprintf(fp9,"%f ",weight[i]);
        fprintf(fp9,"\n");
        if ((opt_it==0)||(opt_it==1))
        {
                for(i=0;i<n;i++)
                {
                        delta_x[i]=0.0;
                }
        }
```

```
while (g_norm>gg)
{
	if ((opt_it==0)||(opt_it==1))
	{
		for(i=0;i<n;i++)
		{
			delta_x[i]=0.0;
		}
	}
	if(opt_it==1)
	{
		input_vector1(a_step,n);
		copy_matrix(a_step_old,a_step,n);
	}
	if(opt_it>1)
	{
		mult_scalarx(c_gradient_0,c_gradient_1,gpr,n);
		sigr(gpr,a_step_old,a_step,d_max,d_min,n);
		copy_matrix(a_step_old,a_step,n);
		/*printf("%f\n",gpr[1]);*/
		if(gpr<0)
		{
			for(i=0;i<n;i++)
			{
				delta_x[i]=-1.*delta_x_prev[i];
				c_gradient_1[i]=0.0;
				printf("gpr<0");
			}
			printf("gpr<0");
		}
		else
		{
			sigr2(c_gradient_1,a_step,delta_x,n);
		}
		copy_matrix(delta_x_prev,delta_x,n);
	}
	add_matrix(weight,weight,delta_x,n);
	calculate_gradient(weight,c_gradient);
	if(opt_it==0)
	{
		for(i=0;i<n;i++)
		{
			c_gradient_0[i]=c_gradient[i];
		}
	}
	if(opt_it==1)
	{
		for(i=0;i<n;i++)
		{
			c_gradient_1[i]=c_gradient[i];
		}
	}
	if(opt_it>1)
	{
		for(i=0;i<n;i++)
		{
			c_gradient_0[i]=c_gradient_1[i];
			c_gradient_1[i]=c_gradient[i];
		}
	}
	criterion = calculate_cost_criterion(weight);
	/*weight[0]=1;*/
	calculate_norms();
	opt_it++;
	printf("iter=%d\tcriterion=%f\n",opt_it,criterion);
	ticks = clock() - ticks_begin;
	cpu_time = (double) ticks/CLOCKS_PER_SEC;
	fprintf(fp6,"%f\n",cpu_time);
```

```
                fprintf(fp8,"%f\n",criterion);
                for (i=0;i<n;i++)
                        fprintf(fp9,"%f\t ",weight[i]);
                fprintf(fp9,"\n");
        }   /*   BIG While    */
        /*for (i=0;i<n;i++)
        {
                if (i==0)
                        fprintf(fp9,"%f %f ",weight[i],c_gradient[i]);
                else
                fprintf(fp9,"%f ",weight[i]);
        }
        fprintf(fp9,"\n");*/
        /*fprintf(fp9,"%f %f\n",weight[0],weight[1]);*/
        fclose(fp6);
        fclose(fp8);
        fclose(fp9);
}
```

## opt_util1.h

```
#include <stdio.h>
#include "neuron1.h"
#include "functions1.h"

void line_optimisarion(void);
void constrain_weights(void);
void constraint_gradient(void);
void initialise_search_direction(double *);
void search_direction_function(void);
void line_optimisation(void);
void calculate_norms(void);
void rprop(void);
```

## math_utils1.cpp

```
#include "math_utils1.h"

void copy_matrix(double *dest,double *src,int n)
{
        int i;
        for(i=0;i<n;i++)
        dest[i]=src[i];
}

double scalar_product(double *mtr1,double *mtr2,int n)
{
        int i;
        double result;
        result=0.0;
        for(i=n;i;i--)
                result += *mtr1++ * *mtr2++;
        return result;
}

void sub_matrix(double *dest,double *mtr1,double *mtr2,int n)
{
        int i;
        for(i=n;i;i--)
                *dest++=*mtr1++-*mtr2++;
}

void scalar_matrix(double *dest,double *mtr,double scal,int n)
{
        int i;
        for(i=n;i;i--)
                *dest++ = scal* *mtr++;
}

void add_matrix(double *dest,double *mtr1,double *mtr2,int n)
```

```
{
	int i;
	for(i=n;i;i--)
		*dest++ = *mtr1++ + *mtr2++;
}

void print_matrix(double *mtr,int n)
{
	int i;
	for(i=0;i<n;i++)
		printf("%d\t%.5f\n",i,mtr[i]);
}

void mult_vecsym(double  *a, double  *b, double  *l,int n)
{
	int i, j;
	double *bu;
	for (i=0;i<n;i++)
	{
		l += i;
		bu = b+i;
		for (j=0;j<n-i;j++)
	   {
			*l++ = *(l+j*n-j) = *a * *bu++;
	   }
		a++;
	}
}


void mult_complex(double  *a, double  *b, double  *c,int n)
{
	int i, j;
	double *bu, *cu;
	cu=c;
	for  (i=n;i;i--)
		*cu++=0;
   cu=c;
	for (i=n;i;i--)
	{
		bu=b;
		for (j=n;j;j--)
	   *cu += *a++ * *bu++;
		cu++;
   }
}

void mult_complex1(double  *k, double  *a, double  *l,int n)
{
	int i, j;
	double *bu;
   bu=l;
	for  (i=n;i;i--)
		*bu++=0;
	for (i=n;i;i--)
	{
		bu=l;
		for (j=n;j;j--)
	   {
			*bu++ += *a++ * *k;
	   }
		k++;
	}
}

void mult_scalar(double  *a,double  x,double  *b,int n)
{
	int i,j;
```

```
	for (i=n;i;i--)
	{
		for (j=n;j;j--)
		{
			*b++ = x * *a++;
		}
	}
}

void unit_matrix(double  *a,int n)
{
	int i;
	for (i=n*n;i;i--)
   if (!((i%(n+1))-1))
		*a++=1;
	else
		*a++=0;
}

void mult_sym(double *a,double x,int n)
{
	int i,j;
	double *bu;
	bu=a;
	for (i=0;i<n;i++)
	{
		bu += i;
		for(j=0;j<n-i;j++)
		{
			*bu *= x;
			*(bu+j*n-j) = *bu;
			bu++;
		}
	}
}

void mult_vector2(double  *z,double  *k,double  *C,int n)
{
	int i, j;
	double *b;
	for (i=0;i<n;i++)
	{
		b=k;
		for (j=0;j<n;j++)
		*C++ = *z * *b++;
		z++;
	}
}

void sigr(double  *a, double  *b, double  *c, double d1, double d2, int nr)
{
	int i;
	for(i=0;i<nr;i++)
	{
		if(a[i]<0)
			if(0.5*b[i]>d2)
				c[i]=0.5*b[i];
			else
				c[i]=d2;
		if(a[i]>0)
			if(1.2*b[i]<d1)
				c[i]=1.2*b[i];
			else
				c[i]=d1;
		if(a[i]==0)
			c[i]=b[i];
	 }
}
```

```
void sigr2(double  *a, double  *b, double  *c, int nr)
{
	int i;
	for(i=0;i<nr;i++)
	{
		if(a[i]>0)
			c[i]=-1.* b[i];
		if(a[i]<0)
			c[i]=1.* b[i];
		if(a[i]==0)
			c[i]=0.;
	}
}

void input_vector1(double *a, int n)
{
	int i;
	for(i=0;i<n;i++)
		*a++ =0.1;
}

void mult_scalarx(double *a,double *xa,double *b,int n)
{
	int i;
	for (i=n;i;i--)
	{
		*b++ = *xa++ **a++;
	}
}

void ypol_a(double  *v, double  *ypol,int n)
{
	int i;
	*ypol++ = 1./(1200. * *v**v-400.**(v+1) + 2.);
	v++;
	for (i=1;i<n-1;i++)
	{
		*ypol++ = 1./(1200. * *v**v-400.**(v+1) + 2.)+ 1./200. ;
		v++;
	}
	*ypol = 1./200. ;
}

void unit_matrix1(double  *a,double *b,int n)
{
	int i;
	for (i=n*n;i;i--)
		if (!((i%(n+1))-1))
			*a++=*b++;
		else
			*a++=0;
}
```

## math_utils1.h

```
#include "neuron1.h"

void copy_matrix(double *,double *,int);
double scalar_product(double *,double *,int);
void sub_matrix(double *,double *,double *,int);
void scalar_matrix(double *,double *,double,int);
void add_matrix(double *,double *,double *,int);
void print_matrix(double *,int);
void mult_vecsym(double  *, double  *, double  *,int);
void mult_complex(double  *, double  *, double  *,int);
void mult_complex1(double  *, double  *, double  *,int);
void mult_scalar(double  *,double  ,double  *,int);
```

```
void unit_matrix(double  *,int);
void mult_sym(double *,double ,int);
void mult_vector2(double  *,double  *,double  *,int );
void sigr(double  *, double  *, double  *, double ,double , int );
void sigr2(double  *, double  *, double  *, int );
void input_vector1(double *, int );
void mult_scalarx(double *,double *,double *,int );
void ypol_a(double  *, double  *,int );
void unit_matrix1(double  *,double *,int );
```