ΜΕΘΟΔΟΙ ΤΟΠΙΚΗΣ ΚΑΙ ΚΑΘΟΛΙΚΗΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ

Η ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Υποβάλλεται στην

ορισθείσα από τη Γενική Συνέλευση Ειδικής Σύνθεσης

του Τμήματος Πληροφορικής

Εξεταστική Επιτροπή

από τον

ΚΩΝΣΤΑΝΤΙΝΟ ΒΟΓΚΛΗ

ως μέρος των Υποχρεώσεών του για τη λήψη

ΔΙΔΑΚΤΟΡΙΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Ιούνιος 2010

Thesis Committee

Isaac E. Lagaris (Supervisor), Professor, Department of Computer Science, University of Ioannina

Nikolaos P. Galatsanos, Professor Department of Electrical and Computer Engineering, University of Patras

Aristeidhs Lykas, Associate Professor, Department of Computer Science, University of Ioannina

Thesis Approve Committee

Isaac E. Lagaris, Professor, Department of Computer Science, University of Ioannina

Nikolaos P. Galatsanos, Professor, Department of Electrical and Computer Engineering, University of Patras

Ioannis Demetropoulos, Professor, Department of Informatics and Telecommunications Engineering, University of Western Macedonia

Charalambos Mpotsaris, Department of Regional Economic Development ,University of Central Greece

Aristeidhs Lykas, Associate Professor, Department of Computer Science, University of Ioannina

Konstantinos E. Parsopoulos, Assistant Professor, Department of Computer Science, University of Ioannina

Dimitrios Papageorgiou, Assistant Professor, Department of Materials Science and Engineering, University of Ioannina

Ευχαριστιες

Θα ήθελα να απευθύνω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου κ. Ισαάκ Λαγαρή για την πολύτιμη βοήθεια του για την περάτωση της παρούσας εργασίας και για τη συνεχή και αδιάκοπη στήριξή του κατά τη διάρκεια των σπουδών μου.

Θα ήθελα επίσης να ευχαριστήσω τους συνεργάτες σε όλη την πορεία του διδακτορικού μου, Δημήτρη Παπαγεωργίου, Παναγιώτη Χατζηδούκα και Κωνσταντίνο Παρσόπουλο.

Επίσης ευχαριστώ από την καρδιά μου τους πολύ καλούς μου φίλους Χάρη Παπαδόπουλο, Τσίπουρα Μάρκο (κουμπάρος), Γιαννίκου Αναστασία (κουμπάρα), Καζιάννη Σπύρο (μέγιστος Φυσικός 1), Λιόντο Ιωάννη (μέγιστος Φυσικός 2), Αμοιρόπουλο Κων/νο (μέγιστος Φυσικός 3), Χασάνη Βασίλη (Α' Μηχανικό).

Ευχαριστώ τους γονείς μου για την υλική και ηθική συμπαράστασή τους κατά τη διάρκεια των σπουδών μου, τον αδερφό μου που με ανέχεται τόσα χρόνια.

Τέλος, ευχαριστώ την Πηνελόπη, που χωρίς την υπομονή της, την υποστήριξη της και την αγάπη της δεν θα είχα επιτύχει τους στόχους μου.

Κωνσταντίνος Βόγκλης
Ιωάννινα, Ιούλιος 2010

Extended Abstract

Costas Voglis, PhD Computer Science Department, University of Ioannina Greece. June, 2010. Methods for Local and Global Optimization. Thesis Supervisor: Isaac E. Lagaris

In this thesis new methods for local and global optimization are presented their behavior is analyzed and experimentally tested. Also a broad review of standard techniques for local and global optimization is given.

The issues presented in this thesis vary from one dimensional optimization to stochastic methods for global optimization applied in problems in many dimensions. Each topic described, is presented in clear algorithmic form and compared to standard competitive methods.

In the topic of local optimization four new techniques are presented and one modification of an existing method. Concerning global optimization a stochastic global framework consisting of four steps is presented. For each step in this framework the contribution of this thesis is exposed.

This thesis is divided in two main parts: In the first part the research on local optimization is presented and in the second part the results for the global optimization counterpart.

Part 1: Local Optimization

Chapter 2

In Chapter 2 a complete bibliographic survey of the most important methods in local optimization is presented.

Chapter 3

In this chapter, an algorithm for solving a quadratic programming problem with positive definite Hessian and bound constraints, that employs a Lagrange multiplier approach is presented. The quadratic programming problem with simple bounds is stated as:

$$\begin{aligned} q(x) = \min_x \ & \frac{1}{2} x^T B x + x^T d, \\ \text{subject to: } & a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\} \end{aligned} \tag{1}$$

where $x, d \in R^n$ and B is a symmetric, positive definite $n \times n$ matrix. The proposed method falls in the category of active set techniques. The algorithm, at each iteration, modifies the minimization parameters both in the primal space and in the dual space (Lagrange multipliers). The method may be profitably used on a number of problems from the fields of Physics, Chemistry, Computer Science and Engineering. Comparative results of numerical experiments are reported demonstrating the advantages of the proposed approach.

The algorithm presented is an infeasible active set algorithm, which generates a finite number of iterations that are not necessarily descent. In each step the first order optimality condition along with the complementarity constraint are maintained, until primal and dual feasibility hold. Two closely related methods in the literature are the Projected Newton method and the infeasible method of Kunisch and Rendl that treats only upper bounds.

Chapter 4

A trust region algorithm for unconstrained and bound constrained nonlinear optimization problems is presented in Chapter 4. The trust region is a rectangular hyperbox in contrast with the commonly used hyperellipsoid. The resulting quadratic subproblems are solved approximately by an adaptation of Powell's dogleg method for rectangular trust regions and a the novel quadratic programming algorithm presented in Chapter 3. The problem we are concerned is

$$\begin{aligned} &\min_x f(x), \\ &\text{subject to: } a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\} \end{aligned}$$

where $x, a, b \in R^n$.

The method developed that adopts a rectangular shape for the trust region. This geometry has the obvious advantage of the linearity of the subproblem constraints and in addition allows effortless adaptation to bound constrained problems. The emerging quadratic subproblems are of the sort:

$$\min_s m(s) = \frac{1}{2} s^T B s + s^T g \quad \text{subject to:} \quad \alpha_i \leq s_i \leq \beta_i \tag{2}$$

a modification of Powell's dogleg technique is developed to obtain an approximate solution and an exact technique based on quadratic algorithm in Chapter 3.

We embed this scheme in a quasi–Newton framework that uses a positive definite approximation to the Hessian matrix. This renders the problem in Eq.4.1 a strictly convex one, and hence the dogleg technique and the convex quadratic solver are applicable.

Chapter 5

In this Chapter a local search method suitable for supervised training of feed-forward artificial neural networks, with one hidden layer and sigmoidal activation functions is

developed. The resulting Sum-of-Squares objective function is minimized using a hybrid technique that switches between the Gauss–Newton (GN) approach in the small residual case, and Newton's method in case where large residuals are detected. This is done in the spirit of Fletcher and Xu where instead of Newton's method, a variable metric method (BFGS) was preferred in order to avoid the calculation of the Hessian matrix, which in the general case is both costly and cumbersome. In the special case that is considered here, the Hessian matrix can be expressed analytically and calculated efficiently by taking advantage of the properties of the sigmoidal activation function and its derivatives.

The Sum-of-Squares problem is stated as

$$\begin{aligned} \min_x F(x) &= \sum_{i=1}^{m} f_i^2(x), \\ \text{subject to: } & a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\} \end{aligned}$$

where $f_i : R^n \to R$ $i = 1, \ldots, m$ continuous and differentiable functions and $x, a, b \in R^n$.

In comparing GN and Newton methods, the GN is generally preferred for zero residual problem (ZRP) that is when $\mathbf{r}(x^*) = 0$, whereas Newton-like methods are preferred for large residual problems (LRP) or when J_k looses rank.

Usually is not known beforehand whether a problem will turn out to have small or large residuals at the solution. It seems reasonable, therefore, to consider *hybrid algorithms*, which would behave like Gauss-Newton if the residuals turn out to be small (and take advantage of the cost savings associated with these methods) but switch to Newton like steps if the residuals at the solution are large (with the cost of approximating or computing second order derivatives).

Chapter 6

In Chapter 6 a software library for numerically estimating first and second order partial derivatives of a function by finite differencing is presented. Various truncation schemes are offered resulting in corresponding formulas that are accurate to order $O(h)$, $O(h^2)$, and $O(h^4)$, h being the differencing step. The derivatives are calculated via forward, backward and central differences. Care has been taken that only feasible points are used in the case where bound constraints are imposed on the variables. The Hessian may be approximated either from function or from gradient values. There are three versions of the software: a sequential version, an OpenMP version for shared memory architectures and an MPI version for distributed systems (clusters). The parallel versions exploit the multiprocessing capability offered by computer clusters, as well as modern multicore systems and due to the independent character of the derivative computation, the speed up scales almost linearly with the number of available processors/cores.

Part 2: Global Optimization

In the second part of this dissertation, algorithms for a certain class of stochastic global optimization are presented. Stochastic two-phase clustering and sampling techniques are

the main concern of this thesis. These algorithms consist of two phases: a global phase where the search space is explored using a sampling algorithms and a local phase realized by a local optimization algorithm. In this thesis the general global optimization problem is tackled, that can be formulated as:

$$\begin{aligned} &\text{Find all minima } f(x), \\ &\text{subject to: } a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\} \end{aligned}$$

Obviously, if all minima are retrieved the global minimum is found to. This problem is presented in bibliography [24, 33, 36, 77, 86, 156, 163].

Below a general algorithm of a stochastic two-phase clustering algorithm is presented.

General Algorithm 1: Stochastic Two-Phase Clustering

Step 1. Sample search space.

Step 2. Cluster sample points int groups that correspond to the same minimum.

Step 3. From representative points of the cluster start a local search.

Step 4. Check for termination.

In this thesis an alternative methodology to the General Algorithm 1 is proposed. Instead of applying clustering to identify already found local minima, one can create a suitable adaptive sampling distribution that will not take samples around already found local minima. The general algorithm for adaptive sampling distribution creation in the global optimization framework is given bellow:

General Algorithm 2: Stochastic Two-Phase Distribution Driven

---

Step 1. Sample from the distribution.

Step 2. Start a local search from the sampled point.

Step 3. Update distribution parameters.

Step 4. Check for termination.

In global optimization bibliography many contributions have been made for every step of General Algorithm 1. The ideas described in this thesis, follow the same line of research. Considering steps 1 and 3 of the General Algorithm 2, two alternative sampling techniques are described in Chapters 8 and 9 respectively. In Chapter 10 a clustering algorithm is presented, suitable for Step 2 of General Algorithm 1. In Chapter 11 a local search method appropriate for both general algorithms is presented and finally in Chapter 12 a new termination criterion is introduced.

Chapter 7

In this Chapter an introduction to stochastic, two-phase optimization with clustering is performed and a detailed bibliographic review is given.

Chapter 8

A stochastic global optimization method based on *Multistart* is presented. In this, the local search is conditionally applied with a probability that takes in account the topology of the objective function at the detail offered by the current status of exploration. As a result, the number of unnecessary local searches is drastically limited, yielding an efficient method. Results of its application on a set of common test functions are reported, along with a performance comparison against other established methods of similar nature.

The method is based on the definition of the *region of attraction* of the local minimum and makes use of an *maximum attraction radius* in order to define this region. By making use a probabilistic model around each minimum recognizing the region of attraction of various shapes and sizes is achieved. When a sample point is detected inside a region of attraction, a local search is **not** performed. On the other size, when a sample point is considered outside **all** regions of attraction then a local search is performed. The proposed method can be also seen as a variation of General Algorithm 2, where the sampling is defined implicitly.

Chapter 9

In this chapter a novel method for selecting candidate starting points for stochastic two-phase algorithms, is proposed. The sampling method takes into account previous local searches. The information revealed from the local search forms a normal distribution around the most recently found local minimum. This is a direct way to implement General Algorithm 2, presented in introduction. The final sampling distribution is a weighted sum of all normal distributions created around the retrieved local minima.

Chapter 10

In Chapter 10 a new approach for clustering in stochastic global optimization framework is presented. The proposed algorithm attempts to cluster sample points around minima using the global k-means algorithm enhanced with spectral information. The major contribution is a novel way of introducing gradient information into the clustering problem. Gradient information is proven to be very helpful in defining clusters around minima. After the cluster creation step the minima are retrieved using a local optimization method starting from cluster centers.

The proposed approach can be described in brief in Algorithm 0.1 :

Algorithm 0.1 The proposed clustering method

S1 *Sample points in the region of interest*: For this step we use two alternatives:

S2 *Concentrate the sample to obtain groups around the local minima*: Displace sample points by using a fraction of the negative gradient or few steps of a local optimizer.

S3 *Recognize these groups by the aid of a clustering method*: In general out clustering method consists on two main steps (which will be thoroughly analyzed later):

(a) Estimate the number of clusters k formed by the concentrated sampled points

(b) Apply global k-means (or a proposed variation) seeking k clusters.

S4 *Stopping condition*: Any stopping criterion from the bibliography can be used.

Chapter 11

Stochastic methods based on multistart, that employ a clustering scheme to separate different regions of attractions have proven to be quite successful. The research in this direction was pioneered by Rinnoy Kan and its group in a series of articles. Various authors followed up this line, see for example Torn and Viitanen, Schoen and Locatelli, Ali and Storey and a host of methods and software implementations have appeared in the literature. A common feature of these methods is the use of a local search (LS), i.e. a procedure for locating a local minimum. The characteristics of this procedure play an important role as far as the performance and the effectiveness of the global method is concerned. If by $x^* = \mathcal{L}(x)$ we denote that a local search started at point x, will end up finding the local minimizer x^*, then the region of attraction of a minimizer x^* may be defined as the set $A(\mathcal{L}, x^*) = \{x_i, x^* = \mathcal{L}(x_i)\}$ and depends in addition to the position of the minimum x^*, on the LS procedure.

If x^* and y^* are distinct local minima $A(\mathcal{L}, x) \cap A(\mathcal{L}, y) \neq \emptyset$ provided that the local search is deterministic. Stochastic LS procedures create overlapping regions of attraction a fact that in the framework is rather undesirable. Also regions of attraction may be contiguous or not. Evidently a non–contiguous region can not be described by a single cluster, and hence the existence of such regions may influence the performance of the method negatively. So a proper LF for clustering should be such that the regions of attraction that it creates are contiguous. Vrahatis et al have provided a tool for visualizing the regions of attraction. An interesting fact is that all of the most successful LS search create disjoint regions. Hence a LS with contiguous regions of attraction would be very useful for clustering methods.

Chapter 12

For a broad class of global optimization problems, it can never be verified in finite time that the global optimum is identified with certainty. Therefore a need emerges for stopping rules which decide if the expected benefit of further searching outweighs the required computational effort.

Stopping rules have to decide for the path between the Scylla of computational efficiency and the Charybdis of the completeness warranty. In other words their objective is to collect the complete set of the existent local minima with the least computational effort. The ideal case would be to stop the search as soon as all the minima have been discovered. Since this is not possible, further searching is necessary to ensure that there are no left–out minima, a fact that inevitably leads to a compromise. So the stopping rules, depending on the specific problem at hand, negotiate either for efficiency or for a degree of completeness.

In this Chapter a new stopping rule is described that can be applied in every multistart-like global optimization framework. The basic assumption is that one can calculate theoretically the relation between the number of recovered minima m and the number of local searches k for a problem that has w distinct local minima. Also suppose that this is a relation of the sort

$$m = N \equiv N^{(k)}(w), \quad N \to w \text{ as } k \to \infty \tag{3}$$

Imagine now that one applies multistart-based algorithm and plots the number of recovered minima versus the number of local searches.

One then at the k-th local search, may compare the *experimental* curve with the *theoretical* one and find which w is the one that produces the best match. If this is possible then at k-th local search we will now the number of expected local minima and hence a very efficient stopping rule may emerge.

The stopping rule proposed is based on the assumption that *The probability of locating a local minimum, among the w distinct ones, by applying a local search is* $p = \frac{1}{w}$.

Περιληψη

Βόγκλης Κωνσταντίνος του Αλεξάνδρου και της Ευφροσύνης, PhD Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούνιος, 2010. Μέθοδοι Τοπικής και Καθολικής Βελριστοποίησης. Επιβλέποντας: Ισαάκ Η. Λαγαρής

Στην παρούσα διατριβή πέραν μιας σύντομης βιβλιογραφικής ανασκόπησης των μεθόδων αιχμής για το πρόβλημα της τοπικής και καθολικής βελτιστοποίησης (ελαχιστοποίησης), παρουσιάζονται νέοι αλγόριθμοι, τόσο για τοπική όσο και για καθολική βελτιστοποίηση, αναλύεται η συμπεριφορά τους και εξετάζεται πειραματικά η απόδοσή τους.

Το εύρος θεμάτων της παρούσας διατριβής ξεκινά από μέθοδο μονοδιάστατης τοπικής ελαχιστοποίησης και φτάνει σε στοχαστικές μεθόδους για τη λύση του προβλήματος του καθολικού ελαχίστου για συναρτήσεις πολλών μεταβλητών. Το κάθε θέμα που περιγράφεται συνοδεύεται από αναλυτικές αναφορές στη βιβλιογραφία, παρουσιάζεται σε αλγοριθμική μορφή και συγκρίνεται με αντίστοιχες μεθόδους.

Όσο αφορά την τοπική βελτιστοποίηση παρουσιάζονται σε αυτή τη διατριβή τέσσερεις νέες τεχνικές και μία τροποποίηση υπάρχουσας τεχνικής. Μια από τις νέες μεθόδους που αφορά μονοδιάστατη τοπική ελαχιστοποίηση θα μελετηθεί στο πλαίσιο της Καθολικής Ελαχιστοποίησης, αφού αναπτύχθηκε για να λύσει προβλήματα στο γενικό πλαίσιο Καθολικής Ελαχιστοποίησης.

Η παρούσα διατριβή χωρίζεται σε δυο τμήματα: Στο πρώτο τμήμα παρουσιάζουμε τα ευρήματα που αφορούν τοπική ελαχιστοποίηση και στο δεύτερο, μεθοδολογία για καθολική ελαχιστοποίηση. Ακολουθεί συνοπτική περιγραφή του κάθε τμήματος.

1ο Τμήμα: Τοπική Ελαχιστοποίηση

2ο Κεφάλαιο

Στο δεύτερο Κεφάλαιο παρουσιάζουμε μια εκτενή βιβλιογραφική αναφορά στις πιο σημαντικές τεχνικές τοπικής ελαχιστοποίησης, ξεκινώντας από τον ορισμό του ελαχίστου, τις ικανές και αναγαίες συνθήκες για την ύπαρξη του και προχωρόντας σε αλγορίθμους ανάλογα με την πληροφορία που διαθέτουμε για την αντικειμενική συνάρτηση. Διαλέξαμε την ταξινόμηση των μεθόδων κατ' αρχήν σε μονοδιάστατες και πολυδιάστατες και κατά δεύτερον σε μεθόδους που δεν χρησιμοποιούν παραγώγους, σε μεθόδους που χρησιμοποιούν 1η παράγωγο και σε μεθόδους που χρησιμοποιούν πρώτες και δεύτερες παραγώγους. Η σύνοψη αυτή προσπαθεί να ορίσει τις βάσεις πάνω στις οποίες στηρίχτηκε η έρευνα και να επιδείξει τα σημεία που

ερευνήθηκαν και αναπτύχθηκαν περαιτέρω.

3ο Κεφάλαιο

Στο Κεφάλαιο 3, παρουσιάζεται μια νέα τεχνική για την επίλυση του προβλήματος του κυρτού τετραγωνικού προγραμματισμού (convex quadratic programming) με απλά όρια (bound constraints). Το πρόβλημα που λύνεται περιγράφεται ως:

$$q(x) = \min_x \ \frac{1}{2}x^T Bx + x^T d,$$
$$\text{υπό τον περιορισμό: } a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\}$$

όπου $x, d \in R^n$ και B συμμετρικός, θετικά ορισμένος $n \times n$ πίνακας. Το συγκεκριμένο πρόβλημα, αν και αρκετά εξειδικευμένο, βρίσκει πληθώρα εφαρμογών σε προβλήματα κατασκευών, μηχανικής, βιοιατρικής, υπολογιστικής φυσικής ακόμα και ταξινόμησης δεδομένων στα πλαίσια της υπολογιστικής νοημοσύνης. Ένας ακόμα βασικός λόγος που μας οδήγησε να ασχοληθούμε με το συγκεκριμένο πρόβλημα είναι ότι προκύπτει σαν υποπρόβλημα σε μεθόδους τοπικής ελαχιστοποίησης που χρησιμοποιούν περιοχές εμπιστοσύνης. Τέτοιες μέθοδοι, που θα παρουσιαστούν στο επόμενο Κεφάλαιο της διατριβής, απαιτούν σε κάθε επανάληψή τους την επίλυση ενός προβλήματος κυρτού τετραγωνικού προγραμματισμού με απλούς περιορισμούς στις μεταβλητές.

Η επίλυση ακολουθεί την μεθοδολογία των πολλαπλασιαστών Lagrange και ικανοποιεί τις συνθήκες Kunh-Tucker για την ύπαρξη ελαχίστου.

Σε κάθε επανάληψη ορίζονται τρία σύνολα δεικτών, και ανατίθενται τιμές τόσο στις παραμέτρους x όσο και στους πολλαπλασιαστές Lagrange λ, μ ανάλογα με το σύνολο που ανήκουν. Με τον τρόπο αυτό επιτυγχάνουμε να λύνουμε σε κάθε επανάληψη ένα γραμμικό σύστημα το πολύ n τάξης, από τη λύση του οποίου θα προκύψουν τα σύνολα της επόμενης επανάληψης.

Βασικό υπολογιστικό κόστος του αλγορίθμου είναι η λύση σε κάθε επανάληψη ενός γραμμικού συστήματος το πολύ n τάξης. Εκτεταμένα πειράματα σε πλήθος πραγματικών και τεχνητών προβλημάτων αποδεικνύουν την ανωτερότητα της μεθόδου, έναντι ανταγωνιστικών τεχνικών στη βιβλιογραφία.

4ο Κεφάλαιο

Το Κεφάλαιο 4, περιλαμβάνει μελέτη του προβλήματος τοπικής ελαχιστοποίησης με απλά όρια χρησιμοποιώντας την τεχνική των περιοχών εμπιστοσύνης (trust region). Το πρόβλημα που μας απασχολεί σε αυτήν την περίπτωση είναι το:

$$\min_x f(x),$$
$$\text{υπό τον περιορισμό: } a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\}$$

όπου $x, a, b \in R^n$. Σύμφωνα με την μέθοδο της περιοχής εμπιστοσύνης η συνάρτηση $f(x)$ προσεγγίζεται τοπικά με ένα κυρτό τετραγωνικό μοντέλο το οποίο ελαχιστοποιείται σε κάθε επανάληψη υπό τον περιορισμό μιας περιοχής ενδιαφέροντος (trust region).

Η περιοχή ενδιαφέροντος στην βιβλιογραφία είναι συνήθως σφαιρική (όπου και καθορίζεται από μια ακτίνα Δ_k). Στην παρούσα διατριβή, παρουσιάζουμε μια διαφορετική προσέγγιση χρησιμοποιώντας υπερ-ορθογώνια (hyperrectangle) περιοχή ενδιαφέροντος (που καθορίζεται από την πλευρά του υπερ-ορθογωνίου Δ_k).

Η επιλογή αυτή βασίζεται στην απλή παρατήρηση ότι εάν το πρόβλημα έχει ήδη απλά όρια σαν περιορισμούς, η τομή των απλών ορίων με το υπερ-ορθογώνιο της περιοχής εμπιστοσύνης είναι επίσης υπερ-ορθογώνιο. Έτσι το υποπρόβλημα που πρέπει να λυθεί σε κάθε επανάληψη είναι σαν αυτό που μελετήσαμε στο Κεφάλαιο 3, δηλαδή κυρτή τετραγωνική συνάρτηση με περιορισμούς απλά όρια.

Το υποπρόβλημα λύνεται με δυο τεχνικές. Μια προσεγγιστική βασισμένη στην τεχνική κυνόπους (dogleg) που ανέπτυξε ο Powell και μια ακριβής χρησιμοποιώντας τον αλγόριθμο που περιγράψαμε στο Κεφάλαιο 3 της διατριβής. Η προσεγγιστική λύση που παρουσιάζουμε είναι ελαφρώς τροποποιημένη για να ταιριάζει στην υπερ-ορθογώνια περιοχή εμπιστοσύνης. Από την άλλη μεριά η ακριβής λύση, αν και υπολογιστικά πιο ακριβή, συγκλίνει ταχύτερα στο τοπικό ελάχιστο. Ο αλγόριθμος υπερ-ορθπογώνιας περιοχής εμπιστοσύνης μαζί με τις δυο μεθόδους επίλυσης του τετραγωνικού υποπροβήματος δοκιμάστηκε εκτενώς σε προβλήματα με περιορισμούς, αλλά και χωρίς περιορισμούς με εξαιρετική απόδοση.

5ο Κεφάλαιο

Στο Κεφάλαιο 5, παρουσιάζεται μια τροποποίηση της γνωστής μεθόδου των Fletcher και Xu για προβλήματα ελαχιστοποίησης συναρτήσεων της μορφής ελαχίστων τετραγώνων, καθώς και την εφαρμογή της σε προβλήματα εκπαίδευσης νευρωνικών δικτύων. Στην μελέτη αυτή το πρόβλημα που επιλύεται διατυπώνεται ως:

$$\min_x F(x) = \sum_{i=1}^{m} f_i^2(x),$$
$$\text{υπό τον περιορισμό: } a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\}$$

όπου $f_i : R^n \to R$ συνεχείς και παραγωγίσιμες συναρτήσεις για $i = 1, \ldots, m$ και $x, a, b \in R^n$. Η μεθοδολογία των Fletcher και Xu αποτελεί ένα συνδυασμό δυο πολύ γνωστών προσεγγίσεων του Εσσιανού πίνακα. Της BFGS για τοπική ελαχιστοποίηση και της μεθόδου του Gauss-Newton για την ειδική περίπτωση του αθροίσματος τετραγώνων.

Η μέθοδος μπορεί να χρησιμοποιήσει σε κάθε επανάληψη τον ένα ή τον άλλο αλγόριθμο ανάλογα με ένα κριτήριο που βασίζεται στο ποσοστό μείωσης της αντικειμενικής συνάρτησης. Το κριτήριο αυτό προσπαθεί να διαγνώσει αν το πρόβλημα ανήκει στην κατηγορία του μικρού υπολοίπου (small residual) ή σε αυτή του μεγάλου υπολοίπου (large residual). Στην πρώτη κατηγορία η λύση είναι κοντά στο μηδέν, ενώ στην δεύτερη κατηγορία όχι.

Για να κατανοήσουμε το διαχωρισμό θα δώσουμε τον τύπο του πίνακα δευτέρων παραγώγων (hessian matrix) σε προβλήματα ελαχίστων τετραγώνων.

$$H(x_k) = \nabla^2 F(x_k) = J^T(x_k) J(x_k) + \sum_{i=1}^{m} f_i(x_k) \nabla^2 f_i(x_k)$$

όπου $J(x), n \times m$ πίνακας με στοιχεία $\dfrac{\partial f_i(x)}{\partial x_j}$.

Τα προβλήματα μικρού έχουν λύση x^* στην οποία $f_i(x^*) \simeq 0$ και κατά συνέπεια κοντά στη λύση ο εσσιανός πίνακας μπορεί να προσεγγιστεί με ακρίβεια με την προσέγγιση $H(x_k) = J^T(x_k)J(x_k)$, που ουσιαστικά είναι η προσέγγιση Gauss-Newton. Σε αντίθετη περίπτωση η προσέγγιση Gauss-Newton δεν είναι ακριβής μιας και ο όρος $\sum_{i=1}^{m} f_i(x_k)\nabla^2 f_i(x_k)$ είναι σημαντικός. Στην περίπτωση αυτή καλό θα ήταν να χρησιμοποιηθεί μια καλύτερη προσέγγιση όπως πχ. η BFGS ενημέρωση.

Η συνεισφορά της διατριβής έγκειται σε δυο σημεία. Πρώτον, η BFGS αντικαταστάθηκε από την μέθοδο Newton που χρησιμοποιεί δεύτερες παραγώγους και δεύτερον η μέθοδος εφαρμόστηκε για πρώτη φορά στην εκπαίδευση νευρωνικών δικτύων. Για το σκοπό αυτό, υπολογίσαμε αναλυτικά τις πρώτες και δεύτερες παραγώγους για ένα οποιοδήποτε νευρωνικό δίκτυο με ένα κρυμμένο επίπεδο. Συνοπτικά ο αλγόριθμος που παρουσιάζεται χρησιμοποιεί, σε κάθε επανάληψή του, το κριτήριο των Fletcher & Xu για να αποφασίσει αν θα χρησιμοποιήσει την Gauss-Newton (φτηνή) προσέγγιση ή τον πλήρη εσσιανό πίνακα, ο οποίος είναι υπολογιστικά πιο ακριβός.

6ο Κεφάλαιο

Επειδή ο υπολογισμός των παραγώγων παίζει πάρα πολύ σημαντικό λόγο σε αλγόριθμους ελαχιστοποίησης, α μια παράλληλη μια βιβλιοθήκη λογισμικού (NDL) που παρέχει τη δυνατότητα για παράλληλο υπολογισμό πρώτων και δεύτερων παραγώγων με τη μέθοδο των πεπερασμένων διαφορών (finite differences). Δυο σημαντικοί λόγοι μας οδήγησαν στην ανάπτυξη της βιβλιοθήκης. Ο πρώτος είναι η ανυπαρξία ελεύθερου πακέτου λογισμικού για τον παράλληλο υπολογισμό των παραγώγων. Ο δεύτερος λόγος έχει να κάνει με τη ραγδαία ανάπτυξη και χρήση παράλληλων υπολογιστικών συστημάτων, είτε σε επίπεδο συστάδων υπολογιστών (clusters), είτε σε αρχιτεκτονικές πολλών πυρήνων (multicore). Ο κάθε ένας πλέον μπορεί διαθέτει ένα πολυπύρηνο υπολογιστικό σύστημα, και η NDL το εκμεταλλεύεσαι στο έπακρο.

Η βιβλιοθήκη παρέχει μια αναλυτική διεπαφή για κάθε παράγωγο που θέλουμε να υπολογιστεί, την ακρίβεια που επιθυμεί ο χρήστης και αν υπάρχουν όρια στις μεταβλητές. Η βιβλιοθήκη μπορεί να χρησιμοποιηθεί σε εφαρμογές βελτιστοποίησης στις οποίες αναλυτικές παράγωγοι είναι δύσκολο να υπολογιστούν και υποστηρίζει διεπαφές για τις πλατφόρμες MPI και OpenMP.

Συνοπτικά η βιβλιοθήκη που παρουσιάζεται έχει τα εξής χαρακτηριστικά που την ξεχωρίζουν από τον ανταγωνισμό:

- *Α*υτόματος υπολογισμός του βήματος για επίτευξη επιθυμητής ακρίβειας προσέγγισης: Είναι γνωστό ότι στις μεθόδους πεπερασμένων διαφορών υπολογίζονται οι τιμές της συνάρτησει σε μικρές μετατοπίσεις γύρω από το σημείο που μας ενδιαφέρει η παράγωγος. Ανάλογα με την μετατόπιση και τον τύπο που χρησιμοποιούμε, μπορούμε να επιτύχουμε και συγκεκριμένη ακρίβεια προσέγγισης. Ο χρήστης είναι ελεύθερος να επιλέξει μόνο την επιθυμητή ακρίβεια και η βιβλιοθήκη κάνει τους υπολογισμού

για το βήμα.

- *Υ*ποστήριξη απλών ορίων στις μεταβλητές : Η βιβλιοθήκη με κατάλληλους ελέγχους σέβεται το πεδίο ορισμού της συνάρτησης που παρέχει ο χρήστης. Έτσι οδηγείται στο να χρησιμοποιήσει προς τα μπρος, κεντρικές ή προς τα πίσω διαφορές, ανάλογα με το αν παραβιάζονται τα όρια.

- *Η* υποστήριξη πλατφόρμας MPI και OpenMP: Τα σύγχρονα παράλληλα υπολογιστικά συστήματα μπορούν να χωριστούν είτε σε κατανεμημένα συστήματα πολλών επεξεργαστών που επικοινωνούν με γρήγορα τοπικά δίκτυα (cluster of nodes) ή σε συστήματα κοινής μνήμης με πολλούς επεξεργαστές που επικοινωνούν με διαύλους (shared memory multiprocessors). Και για τις δυο περιπτώσεις, η βιβλιοθήκη παρέχει τρόπους εκτέλεσης, που μπορεί να είναι και *μεικτοί*.

- *Ε*πιλογή τρόπου κατανομής των κλήσεων για τον υπολογισμό των παραγώγων: Η βιβλιοθήκη NDL παρέχει τη δυνατότητα να μεταβάλει κανείς τον τρόπο με τον οποίο κατανέμονται οι συναρτησιακές κλήσεις στους διαθέσιμους επεξεργαστές.

Η βιβλιοθήκη έχει δοκιμαστεί ενδελεχώς σε όλες τις περιπτώσεις χρήσης. Επίσης, έχουν γίνει μετρήσεις για τον υπολογισμό της επιτάχυνσης (speed-up) σε διάφορα παράλληλα συστήματα και φαίνεται ότι η παρούσα βιβλιοθήκη μπορεί να επιφέρει σημαντικά κέρδη σε χρόνο υπολογισμού. Αυτό επιβεβαιώνεται από τις καμπύλες της επιτάχυνσης, οι οποίες προσεγγίζουν τη θεωρητική ακόμα και για πολύ *γρήγορες* συναρτήσεις.

2ο Τμήμα: Καθολική Ελαχιστοποίηση

Στο 2ο τμήμα της διατριβής θα παρουσιαστούν τα αποτελέσματα της έρευνας για το αντικείμενο της καθολικής ελαχιστοποίησης. Σε γενικές γραμμές η έρευνα μας κατευθύνθηκε σε μια συγκεκριμένη, αν και ευρεία, κατηγορία τεχνικών που αναφέρονται στην βιβλιογραφία ως στοχαστικές τεχνικές δυο φάσεων με ομαδοποίηση (stochastic two-phase clustering techniques). Οι τεχνικές αυτές περιλαμβάνουν δυο φάσεις, μια καθολική κατά την οποία εξερευνάται ο χώρος αναζήτησης και μια τοπική όπου συνήθως εφαρμόζεται ένα αλγόριθμος τοπικής ελαχιστοποίησης. Σύμφωνα με τον ορισμό, το καθολικό ελάχιστο είναι ένα σημείο x^* για το οποίο ισχύει $f(x) > f(x^*),\ \forall x \neq x^*, a \leq x \leq b$. Στην παρούσα διατριβή δεν ασχολούμαστε μόνο με την ύπαρξη ενός καθολικού ελαχίστου, αλλά αντιμετωπίζουμε ένα πιο γενικό πρόβλημα:

$$\begin{aligned} &\text{Βρες όλα τα ελάχιστα της } f(x), \\ &\text{υπό τον περιορισμό: } a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\} \end{aligned}$$

Προφανώς, εάν βρούμε όλα τα ελάχιστα μπορούμε να επιλέξουμε το μικρότερο σαν *ένα* καθολικό ελάχιστο. Το πρόβλημα του εντοπισμού όλων των ελαχίστων εμφανίζεται πολύ συχνά στη βιβλιογραφία πχ. [24, 33, 156, 86, 163, 77, 36]

Συγκεκριμένα το γενικό περίγραμμα της μεθοδολογίας που μελετήσαμε εμφανίζεται παρακάτω:

Γενικός Αλγόριθμος 1: Stochastic Two-Phase Clustering

Βήμα 1. Δειγματοληψία του χώρου αναζήτησης.

Βήμα 2. Ομαδοποίησηση των σημείων που ανήκουν στην ομάδα που οδηγεί στο ίδιο ελάχιστο.

Βήμα 3. Από αντιπροσωπευτικά σημεία των ομαδοποιήσεων με τοπική ελαχιστοποίηση και εντοπισμός τα ελάχιστα.

Βήμα 4. Έλεγχος κριτηρίου τερματισμού.

Ο γενικός αλγόριθμος που περιγράψαμε, στην ακραία του μορφή ανάγεται στον κλασικό αλγόριθμο πολλαπλής εκκίνησης (Multistart). Συγκεκριμένα, αν στο Βήμα 1, ορίσουμε ομοιόμορφη κατανομή και στο Βήμα 2 δεν ορίσουμε καμία ομαδοποίηση αλλά ξεκινάμε από όλα τα σημεία τοπικές ελαχιστοποιήσεις (Βήμα 3), τότε έχουμε περιγράψει την μέθοδο Multistart. Πολύ μεγάλη ερευνητική προσπάθεια έχει καταβληθεί τα τελευταία 20 χρόνια για να βελτιωθεί ο αλγόριθμος Multistart με απώτερο σκοπό: *τη διενέργεια μιας τοπικής ελαχιστοποίησης ανά ελάχιστο.*

Στην παρούσα διατριβή πέρα από τη συμβολή μας σε συγκεκριμένα βήματα του γενικού αλγορίθμου 1, ο οποίος έχει μελετηθεί εξαντλητικά από ερευνητές τα τελευταία 20 χρόνια, προτείναμε και μια άλλη ισοδύναμη και γενική μεθοδολογία. Αντί να προσπαθεί κανείς να ομαδοποιήσει, εκ των υστέρων, τυχαία δείγματα στο χώρο αναζήτησης με κοινή ιδιότητα ότι οδηγούν στο ίδιο ελάχιστο, μπορεί να παράγει τυχαία σημεία με τέτοιο τρόπο, ώστε να αποφεύγονται τα ήδη ευρεθέντα ελάχιστα. Και με τις δυο μεθοδολογίες ο απώτερος σκοπός της διενέργειας μιας τοπικής ελαχιστοποίησης ανά ελάχιστο είναι ίδιος.

Ο γενικός αλγόριθμος καθοδηγούμενης από εκπαιδευόμενες κατανομές αναζήτησης περιγράφεται παρακάτω:

Γενικός Αλγόριθμος 2: Stochastic Two-Phase Distribution Driven

---

Βήμα 1. Δειγματοληψία του χώρου αναζήτησης από την κατανομή.

Βήμα 2. Τοπική ελαχιστοποίηση και εντοπισμός ελαχίστων.

Βήμα 3. Ενημέρωση των παραμέτρων της κατανομής.

Βήμα 4. Έλεγχος κριτηρίου τερματισμού.

Στη βιβλιογραφία έχει παρουσιαστεί πληθώρα μεθόδων που σκοπεύουν να βελτιώσουν το γενικό αλγόριθμο 1 και ή συνεισφορά τους στόχευε κάποιο από τα 4 βήματα. Στην παρούσα διατριβή, στα επόμενα κεφάλαια, θα περιγραφούν οι νέες ιδέες που ενισχύουν τον γενικό αλγόριθμο και υλοποιούν τα βήματα 1 και 3 του γενικού αλγορίθμου 2.

Συγκεκριμένα, όσο αφορά τα Βήματα 1 και 3 του γενικού αλγόρίθμου 2 περιγράφονται δυο διαφορετικές τεχνικές στα κεφάλαια 8 και 9. Στο Κεφάλαιο 10 παρουσιάζεται ένας αλγόριθμο ομαδοποίησης για το Βήμα 2 του γενικού αλγορίθμου 1. Στο Κεφάλαιο 11 περιγράφεται μια τροποποίηση μιας κλασικής μεθόδου τοπικής ελαχιστοποίησης κατάλληλης και για τα δυο προαναφερθέντα αλγοριθμικά πλαίσια. Τέλος στο Κεφάλαιο 12 παρουσιάζεται ένα νέο κριτήριο τερματισμού, οδηγούμενο από την ιδανική περίπτωση ότι κάθε ελάχιστο έχει ίση πιθανότητα να ανακτηθεί από έναν αλγόριθμο τοπικής ελαχιστοποίησης που ξεκινά από τυχαίο σημείο.

7ο Κεφάλαιο

Στο Κεφάλαια 7 παρατίθεται μια εισαγωγή στο πρόβλημα της καθολικής ελαχιστοποίησης, και μια αναλυτική παρουσίαση της βιβλιογραφίας των στοχαστικών μεθόδων και συγκεκριμένα των μεθόδων ομαδοποίησης (clustering global optimization). Στην κατηγορία των στοχαστικών μεθόδων κατατάσσονται όλοι οι αλγόριθμοι στους οποίους αναφέρεται η χρήση τυχαίων μεταβλητών και το αποτέλεσμα της εκτέλεσής τους, με σταθερές παραμέτρους, δεν είναι κατ' ανάγκη το ίδιο. Η αναλυτική εισαγωγή βοηθάει τον αναγνώστη να πάρει μια *γεύση*, περισσότερο αλγοριθμική, σχετικά με την κατηγορία των στοχαστικών αλγορίθμων καθολικής ελαχιστοποίησης.

8ο Κεφάλαιο

Στο 8ο Κεφάλαιο περιγράφεται ένας αλγόριθμος (Adapt) μια υλοποίηση του Γενικού Αλγορίθμου 2. Η Adapt περιγράφει έμμεσα μια κατανομή από την οποία προκύπτουν σημεία για τοπική ελαχιστοποίηση, ορίζοντας *απαγορευμένες* περιοχές γύρω από ελάχιστα.

Η απόφαση για το αν από ένα αρχικό σημείο θα πρέπει να ξεκινήσει τοπική ελαχιστοποίηση, παίρνεται με τη βοήθεια πληροφορίας από τα ήδη συλλεχθέντα ελάχιστα, της θέσης τους και μιας πληροφορίας σχετικής με το μέγεθος της περιοχής τους[1].

Αφού περιγραφεί αναλυτικά η μέθοδος, γίνεται μια ανάλυση της συμπεριφοράς της σε μεγάλο αριθμό επαναλήψεων (αρχικών σημείων) και σημειώνεται ότι αν εκτελεστούν άπειρες επαναλήψεις δειγματοληψίας αρχικών σημείων η πιθανότητα να ΜΗΝ βρεθεί κάποιο ελάχιστο μηδενίζεται. Επίσης κατά την περιγραφή της μεθόδου παρουσιάζονται και τα πρώτα ψήγματα για την ανάγκη της τοπικής μονοδιάστατης ελαχιστοποίησης του Κεφαλαίου 11.

Κλείνοντας το κεφάλαιο σχολιάζουμε μια προφανή παράλληλη υλοποίηση του αλγορίθμου. Εκτεταμένα πειράματα δείχνουν την αποδοτικότητα του αλγορίθμου Adapt σε σύγκριση με δυο state-of-the art αλγορίθμους.

[1]Με τον όρο περιοχή, εννοούμε την περιοχή έλξης (region of attraction) ενός ελαχίστου που ουσιαστικά είναι όλα τα σημεία του χώρου από τα οποία μια τοπική ελαχιστοποίηση οδηγεί στο ελάχιστο

9ο Κεφάλαιο

Στο 9ο Κεφάλαιο, παρουσιάζεται επίσης μια άμεση υλοποίηση του Γενικού Αλγορίθμου 2 με τον ορισμό μιας συνάρτησης δειγματοληψίας, με σκοπό να παρέχει αρχικά σημεία *μακριά* από ήδη υπάρχοντα ελάχιστα.

Η μέθοδος βασίζεται στον αλγόριθμο δειγματοληψίας με απόρριψη (rejection sampling). Ως βασική κατανομή θεωρείται η ομοιόμορφη. Η συνάρτηση πυκνότητας πιθανότητας από την οποία καλούμαστε να δειγματοληπτήσουμε είναι ένα άθροισμα κανονικών κατανομών

$$F(x) = \sum_{i=1}^{Nlocal} \frac{\rho_i}{\sum_j \rho_j} N(x; \mu_i, \Sigma_i)$$

όπου $Nlocal$ ο αριθμός των τοπικών ελαχίστων που έχουν βρεθεί, ρ_i πόσες φορές έχει βρεθεί το κάθε ελάχιστο και μ_i, Σ_i οι παράμετροι της κατανομής που ανατίθεται σε κάθε τοπικό ελάχιστο. Οι παράμετροι μ_i, Σ_i ενημερώνονται on-line κάθε φορά που το ελάχιστο στο οποίο αντιστοιχούν ανακαλυφθεί για ρ-οστη φορά :

- $\mu_i = \mu_{i-1} + \alpha_i(x_i - \mu_{i-1})$
- $\Sigma_i = \Sigma_{i-1} + \alpha_i(x_i - \mu_i)(x_i - \mu_i)^T$,

όπου $\alpha_i \in (0,1), \mu_0 = (0,0,\ldots,0)^T, \Sigma_0 = \alpha I_n$ και x_i το αρχικό σημείο που οδήγησε στο i-οστό ελάχιστο.

Σύμφωνα με τα πειραματικά αποτελέσματα, η νέα μορφή δειγματοληψίας είναι πολύ αποτελεσματική στο να κατευθύνει τα αρχικά σημεία σε μη εξερευνημένες περιοχές.

10ο Κεφάλαιο

Στο 10ο Κεφάλαιο, προτείνεται ένας νέος αλγόριθμος ομαδοποίησης (Βήμα 2 του Γενικού Αλγορίθμου 1) ο οποίος βασίζεται στη συνεργασία δυο πολύ ισχυρών τεχνικών: στον καθολικό αλγόριθμο k-μέσων (global k-means) και στην θεωρία φασματικής ομαδοποίησης (spectral clustering).

Ο αλγόριθμος παίρνει σαν είσοδο αρχικά ομοιόμορφα σημεία στο χώρο αναζήτησης τα οποία έχουν συγκεντρωθεί με την εφαρμογή αλγορίθμου τοπικής ελαχιστοποίησης (της μορφής του Κεφαλαίου 11). Η προτεινόμενη μέθοδος χρησιμοποιεί τα συγκεντρωμένα σημεία (θέσεις και παραγώγους) για να εκτιμήσει το πλήθος k των ομάδων και στη συνέχεια εφαρμόζει τον αλγόριθμο global k-means ή μια παραλλαγή του που λειτουργεί στο φασματικό χώρο για να υπολογίσει τα κέντρα. Από το κάθε κέντρο μια ολοκληρωμένη τοπική ελαχιστοποίηση θα οδηγήσει σε ένα ελάχιστο.

Βασικό υπολογιστικό κόστος του αλγορίθμου, πέρα από τις κλήσεις της συνάρτησης και της παραγώγου, είναι και ο υπολογισμός των ιδιοτιμών ενός $N \times N$ πίνακα συσχέτισης των N αρχικών σημείων.

Η συνεισφορά της μεθόδου έγκειται (α) στην εφαρμογή τόσο των φασματικών μεθόδων όσο και του αλγορίθμου global k-means στο συγκεκριμένο πρόβλημα (β) στην χρήση της παραγώγου στα συγκεντρωμένα σημεία (στον πίνακα συσχέτισης) για να βοηθηθεί η

ομαδοποίηση (γ) στην απόπειρα εκτίμησης του αριθμού των ομάδων με βάση τη φασματική πληροφορία (δ)στην παραλλαγή του αλγορίθμου global k-means ώστε να λαμβάνει υπόψη τη φασματική πληροφορία.

Ο αλγόριθμος συμπεριφέρεται πολύ καλά πειραματικά και ο αριθμός των τοπικών ελαχιστοποιήσεων τις οποίες ξεκινάει είναι πολύ μικρος. Επίσης, ο αλγόριθμος ομαδοποίησης έχει μόνο μια παράμετρο που πρέπει να ρυθμιστεί από τον χρήστη η οποία είναι ο εκτιμώμενος αριθμός των γειτόνων ανά ομάδα.

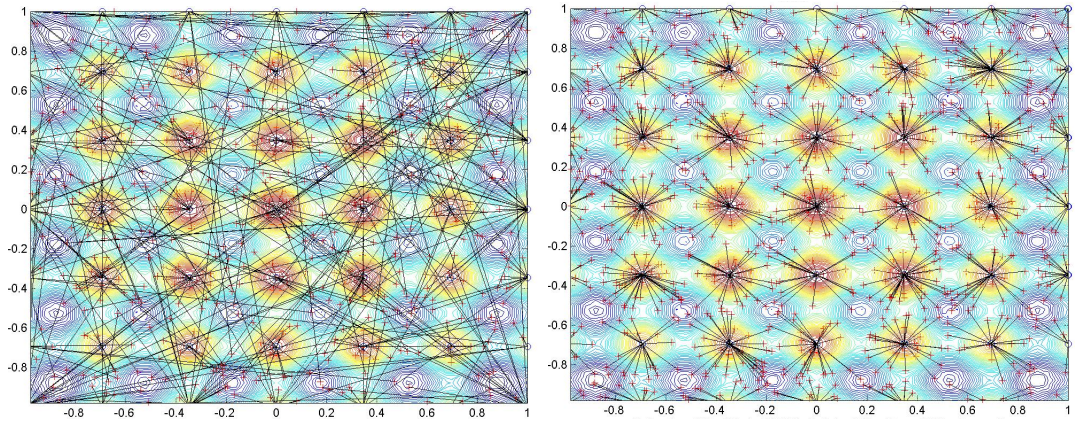
11ο Κεφάλαιο

Στο 11ο Κεφάλαιο της διατριβής παρουσιάζεται ένας αλγόριθμος μονοδιάστατης τοπικής ελαχιστοποίησης (γραμμικής αναζήτησης), ο οποίος είναι κατάλληλος για τη μέθοδο τοπικής ελαχιστοποίησης και στα δυο γενικά αλγοριθμικά πλαίσια. Ο αλγόριθμος ανήκει στην κατηγορία των μεθόδων με γραμμική αναζήτηση. Μπορεί να χρησιμοποιήσει οποιαδήποτε προσέγγιση του εσσιανού πίνακα (BFGS, DFP, πραγματικό εσσιανό) και εξασφαλίζει επαρκή πτώση σε κάθε επανάληψη με μια τροποποίηση του γνωστού αλγορίθμου γραμμικής αναζήτησης με οπισθοχώρηση (backtracking line search).

Η διαφορά του προτεινόμενου αλγορίθμου γραμμικής αναζήτησης έγκειται στο ότι, προσπαθεί να εντοπίσει το *κοντινότερο* δυνατό ελάχιστο στο αρχικό σημείο και όχι *ένα τυχαίο* ελάχιστο. Συγκεκριμένα, ο αλγόριθμος γραμμικής αναζήτησης επιχειρεί να καταλήξει στο ίδιο ελάχιστο που θα κατέληγε και μονοδιάστατη αναζήτηση με απειροελάχιστο βήμα κατά την αντίθετη διεύθυνση της παραγώγου. Ζητάμε λοιπόν έναν αλγόριθμο γραμμικής αναζήτησης που να εξασφαλίζει απόλυτη πτώση (strictly descent) ή όπως την περιγράφουν οι Rinnoy-Kan και Timmer:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k p_k, \text{ με } \alpha_k \text{ τέτοιο} \\ f(x_k + \beta p_k) &\leq f(x_k + \gamma p_k), \quad \forall \ 0 \leq \gamma \leq \beta \leq \alpha_k \end{aligned}$$

Η απαίτηση αυτή εξυπηρετούσε τις θεωρητικές αποδείξεις στις πρωτοπόρες εργασίες σχετικά με τους αλγορίθμους ομαδοποίησης. Πάρόλα αυτά Δεν έχει παρουσιαστεί στη βιβλιογραφία, καμία μέθοδος που να υλοποιεί αυτήν την απαίτηση. Το παρακάτω σχήμα, παρουσιάζει τα αρχικά και τελικά σημεία τοπικών ελαχιστοποιήσεων από ομοιόμορφα τυχαία αρχικά σημεία, χρησιμοποιώντας μια BFGS στο αριστερό σχήμα και τον αλγόριθμο γραμμικής αναζήτησης που παρουσιάζεται στο Κεφάλαιο 11 στο δεξί σχήμα:



Από τα παραπάνω σχήματα είναι προφανές ότι ένας αλγόριθμος καθολικής ελαχιστοποίησης που στηρίζεται στον ορισμό της περιοχής έλξης (region off attraction) και στην απόσταση των αρχικών σημείων με τα ελάχιστα που καταλήγουν θα βασιστεί σε παραποιημένες πληροφορίες εάν χρησιμοποιήσει τοπική αναζήτηση που δεν είναι strictly descent. Με βάση λοιπόν την παραπάνω παρατήρηση και την παράλληλη ανάπτυξη παραλλαγών του Γενικού Αλγόριθμου Καθολικής Ελαχιστοποίησης (βλ. Adapt), κρίθηκε επιτακτική η ανάγκη ανάπτυξης ενός αλγορίθμου που από τη μια θα εξασφαλίζει σύγκλιση στο κοντινότερο ελάχιστο και από την άλλη θα είναι αποδοτικός όπως οι κλασσικοί αλγόριθμοι γραμμικής αναζήτησης.

Ο αλγόριθμος γραμμικής αναζήτησης που παρουσιάζεται στο παρών κεφάλαιο, σε αντίθεση με τον κλασικό αλγόριθμο οπισθοχώρησης, επιχειρεί σταδιακά βήματα προς τα εμπρός. Τα βήματα ξεκινάνε από το μικρότερο δυνατό και μεγαλώνουν σταδιακά μέχρι είτε να φτάσουμε στο μέγιστο βήμα (=1), είτε να αρχίσει να αυξάνεται η τιμή της συνάρτησης.

Ο αλγόριθμος συγκρίνεται πειραματικά κατ' αρχήν για να ελεγχθεί η αποδοτικότητά του, δηλαδή πόσο κοστίζει υπολογιστικά μια τέτοια απαίτηση και κατά δεύτερον για να προβληθεί η χρησιμότητα στο πλαίσιο ενός αλγορίθμου καθολικής ελαχιστοποίησης. Ταυτόχρονα, προτείνεται η παραλληλοποίηση του αλγορίθμου που θα μείωνε το κόστος του.

12ο Κεφάλαιο

Στο τελευταίο Κεφάλαιο της διατριβής παρουσιάζεται μια μελέτη ενός νέου κριτηρίου τερματισμού με εφαρμογή σε όλες τις στοχαστικές μεθόδους με δυο φάσεις και χρήση ομαδοποίησης. Το κριτήριο τερματισμού βασίζεται στην υπόθεση ότι κάθε τοπικό ελάχιστο έχει την ίδια πιθανότητα να ανακτηθεί από μια τυχαία ομοιόμορφη δειγματοληψία, ή αλλιώς οι περιοχές έλξης γύρω από όλα τα ελάχιστα έχουν (περίπου) το ίδιο μέγεθος. Σίγουρα η απαίτηση αυτή μπορεί να φαίνεται πολύ περιοριστική. Σκοπός, όμως, των μεθόδων ομαδοποίησης είναι να μην επιτρέπουν την επαναληπτική εύρεση των ίδιων ελαχίστων ανεξάρτητα από το μέγεθος των περιοχών έλξης. Με τον τρόπο αυτό όλα τα ελάχιστα ανακτώνται ομοιόμορφα.

Το κριτήριο τερματισμού προήλθε από το παρακάτω πείραμα:

Ας υποθέσουμε ότι έχουμε ένα κουτί με w διαφορετικές σφαίρες, αριθμημένες συνεχόμενα $1, 2, \ldots w$. Επιλέγουμε μια σφαίρα τυχαία, σημειώνουμε τον αριθμό της και την τοποθετούμε

στο κουτί. Αυτό θεωρείται μια επανάληψη. Αν ο αριθμός της σφαίρας εμφανίζεται για πρώτη φορά αυξάνουμε τον αριθμό m των διαφορετικών που έχουμε βρει

Ας υποθέσουμε ότι στην k επανάληψη, η πιθανότητα m σφαίρες (ελάχιστα) να έχουν βρεθεί είναι $p_m^{(k)}$. Τότε, η αναμενόμενη τιμή των διαφορετικών σφαιρών θα δίνεται από:

$$< N >^{(k)} = \sum_{i=1}^{k} i \cdot p_i^{(k)} = p_1^{(k)} + 2p_2^{(k)} + \cdots + kp_k^{(k)}$$

όπου η πολύ σημαντική πιθανότητα $p_m^{(k)}$ θα υπολογίζεται από τον αναδρομικό τύπο:

$$p_i^{(k+1)} = p_\alpha p_i^{(k)} + p_\beta p_{i-1}^{(k)}$$

Η παραπάνω αναδρομική σχέση ανάγεται στο εξής: Η πιθανότητα στην $k+1$ επανάληψη να έχουν ανακαλυφθεί i σφαίρες συσχετίζεται με

- την πιθανότητα στην k επανάληψη να έχουν ήδη βρεθεί i διαφορετικές σφαίρες και στην επόμενη να *μη βρεθεί* καμία καινούργια (και αυτό με πιθανότητα p_α)
- την πιθανότητα στην k επανάληψη να έχουν ήδη βρεθεί $i-1$ διαφορετικές σφαίρες και στην επόμενη να βρεθεί μία ακόμα καινούργια σφαίρα (και αυτό με πιθανότητα p_β)

Το πρόβλημα τώρα ανάγεται στον καθορισμό των πιθανοτήτων p_α και p_β. Με την υπόθεση ότι κάθε σφαίρα μπορεί να ανακτηθεί με ίση πιθανότητα με όλες τις άλλες έχουμε

Η πιθανότητα να επιλέξουμε μια από τις i ανακτηθείσες w σφαίρες να είναι $p_\alpha = \frac{i}{w}$.

Η πιθανότητα μην επιλέξουμε μια καινούργια σφαίρα $p_\beta = \frac{w-(i-1)}{w}$.

Αν στην παραπάνω ανάλυση θεωρήσουμε αντί για σφαίρες τοπικά ελάχιστα και αντιστοιχήσουμε την επιλογή με τοπική ελαχιστοποίηση, προκύπτει ότι $< N >^{(k)}$ θα είναι ο αναμενόμενος αριθμός τοπικών ελαχιστοποιήσεων. Το κριτήριο τερματισμού θα ορίζεται με βάση τη διακύμανση της ποσότητας $< N >^{(k)}$ και τον πραγματικό αριθμό ελαχιστοποιήσεων. Δηλαδή, κάποια στιγμή που δεν θα ανακαλύπτονται πλέον ελάχιστα η πειραματική εκτίμηση του αριθμού των τοπικών ελαχιστοποιήσεων θα ταυτίζεται με τη θεωρητική $< N >^{(k)}$.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Chapter 1

Introduction to Optimization

In recent years, the field of Optimization, has undergone a rapid development. This is mainly due to the fact that optimization has found applications in many interesting areas of science and technology, including molecular biology, imaging, digital signal processing, portfolio management, networks and more. Another important factor that favored this development is the tremendous growth in computing power that we have witnessed in our times. Methods that were once considered inapplicable, due to the long CPU times they required, nowadays may be common practice, since the CPU times may have dropped a few orders of magnitude. Many different methods have been designed to treat a host of diverse classes of problems. For example we have continuous and discrete problems, constrained and unconstrained, and so on so forth. There are a great many applications that can be formulated as continuous optimization problems; for instance,

- finding the optimal trajectory for an aircraft or a robot arm;
- identifying the seismic properties of a piece of the earths crust by fitting a model of the region under study to a set of readings from a network of recording stations;
- designing a portfolio of investments to maximize expected return while maintaining an acceptable level of risk;
- controlling a chemical process or a mechanical device to optimize performance or meet standards of robustness;
- computing the optimal shape of an automobile or aircraft component;
- identifying parameters in machine learning problems

1.1 Mathematical formulation

Mathematically speaking, optimization is the minimization or maximization of a function subject to constraints on its variables. We use the following notation:

x is the vector of variables, also called unknowns or parameters;

f is the objective function, a function of x that we want to maximize or minimize;

c is the vector of constraints that the unknowns must satisfy. This is a vector function of the variables x.The number of components in c is the number of individual restrictions that we place on the variables.

The optimization problem may be stated as

$$\min_{x \in \mathcal{R}^n} f(x) \quad \text{subject to } x \in S \tag{1.1}$$

where S is a compact subset or $\mathcal{R}^n$. However in the optimization literature the set S is preferably represented by two set of nonlinear (in general) functions. The first set imposes equality and the second inequality constraints. Hence, the general optimization problem is now transformed into:

$$\min_{x \in \mathcal{R}^n} f(x) \text{ subject to } \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}. \end{cases} \tag{1.2}$$

Here f and each c_i are scalar-valued functions of the variables x, and $\mathcal{I}, \mathcal{E}$ are sets of indices. The set $S = \{\ x \mid c_i(x) = 0,\ i \in \mathcal{E} \text{ and } c_i(x) \geq 0,\ i \in \mathcal{I}\}$ is known as *feasible set.*

As a simple example consider the problem

$$\min_{x \in \mathcal{R}^n} (x_1 - 2)^2 + (x_1 - 1)^2 \text{ subject to } \begin{cases} x_1^2 - x_2 \leq 0, \\ x_1 + x_2 \leq 2. \end{cases} \tag{1.3}$$

We can write the problem 1.3 in the form of 1.2 by defining

$$\begin{aligned} f(x) &= (x_1 - 2)^2 + (x_1 - 1)^2, x = (x_1, x_2)^T \\ c(x) &= \begin{bmatrix} -x_1^2 + x_2 \\ -x_1 - x_2 + 2 \end{bmatrix}, \quad \mathcal{I} \in \{1, 2\}, \quad \mathcal{E} = \{\} \end{aligned}$$

Figure 1.1 illustrates the contours of the function $f(x)$ and also shows the *feasible region* which is the set of points satisfying all the constraints (The "infeasible side" of the inequality constraints is shaded).

1.2 Continuous Versus Discrete Optimization

The generic term *discrete optimization* usually refers to problems in which the solution we seek is one of a number of objects in a finite set. By contrast, *continuous optimization* problems find a solution from an uncountably infinite set off typically a set of vectors with real components. Continuous optimization problems are normally easier to solve, because the smoothness of the functions makes it possible to use objective and constraint

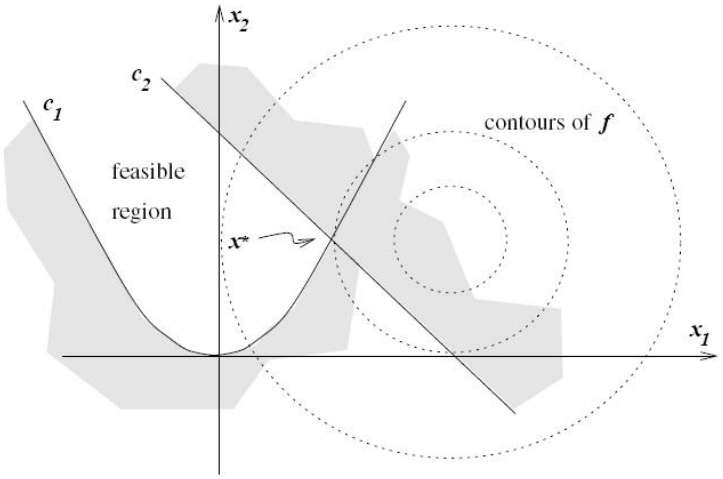


Figure 1.1: Geometrical representation of a general optimization problem

information at a particular point x to deduce information about the functions behavior at all points close to x. The same statement cannot be made about discrete problems, where points that are "close" in some sense may have markedly different function values. Moreover, the set of possible solutions is too large to make an exhaustive search for the best value in this finite set. Some models contain variables that are allowed to vary continuously and others that can attain only integer values; we refer to these as mixed integer programming problems.

1.3 Constrained And Unconstrained Optimization

the objective function and constraints (linear, nonlinear, convex), the number of variables (large or small), the smoothness of the functions (differentiable or non differentiable), and so on. Possibly the most important distinction is between problems that have constraints on the variables and those that do not. This book is divided into two parts according to this classification. Unconstrained optimization problems arise directly in many practical applications. If there are natural constraints on the variables, it is sometimes safe to disregard them and to assume that they have no effect on the optimal solution. Unconstrained problems arise also as reformulations of constrained optimization problems, in which the constraints are replaced by penalization terms in the objective function that have the effect of discouraging constraint violations. Constrained optimization problems arise from models that include explicit constraints on the variables. These constraints may be simple bounds such as $0 \leq x1 \leq 100$, more general linear constraints such as $\sum_i x_i \leq 1$, or nonlinear inequalities that represent complex relationships among the variables.

1.4 Global And Local Optimization

The fastest optimization algorithms seek only a local solution, a point at which the objective function is smaller than at all other feasible points in its vicinity. They do not always find the best of all such minima, that is, the global solution. Global solutions are necessary (or at least highly desirable) in some applications, but they are usually difficult to identify and even more difficult to locate. An important special case is *convex programming*, in which all local solutions are also global solutions. Linear programming problems fall in the category of convex programming. However, general nonlinear problems, both constrained and unconstrained, may possess local solutions that are not global solutions.

1.5 Stochastic and Deterministic Optimization

Global optimization algorithms are usually broadly divided into deterministic and stochastic. Deterministic methods provide a theoretical guarantee of locating the global minimum, or at least a local minimum whose objective function value differs by at worst ϵ from the global one for a given $\epsilon > 0$. Stochastic methods only offer a guarantee in probability. On the other hand, stochastic methods are usually faster in locating a global optimum than deterministic ones. Moreover, stochastic methods adapt better to black-box formulations and extremely ill-behaved functions, whereas deterministic methods usually rest on at least some theoretical assumptions about the problem formulation and its analytical properties. Comparisons between deterministic and stochastic global optimization methods are scarce in the literature.

1.6 Optimality Conditions

In this section we provide the mathematical formulations for optimality conditions for both constrained and unconstrained optimization problems. These conditions will be continuously reference throughout this thesis.

1.6.1 Unconstrained Problems

The unconstrained optimization problem can be stated as:

$$\min_x f(x) \quad x \in R^n \tag{1.4}$$

We provide the necessary definitions of what is a minimum for problem in Eq 1.4.

Definition 1.1. A point x^* is a *global minimizer* if $f(x^*) \leq f(x)$ for all $x \in \mathcal{R}^N$.

Definition 1.2. A point x^* is a *local minimizer* if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$.

Definition 1.3. A point x^* is a strict local minimizer (also called a strong local minimizer) if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x^*$.

Definition 1.4. A point x^* is an isolated local minimizer if there is a neighborhood $\mathcal{N}$ of x^* such that x^* is the only local minimizer in $\mathcal{N}$.

Definition 1.5. We call x^* a stationary point if $\nabla f(x^*) = 0$.

In the following theorems we state the necessary and the sufficient conditions for the existence of a local minimizer. Necessary conditions for optimality are derived by assuming that x^* is a local minimizer and then proving facts about $\nabla f(x^*)$ and $\nabla^2 f(x^*)$.

Theorem 1.1 (First order necessary conditions). *If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.*

Theorem 1.2 (Second order necessary conditions). *If x^* is a local minimizer of f and $\nabla^2 f$ is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.*

We now describe sufficient conditions, which are conditions on the derivatives of f at a point x^* that guarantee that x^* is a local minimizer.

Theorem 1.3 (Second order sufficient conditions). *Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.*

Theorem 1.4. *When f is convex, any local minimizer x^* is a global minimizer of f. In addition if f is differentiable, then any stationary point x^* is a global minimizer of f.*

1.6.2 Constrained Problems

Consider now the general problem of Eq 1.1 using the formulation of Eq 1.2. In this Section we provide the mathematical characterizations of the solutions of Eq 1.2. Recall that for the unconstrained optimization problem of previous Section, we characterized solution points x^* in the following way:

- Necessary conditions: Local minima of unconstrained problems have $\nabla f(x^*) = 0$ and $\nabla f^2(x^*)$ positive semidefinite.
- Sufficient conditions: Any point x^* at which $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite is a strong local minimizer of f.

Our aim in this chapter is to derive similar conditions to characterize the solutions of constrained optimization problems.

Definition 1.6. A vector x^* is a local solution of the problem Eq 1.1 if $x^* \in \mathcal{S}$ and there is a neighborhood N of x^* such that $f(x^*) \leq f(x)$ for $x \in N \cap S$.

Definition 1.7. A vector x^* is a strict local solution (also called a strong local solution) if $x^* \in S$ and there is a neighborhood N of x^* such that $f(x^*) < f(x)$ for all $x \in N \cap S$ with $x \neq x^*$.

Definition 1.8. A point x^* is an isolated local solution if $x \in S$ and there is a neighborhood N of x^* such that x^* is the only local minimizer in $N \cap S$.

Using the formulation of Eq 1.2 we can now provide formal definitions for necessary and sufficient optimality conditions.

Definition 1.9. The *Lagangian function* for the constrained optimization problem of Eq 1.2 is defined as:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

Definition 1.10. The *active set* $A(x)$ at any feasible point x is the union of the set $\mathcal{E}$ with the indices of the active inequality constraints; $A(x) = \mathcal{E} \cup \{i \in I \mid c_i(x) = 0\}$.

Definition 1.11. Given the point x^* and the active set $A(x^*)$, we say that the *linear independence constraint qualification (LICQ)* holds if the set of active constraint gradients $\{\nabla c_i(x^*),\ i \in A(x^*)\}$ is linearly independent.

The above conditions allows us to state the following optimality conditions for a general nonlinear programming problem of Eq 1.2. These conditions provide the foundation for many of the algorithms presented in the bibliography. They are called first-orderconditions because they concern themselves with properties of the gradients (first-derivative vectors) of the objective and constraint functions.

Theorem 1.5. *First-Order Necessary Conditions Suppose that x^* is a local solution of Eq 1.2 and that the LICQ holds at x^* . Then there is a Lagrange multiplier vector λ^*, with components $\lambda_i^*, i \in \mathcal{E} \cup \mathcal{I}$, such that the following conditions are satisfied at (x^*, λ^*)*

$$\begin{aligned}
\nabla \mathcal{L}(x^*, \lambda^*) &= 0, && (1.5)\\
c_i(x^*) &= 0, \quad \textit{for all } i \in \mathcal{E}, && (1.6)\\
c_i(x^*) &\geq 0, \quad \textit{for all } i \in \mathcal{I}, && (1.7)\\
\lambda_i^* &\geq 0, \quad \textit{for all } i \in \mathcal{I}, && (1.8)\\
\lambda_i^* c_i(x^*) &= 0, \quad \textit{for all } i \in \mathcal{E} \cup mathcal I. && (1.9)
\end{aligned}$$

The conditions described in Eq 1.5-1.9 are also known as *Karush-Kuhn-Tucker* conditions, or *KKT conditions* for short.

In order to derive the Second Order Optimality Conditions for the general constrained case we need some more definitions.

Definition 1.12. Given a point x^* and the active constraints set $A(x^*)$, we define the set F_1 as

$$F_1 = \left\{ \alpha d \mid \alpha > 0, \begin{array}{c} d^T \nabla c_i(x^*) = 0, \text{ for all } i \in \mathcal{E}, \\ d^T \nabla c_i(x^*) \geq 0, \text{ for all } i \in A(x^*) \cap \mathcal{I} \end{array} \right.$$

Note that F_1 is the tangent cone to the feasible set at x^*.

Given F_1 from Definition 1.12 and some Lagrange multiplier vector λ^* satisfying the KKT conditions we define a subset $F_2(\lambda^*)$ of F_1 by

Definition 1.13.

$$F_2(\lambda^*) = \{w \in F_1 \mid \nabla c_i(x^*)^T w = 0, \text{ for all } i \in A(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0\}$$

or alternativly

$$w \in F_2(\lambda^*) \equiv \left\{ \begin{array}{ll} \nabla c_i(x^*)^T w = 0 & \text{for all } i \in \mathcal{E}, \\ \nabla c_i(x^*)^T w = 0 & \text{for all } i \in A(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0, \\ \nabla c_i(x^*)^T w \geq 0 & \text{for all } i \in A(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* = 0. \end{array} \right.$$

The subset $F2(\lambda^*)$ contains the directions w that tend to the active inequality constraints for which the Lagrange multiplier component λ^* is positive, as well as to the equality constraints.

Theorem 1.6 (Second-Order Necessary Conditions). *Suppose that x^* is a local solution of problem of Eq 1.2 and that the LICQ condition is satisfied. Let λ be a Lagrange multiplier vector such that the KKT conditions (Eqs 1.5-1.9) are satisfied, and let $F2(\lambda^*)$ be defined as above. Then*

$$w^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*) w \geq 0, \quad \textit{for all } w \in F2(\lambda^*). \tag{1.10}$$

Theorem 1.7 (Second-Order Sufficient Conditions). *Suppose that x^* is a local solution of problem of Eq 1.2 and that the LICQ condition is satisfied. Let λ be a Lagrange multiplier vector such that the KKT conditions (Eqs 1.5-1.9) are satisfied, and let $F2(\lambda^*)$ be defined as above. Then*

$$w^T \nabla_{xx} \mathcal{L}(x^*, \lambda^*) w > 0, \quad \textit{for all } w \in F2(\lambda^*), w \neq 0. \tag{1.11}$$

1.7 Optimization Algorithms

Optimization algorithms are iterative. They begin with an initial guess of the optimal values of the variables and generate a sequence of improved estimates until they reach a solution. The strategy used to move from one iterate to the next distinguishes one algorithm from another. Most strategies make use of the values of the objective function f, the constraints c, and possibly the first and second derivatives of these functions. Some algorithms accumulate information gathered at previous iterations, while others

use only local information from the current point. Regardless of these specifics (which will receive plenty of attention in the rest of the book), all good algorithms should possess the following properties:

- Robustness. They should perform well on a wide variety of problems in their class, for all reasonable choices of the initial variables.
- Efficiency. They should not require too much computer time or storage.
- Accuracy. They should be able to identify a solution with precision, without being overly sensitive to errors in the data or to the arithmetic rounding errors that occur when the algorithm is implemented on a computer.

1.8 Parallel Computation

Parallel and distributed computation is having a significant impact upon how large scale scientific computation is performed. To solve the large numerical problems of interest to scientists and engineers today, very powerful computers are necessary, and it appears that many if not all of the most powerful scientific computers of the future (as well as at present) will be parallel and distributed computers. To numerical computation researchers, one of the most interesting aspects of the transition from sequential or vector computers to parallel and distributed computers is that new algorithm development may be required to use the new machines efficiently.

Parallelism is of interest in optimization because many optimization problems are expensive to solve. To understand why this is so, one needs at least the following rudimentary understanding of optimization algorithms. All algorithms for solving optimization problems are iterative. Each iteration involves at least one evaluation of the nonlinear functions ($f(x)$ or $c_i(x)$), and in many cases, of their first (and occasionally second) derivatives. In addition, each iteration involves linear algebraic computations, which generally require $0(n^2)$ or $O(n^3)$ arithmetic operations in the case of problems with a small to moderate (say less than 100) number of variables, and $O(n)$ or $O(n^2)$ arithmetic operations in the case of problems with a larger number of variables.

The main expenses in optimization algorithms, then, can come from at least four possible sources:

1. The nonlinear functions, constraints, and/or derivatives may be expensive to evaluate.
2. The number of variables or constraints, and hence the cost of each iteration aside from function and derivative evaluation, may be large.
3. Many evaluations of the objective function, constraints, or derivatives may be required.

4. Many iterations may be required.

These sources of computational expense in turn lead to three levels at which one may consider introducing parallelism into an optimization algorithm, and which together encompass the general possibilities for utilizing parallelism in optimization:

1. Parallelize the individual evaluations of the objective function, constraints, and/or their derivatives.

2. Parallelize the linear algebra involved in each iteration.

3. Parallelize the optimization process at a high level, either to perform multiple function, constraint, and/or derivative evaluations on multiple processors concurrently, and/or to reduce the total number of iterations required.

The first possibility of parallelization is the most common one, although each researcher creates its own custom code and there is a lack of general purpose high performance code in the literature. *In this thesis we present such an implementation which takes fully advantage of modern parallel architectures.* The second possibility, parallelizing the linear algebraic calculations, may be the concern of optimization researchers if the linear algebra is particular to optimization algorithms. Parallelizing the optimization process in a nonlinear optimization algorithm is likely to lead to a ôcoarse-grainő parallel algorithm. By this we mean an algorithm where each processor performs a significant amount of computation in between each point where it communicates or synchronizes with other processors. For example, if each processor performs at least one function evaluation between communication points, and these function evaluations are even moderately expensive, a coarse-grain parallel algorithm results. Such parallel algorithms are generally well suited to MIMD computers, including shared memory multiprocessors, virtual shared memory multiprocessors, distributed memory multiprocessors, and networks of computers used as multiprocessors. *A basic part of this thesis is the implementation of parallel algorithms both for local and global optimization*

1.8.1 Parallelizing local search

When the number of variables is not too large, say less than 500, unconstrained optimization problems are generally solved either by approaches built around Newtonős method, or by "quasi-Newton" methods. Approaches built around Newtonős method require the *nxn Hessian matrix* $\nabla^2 f(x)$ of second partial derivatives of $f(x)$ to be calculated at each iteration, and the solution of an *nxn* system of linear equations involving this matrix at each iteration. Since both of these requirements can be quite expensive, especially when function and derivative evaluation is expensive or n is not very small, it is more common to use quasi-Newton methods, which avoid these costs, to solve unconstrained optimization problems with small to moderate numbers of variables. Quasi-Newton methods maintain a rough approximation to the Hessian matrix, which is updated after each iteration using information about the step and gradient values at that iteration.

The basic sketch of a Newton/quasi-Newton algorithm is presented bellow:

Algorithm 1.2 Newton / quasi-Newton framework

(1) Check the termination criteria in order to stop or not.

(2) Solve $\boldsymbol{B}^{(k)}\boldsymbol{d}^{(k)} = -\boldsymbol{g}^{(k)}$ for $\boldsymbol{d}^{(k)}$ ($B^{(k)} = \nabla^2 f(x^{(k)})$ for Newton's method)

(3) Apply a selection method to obtain a *better* point $\boldsymbol{x}^{(k+1)}$ (and calculate $f(\boldsymbol{x}^{(k+1)})$) using $\boldsymbol{d}^{(k)}$

(4) Calculate the gradient vector $\boldsymbol{g}^{(k+1)}$.

(5) Update $\boldsymbol{B}^{(k)}$ to $\boldsymbol{B}^{(k+1)}$ using a quasi-Newton formula.

The main opportunities for using parallelism in existing or new quasi-Newton methods correspond to the three general uses of parallelism in optimization algorithms that were mentioned in the previous Section.

- One can parallelize the individual evaluations of $f(x)$ or $\nabla f(x)$ in steps (3) and (4) above.
- One can parallelize the linear algebraic calculations in steps (2) and (4) above.
- One can perform multiple evaluations of $f(x)$ or $\nabla f(x)$ concurrently, either within the algorithmic framework above, or by devising new algorithms for step (3).

Part II

Local Optimization

Chapter 2

Survey on Local Optimization

There is a vast literature on local optimization methods. Excellent detailed descriptions can be found in the books by Fletcher [40], Dennis & Schnabel [35], Gill, Murray & Wright [53], Nash & Sofer [108] to name a few. Here we shall give a very brief introduction to the subject of "Unconstrained Local Optimization" for reasons of self-containment. The local optimization problem can be stated as:

$$Find \;\; x^* \in R^n, such\ that:\ f(x^*) \leq f(x), \;\; \forall x \in D^n \tag{2.1}$$

where $D^n \subset R^n$ is a small neighborhood around x^*. The necessary conditions at the minimum are given by:

$$\nabla f(x^*) = 0, \text{ and } \;\; \nabla^2 f(x^*) \geq 0 labelneces \tag{2.2}$$

while the sufficient conditions by:

$$\nabla f(x^*) = 0, \text{ and } \;\; \nabla^2 f(x^*) > 0 \tag{2.3}$$

We refer to $f(x)$ by the term *objective function*, and to he minimizing point x^* by the term *minimizer*. Often we may use the notation:

$$\begin{aligned} f^* &= f(x^*), \quad f^{(k)} = f(x^{(k)}) \\ g(x) &= \nabla f(x), \quad g^{(k)} = g(x^{(k)}) \\ G(x) &= \nabla^2 f(x), \quad G^* = G(x^*), \quad G^{(k)} = G(x^{(k)}) \end{aligned}$$

There is not a single optimization method appropriate for all problems. The choice of the proper method is highly dependent on the problem itself. Hence in practice, given a problem, it is important to have the ability to predict which method will perform well. To develop such a predictive sense, the understanding of the underlying theory in addition to experience acquired through usage, is required. Therefore, a classification of methods based on the theoretical specifics can be very useful.

2.1 Classification of Methods

An obvious and meaningful way to classify optimization methods, is according to the functional information they use, i.e. if they use function values, first derivatives, second derivatives etc.

- There are methods that only use function values (Direct search methods). These methods are usually appropriate for problems where the objective function is not smooth or contains noise.

- Methods that in addition to function values, make use of the gradient vector are proper for smooth objectives that are continuously differentiable. However the performance of these methods may deteriorate, if the gradient is not analytically available and finite differences are employed to estimate it numerically. In an effort to overcome this difficulty, automatic differentiation methods, based on a generalized chain–rule technique, have been developed [61]. These methods, accept as input the code for the objective function and produce as output the code for its gradient. (No finite differencing is involved).

- Another type of methods are those that make use of higher derivatives. The most popular are the ones based on Newton's method that employ the Hessian matrix, i.e. the matrix of the second derivatives. Again these methods are proper for smooth and twice continuously differentiable objective functions. Numerical estimation of the Hessian may have disastrous effects on their performance. Automatic differentiation methods are again applicable.

A different classification scheme may be based on the dimensionality of the problem. Most methods are designed for small to medium sized problems. Large scale problems need special treatment. While the theoretical background remains the same, practical considerations, such as the ratio of the CPU times spent for the bookkeeping operations and for evaluating the function, play an important role. Since large problems often lead to large and sparse linear systems, iterative methods enter the picture to take advantage of the sparsity structure. For large problems, it is not clear if solving the intermediate subproblems exactly is worthwhile. In fact it has been observed, that if avoided at the early stages of the procedure, significant time savings may be obtained.

A common ingredient of many optimization algorithms is the *"line–search"* procedure, i.e. a search along a direction in the n-dimensional space. This actually is a univariate optimization procedure and plays an important role both in theory and in practice. One then can divide the various multidimensional optimization methods in those that do perform line-searches and to those that do not. In reviewing the local optimization methods, we will follow the classification we mentioned first.

2.2 Direct Search Methods

There are various methods that belong to this class. We refer to [146] for an extended survey. For a more recent exposition see the book by Bazaraa et al [5]. Here we briefly describe the *"Hooke and Jeeves"* method, the *"Roll"* method and the *"Simplex"* or *" Polytope"* method. We proceed with a discussion of line-search methods without derivatives, that are appropriate for use in conjunction with direct search methods.

2.2.1 One dimensional minimization

There are many methods serving this one-dimensional task. To name a few we mention the Fibonacci search, the golden section search, and the quadratic interpolation. Given an interval $[a_1, b_1]$ that brackets a minimum, we will describe the Golden Section method and a method due to Brent [19]. Both methods assume that the function inside the bracket is unimodal.

The Golden Section method

Let $\tau = \frac{\sqrt{5}-1}{2} \approx 0.618$ be the golden section ratio, and $\epsilon > 0$ be a tolerance for the bracket width. The Golden section algorithm follows the steps:

Algorithm 2.3 The Golden Section method

---

1. Set: $c_1 = a_1 + (1-\tau)(b_1 - a_1)$ and $f_c = f(c_1)$
   $d_1 = b_1 - (1-\tau)(b_1 - a_1)$ and $f_d = f(d_1)$

2. **Loop** for $k = 1, 2, \cdots$ **until** $b_{k+1} - a_{k+1} < \epsilon$
   if $f_c < f_d$ **then**
   Set: $a_{k+1} = a_k,\ b_{k+1} = d_k,\ d_{k+1} = c_k$
   $c_{k+1} = a_{k+1} + (1-\tau)(b_{k+1} - a_{k+1})$
   $f_d = f_c,\ f_c = f(c_{k+1})$
   else
   Set: $a_{k+1} = c_k,\ b_{k+1} = b_k,\ c_{k+1} = d_k$
   $d_{k+1} = b_{k+1} - (1-\tau)(b_{k+1} - a_{k+1})$
   $f_c = f_d,\ f_d = f(d_{k+1})$
   end if
   end Loop

The method of Brent

Brent's method locates the minimum λ, within a prescribed tolerance ε. At every iteration j the method keeps track of six points a_j, b_j, u_j, v_j, w_j and λ_j, not necessarily distinct. A minimum always lies in the interval $[a_j, b_j]$.

- λ_j is the point with the least value of f.
- w_j is the point with the next lowest value of f.
- v_j is the previous value of w_j, and u_j is the last point at which f has been evaluated.

Initially $v_1 = w_1 = \lambda_1 = a_1 + \frac{3-\sqrt{5}}{2}(b_1 - a_1)$. The j^{th} iteration is described below. Brent's book [19] presents many algorithms for minimization without using derivatives.

Algorithm 2.4 Brent's method

(1) Test for termination. If $\max(\lambda_j - a_j, b_j - \lambda_j) \leq 2\varepsilon$ then return with λ_j as the approximate position of the minimum.

(2) Calculate p, q, so that $\lambda_j + p/q$ is the turning point of the parabola passing through the points $(v_j, f(v_j))$, $(w_j, f(w_j))$ and $(\lambda_j, f(\lambda_j))$.

(3) Calculate the new point u_{j+1}: Let e be the value of p/q at the second–last cycle. If $|e| \leq \varepsilon$, $q = 0$, $\lambda_j + p/q \notin (a, b)$ or $|p/q| \geq |e|/2$, then take a golden section step, otherwise u_{j+1} is taken to be $\lambda_j + \frac{p}{q}$, except that the distances $|u_{j+1} - \lambda_j|$, $u_{j+1} - a_j$ and $b_j - u_{j+1}$ must be at least ε.

(4) Evaluate f at the new point u_{j+1}.

(5) Update the points a_j, b_j, v_j, w_j and λ_j as necessary.

Hence, it would be extremely useful to anyone who deals frequently with problems that are connected with non-smooth objective functions.

2.2.2 Multidimensional Optimization

The Hooke and Jeeves method

This method [72], performs two types of moves. One is of exploratory nature, while the other is a pattern search. Initially, a *base point* $\mathbf{b_1}$ is chosen, along with steps s_i for the corresponding parameters x_i, and the function $f(\mathbf{b_1})$ is evaluated. Then a sequence of exploration and pattern moves follow.

The Roll method

This method [114] also belongs to the class of pattern search methods. It proceeds by exploring the local topology of the objective function and taking proper steps along each direction separately. In that it resembles the obvious (and ad–hoc) alternating variables method [40]. When however the correlation among the variables becomes important, this procedure cannot proceed further. In order to cure this problem the method performs a line search along a properly formed direction at the end of each cycle.

Algorithm 2.5 Hooke and Jeeves method

Exploration: The purpose of the exploratory phase is to acquire information about $f(\boldsymbol{x})$ around the current base point. This is described next.

1. Evaluate $f(\mathbf{b_1} + \mathbf{s_1}\hat{\mathbf{e}}_\mathbf{1})$, where $\hat{e}_i$ is the unit vector along the i^{th} direction.
   If this leads to a lower value, **then**
   accept $\mathbf{b_1} + \mathbf{s_1}\hat{\mathbf{e}}_\mathbf{1}$ as the new base point and replace $\mathbf{b_1}$.
   go to step 2
   end if
   Evaluate $f(\mathbf{b_1} - \mathbf{s_1}\hat{\mathbf{e}}_\mathbf{1})$
   If this leads to a lower value, **then**
   accept $\mathbf{b_1} - \mathbf{s_1}\hat{\mathbf{e}}_\mathbf{1}$ as the new base point and replace $\mathbf{b_1}$.
   go to step 2
   end if
2. Repeat step 1 for the variable $x_2, x_3, \cdots, x_n$, with steps $s_1, s_2, \cdots, s_n$, and arrive at a new base point $\mathbf{b_2}$ after at most $2n + 1$ function evaluations.
3. **If** $\mathbf{b_2} = \mathbf{b_1}$, **then**
   set $h_i = \frac{1}{2}h_i \quad \forall i = 1, 2, \cdots, n$
   If the steps are smaller than a preset limit, **then**
   return $\mathbf{b_1}$ as the minimizer.
   Terminate.
   end if
   go to step 1
   else
   Start a pattern search from $\mathbf{b_2}$
   end if

Pattern Search: Pattern search attempts to take advantage of the information gathered during the exploratory phase by constructing promising search directions. We detail how a pattern move is made from base point $\mathbf{b_2}$.

1. Move to point $\mathbf{p_1} = \mathbf{b_2} + (\mathbf{b_2} - \mathbf{b_1}) = \mathbf{2b_2} - \mathbf{b_1}$, and apply the exploratory procedure around $\mathbf{p_1}$.
2. **If** the lowest function value is lower than $f(\mathbf{b_2})$, **then**
   The corresponding point is the new base point $\mathbf{b_3}$.
   Repeat the previous step with all indices increased by one.
   else
   Abandon the pattern move from $\mathbf{b_2}$ and apply a new sequence of exploratory moves again from $\mathbf{b_2}$.
   end if

Let $\boldsymbol{x}^c = (x_1^c, x_2^c, \ldots, x_n^c)^T$ be the current point and $f_c = f(\boldsymbol{x}^c)$. Let also s_i be a step associated with each variable x_i, and $a > 1$ an acceleration factor. The algorithm executes the following steps $\forall i = 1, 2, \cdots, n$

Algorithm 2.6 Roll algorithm

(1) Pick a trial point: $x_j^t = x_j^c$ for all $j \neq i$ and $x_i^t = x_i^c + s_i$

(2) Calculate $f_+ = f(\boldsymbol{x}^t)$.

(3) if $f_+ < f_c$ set $\boldsymbol{x}^c = \boldsymbol{x}^t$, $f_c = f_+$ and $s_i = as_i$. Then, go to step 8.

(4) if $f_+ \geq f_c$ pick another trial point as :
$x_j^t = x_j^c$ for all $j \neq i$ and $x_i^t = x_i^c - s_i$

(5) Calculate $f_- = f(\boldsymbol{x}^t)$.

(6) if $f_- < f_c$ set $\boldsymbol{x}^c = \boldsymbol{x}^t$, $f_c = f_-$ and $s_i = -as_i$. Then, go to step 8.

(7) if $f_- \geq f_c$ calculate an appropriate step by: $s_i = -\frac{1}{2}\frac{(f_+ - f_-)}{(f_+ + f_- - 2f_c)}s_i$.

(8) Proceed from step 1 for the next value of i.

After looping over all variables, a line search is performed in the direction $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)^T$.
This is the pattern move of the *Roll* method. The above procedure is repeated until a termination criterion applies.

The Simplex method

This method should not be confused with the well known Simplex method of linear programming. In contrast with the previously described direct search methods, this one maintains not just one, but a population of points, a feature that turns out to be important in cases where the objective function contains noise. Originally this algorithm was designed by Spendley et al. [144] and was refined later by Nelder and Mead [109, 110]. A simplex (or Polytope) in R^n is a construct with $(n + 1)$ vertexes defining a volume element. For instance in two dimensions the simplex is a triangle, in three dimensions it is a tetrahedron, and so on so forth. The input to the algorithm apart from a few parameters of minor importance, is an initial simplex. The algorithm brings the simplex in the area of a minimum, adapts it to the local geometry, and finally shrinks it around the minimizer. It is a derivative-free, iterative method that proceeds toward the minimum using a population of $n + 1$ points (the simplex vertexes) and hence it is expected to be tolerant to noise, in spite its deterministic nature. The method executes the following steps (simplex vertexes are denoted by $\boldsymbol{w}_i$).

(1) Examine the termination criteria to decide whether to stop or not.

(2) Number the simplex vertexes $\boldsymbol{w}_i$, so that the sequence $f_i = f(\boldsymbol{w}_i)$ is sorted in ascending order.

(3) Calculate the centroid of the first n vertexes: $\boldsymbol{c} = \frac{1}{n}\Sigma_{i=0}^{n-1}\boldsymbol{w}_i$

(4) Invert the "worst" vertex $\boldsymbol{w}_n$ as: $\boldsymbol{r} = \boldsymbol{c} + \alpha(\boldsymbol{c} - \boldsymbol{w}_n)$ (usually $\alpha = 1$)

(5) If $f_0 \leq f(\boldsymbol{r}) \leq f_{n-1}$ then
set $\boldsymbol{w}_n = \boldsymbol{r}, f_n = f(\boldsymbol{r})$, and go to step 1
end if

(6) If $f(\boldsymbol{r}) < f_0$ then
Expand as: $\boldsymbol{e} = \boldsymbol{c} + \gamma(\boldsymbol{r} - \boldsymbol{c})$ ($\gamma > 1$, usually $\gamma = 2$)
If $f(\boldsymbol{e}) < f(\boldsymbol{r})$ then
set $\boldsymbol{w}_n = \boldsymbol{e}, f_n = f(\boldsymbol{e})$
else
set $\boldsymbol{w}_n = \boldsymbol{r}, f_n = f(\boldsymbol{r})$
end if
go to step 1
end if

(7) If $f(\boldsymbol{r}) \geq f_{n-1}$ then
If $f(\boldsymbol{r}) \geq f_n$ then
contract as: $\boldsymbol{k} = \boldsymbol{c} + \beta(\boldsymbol{w}_n - \boldsymbol{c})$, ($\beta < 1$, usually $\beta = \frac{1}{2}$)
else
contract as: $\boldsymbol{k} = \boldsymbol{c} + \beta(\boldsymbol{r} - \boldsymbol{c})$
end if
If $f(\boldsymbol{k}) < \min\{f(\boldsymbol{r}), f_n\}$, then
set $\boldsymbol{w}_n = \boldsymbol{k}, f_n = f(\boldsymbol{k})$
else
Shrink the whole Polytope as:
Set $\boldsymbol{w}_i = \frac{1}{2}(\boldsymbol{w}_0 + \boldsymbol{w}_i), f_i = f(\boldsymbol{w}_i)$ for $i = 1, 2, \ldots, n$
end if
go to step 1
end if

The initial simplex may be constructed in various ways. One approach is to pick for the first vertex the current point and the rest of the vertexes by line searches originating from the current point and heading along each of the n directions. The second approach picks again for the first vertex the current point and generates the rest by taking a single step along each of the n directions.

2.3 Methods that use the Gradient

These are the most widely used methods. In this class belong the method of *Steepest descent* introduced by Cauchy, the method of *Conjugate gradients* (there are quite a few variants) and most importantly the *Quasi Newton* or alias the *Variable metric* methods.

2.3.1 Steepest descent

This is the simplest of the methods that use gradient information. It is based on the Taylor expansion: $f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}\cdot\nabla f(\boldsymbol{x}) + O(\boldsymbol{h}^2)$. For a step of given length $|\boldsymbol{h}|$ the drop in the function's value $f(\boldsymbol{x}) - f(\boldsymbol{x}+\boldsymbol{h})$ becomes maximum when the angle between vectors $\boldsymbol{h}$ and $\nabla f(\boldsymbol{x})$ equals π. Hence given an initial point $\boldsymbol{x}^{(1)}$ and a small positive tolerance ϵ for the gradient, the suggested algorithm is given by:

Loop for $k = 1, 2, \cdots$ until $|\nabla f(\boldsymbol{x}^{(k)})| < \epsilon$
  Perform a line search along the direction: $s^{(k)} = -\nabla f(\boldsymbol{x}^{(k)})$
  and obtain so $\boldsymbol{x}^{(k+1)}$
End Loop

2.3.2 Conjugate gradient methods

These, as well as the Quasi Newton methods, are based on a quadratic model for the objective function. In the neighborhood of point $\boldsymbol{x}$ we may expand:

$$f(\boldsymbol{x}+\boldsymbol{h}) \approx Q(\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^T\boldsymbol{g}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{h}^T G(\boldsymbol{x})\boldsymbol{h} \tag{2.4}$$

The conjugate gradient methods are creating conjugate directions that are linearly independent and solve the quadratic problem (i.e. *minimize* $Q(\boldsymbol{h})$ with respect to $\boldsymbol{h}$). A set of vectors $s_i, \quad i = 1, 2, \cdots, n$ is said to be mutually conjugate with respect to a positive definite matrix G, if and only if:

$$s_k^T G s_j = 0, \quad \forall \quad k \neq j \tag{2.5}$$

Since G is assumed positive definite, then for $k = j$ the above quantity is positive, for all $s_j \neq 0$. The conjugate gradient methods rely on the fact that given a quadratic function Q with positive definite Hessian matrix G, then the exact minimum may be found by performing exact line searches along the directions $s_i, \quad i = 1, 2, \cdots, n$ that are mutually conjugate with respect to G. The way these vectors are constructed is important. For example the set of the Hessian eigenvectors form such a set. However diagonalizing the Hessian matrix is not practical, since both the memory requirement and the computational effort are excessive. The advantage of conjugate gradient methods is that they construct these directions with minimal computational effort and without using the Hessian matrix explicitly and so the memory requirement for storing an $n \times n$ matrix is relaxed. The conjugate gradient methods are economical in computer memory since they require only

a few arrays of n–elements each. The backbone algorithm for the Fletcher–Reeves [42], Polak–Ribiere [123] and Hestenes–Stiefel [69] methods is described below:

Initially given a point $\boldsymbol{x}^{(1)}$ we set $^{(1)} = -\boldsymbol{g}^{(1)}$. The k^{th} iteration consists of the following steps:

(1) Perform a line search along $^{(k)}$ and obtain $\boldsymbol{x}^{(k+1)}$.

(2) Check the termination criteria in order to stop or not.

(3) Calculate the gradient vector $\boldsymbol{g}^{(k+1)}$.

(4) Calculate a scalar $\beta^{(k)}$ using one of the following prescriptions:

(a) Fletcher–Reeves: $\beta^{(k)} = \frac{\boldsymbol{g}^{(k+1)^T}\boldsymbol{g}^{(k+1)}}{\boldsymbol{g}^{(k)^T}\boldsymbol{g}^{(k)}}$

(b) Polak–Ribiere: $\beta^{(k)} = \frac{(\boldsymbol{g}^{(k+1)}-\boldsymbol{g}^{(k)})^T\boldsymbol{g}^{(k+1)}}{\boldsymbol{g}^{(k)^T}\boldsymbol{g}^{(k)}}$

(c) Hestenes–Stiefel: $\beta^{(k)} = \frac{(\boldsymbol{g}^{(k+1)}-\boldsymbol{g}^{(k)})^T\boldsymbol{g}^{(k+1)}}{(\boldsymbol{g}^{(k+1)}-\boldsymbol{g}^{(k)})^T\ \ ^{(k)}}$

(5) Calculate a new search direction as: $^{(k+1)} = -\boldsymbol{g}^{(k+1)} + \beta^{(k)}\ \ ^{(k)}$

A note must be made here in order to stress that the incorporated line search must be an accurate one, otherwise the algorithm may converge in a slow rate. Among the variants the one most widely used is that of Fletcher–Reeves. However there is evidence that the Polak–Ribiere algorithm has an advantage. It has been argued that when $s^{(k)}$ becomes almost orthogonal to $g^{(k+1)}$, then very little progress is made and the Polak–Ribiere β becomes zero, hence reseting to the steepest descent direction, a fact that enhances its performance. Conjugate gradient methods were used for large problems due to the fact that require only a few vectors of n elements each. However nowadays the *Limited Memory Quasi Newton* methods are preferred since they maintain low memory requirements and have shown superior performance.

2.3.3 Quasi-Newton methods

These methods are also based on the quadratic model, eqn. (2.4), but unlike the conjugate gradient methods, make explicit use of an $n \times n$ matrix $\boldsymbol{B}$. Minimizing the quadratic model is equivalent to solving:

$$\boldsymbol{G}\boldsymbol{h} = -\boldsymbol{g} \tag{2.6}$$

Quasi Newton (QN) methods, maintain a positive definite matrix $\boldsymbol{B} \approx \boldsymbol{G}$ that approximates the Hessian. This matrix is updated iteratively, in an attempt to offer an improved approximation at every iteration. So a slightly different linear system is solved, namely:

$$\boldsymbol{B}\ \ = -\boldsymbol{g} \tag{2.7}$$

The backbone algorithm for the quasi-Newton methods [40, 35, 53] is presented below. At the start of the k^{th} iteration a point $\boldsymbol{x}^{(k)}$, the gradient $\boldsymbol{g}^{(k)}$ and an approximation $\boldsymbol{B}^{(k)}$ to

the Hessian matrix $\boldsymbol{G}^{(k)}$ are available. A common initial choice is $\boldsymbol{B}^{(0)} = I$ (the identity matrix). The following steps are then executed:

(1) Check the termination criteria in order to stop or not.

(2) Solve $\boldsymbol{B}^{(k)}$ $^{(k)} = -\boldsymbol{g}^{(k)}$ for $^{(k)}$.

(3) Apply a selection method to obtain a "better" point $\boldsymbol{x}^{(k+1)}$.

(4) Calculate the gradient vector $\boldsymbol{g}^{(k+1)}$.

(5) Update $\boldsymbol{B}^{(k)}$ to $\boldsymbol{B}^{(k+1)}$ using a quasi-Newton formula.

Steps 3 and 5 above need further elaboration.
In step 3, to obtain a "better" point either a line search is performed along the quasi-Newton direction, or a trust region strategy is followed.
The line search determines a value $\lambda = \lambda^*$ so as to reduce the value of the function $f(\boldsymbol{x}^{(k)} + \lambda$ $^{(k)})$, according to the so called Wolfe–Powell [59, 161, 126] criteria. The new point is then taken to be $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \lambda^*$ $^{(k)}$.
The trust region strategy minimizes with respect to $\boldsymbol{h}^{(k)}$ the quadratic form:

$$q(\boldsymbol{h}^{(k)}) = f(\boldsymbol{x}^{(k)}) + \boldsymbol{g}^{(k)T}\boldsymbol{h}^{(k)} + \frac{1}{2}\boldsymbol{h}^{(k)T}\boldsymbol{B}^{(k)}\boldsymbol{h}^{(k)}$$

subject to: $\|\boldsymbol{h}^{(k)}\| \leq R^{(k)}$, where $R^{(k)}$ is a properly chosen radius (the trust region radius) so that the quadratic approximation is reliable. In this case the candidate point $\boldsymbol{x}^{(k)} + \boldsymbol{h}^{(k)}$ is either accepted, if it corresponds to a lower value, or rejected otherwise. The trust region radius is then updated to $R^{(k+1)}$ in order to make the quadratic approximation more trustworthy at the next iteration.

The updates in step 5 most widely used are the BFGS [39, 56, 20, 140] update and the DFP [31, 41] update. Using the definitions: $\boldsymbol{\delta}^{(k)} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$ and $\boldsymbol{\gamma}^{(k)} = \boldsymbol{g}^{(k+1)} - \boldsymbol{g}^{(k)}$, we can write down the update formulas (the iteration superscript (k) is dropped on the right hand side).
The BFGS update is:

$$\boldsymbol{B}^{(k+1)} = \boldsymbol{B} + \frac{\boldsymbol{\gamma}\boldsymbol{\gamma}^T}{\boldsymbol{\delta}^T\boldsymbol{\gamma}} - \frac{\boldsymbol{B}\boldsymbol{\delta}\boldsymbol{\delta}^T\boldsymbol{B}}{\boldsymbol{\delta}^T\boldsymbol{B}\boldsymbol{\delta}} \tag{2.8}$$

The DFP update is:

$$\boldsymbol{B}^{(k+1)} = \boldsymbol{B} + \left(1 + \frac{\boldsymbol{\delta}^T\boldsymbol{B}\boldsymbol{\delta}}{\boldsymbol{\delta}^T\boldsymbol{\gamma}}\right)\frac{\boldsymbol{\gamma}\boldsymbol{\gamma}^T}{\boldsymbol{\delta}^T\boldsymbol{\gamma}} - \frac{\boldsymbol{\gamma}\boldsymbol{\delta}^T\boldsymbol{B} + \boldsymbol{B}\boldsymbol{\delta}\boldsymbol{\gamma}^T}{\boldsymbol{\delta}^T\boldsymbol{\gamma}} \tag{2.9}$$

If the initial approximation $\boldsymbol{B}^{(0)}$ is positive definite, the above updates maintain this property. However due to roundoff error involved in the numerical procedure after a number of cycles $\boldsymbol{B}$ may become indefinite. To forbid this from happening, the $\boldsymbol{B}$ matrix is factorized in a way that guarantees its positive definiteness even in the presence of roundoff error, and then update equivalently the factors at every iteration. Among others,

the Cholesky $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T$ [35] and the Goldfarb–Idnani [57] $\boldsymbol{Z}^T\boldsymbol{B}\boldsymbol{Z} = \boldsymbol{I}$ factorizations are the ones most frequently employed in the literature.
The update formulas 2.8 and 2.9 share the interesting property of duality. If one sets $\boldsymbol{H} = \boldsymbol{B}^{-1}$ then the equivalent updates to $\boldsymbol{H}$ can be obtained from the update formulas for $\boldsymbol{B}$, using the Sherman–Morrison–Woodbury rule.

$$(\boldsymbol{B} + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{B}^{-1} - \frac{\boldsymbol{B}^{-1}\boldsymbol{u}\boldsymbol{v}^T\boldsymbol{B}^{-1}}{1 + \boldsymbol{v}^T\boldsymbol{B}^{-1}\boldsymbol{u}} \tag{2.10}$$

where $\boldsymbol{B}$ is a non-singular $n \times n$ matrix, $\boldsymbol{u}, \boldsymbol{v} \in R^n$, and $1 + \boldsymbol{v}^T\boldsymbol{B}^{-1}\boldsymbol{u} \neq 0$.
The DFP update is:

$$\boldsymbol{H}^{(k+1)} = \boldsymbol{H} + \frac{\boldsymbol{\delta}\boldsymbol{\delta}^T}{\boldsymbol{\gamma}^T\boldsymbol{\delta}} - \frac{\boldsymbol{H}\boldsymbol{\gamma}\boldsymbol{\gamma}^T\boldsymbol{H}}{\boldsymbol{\gamma}^T\boldsymbol{H}\boldsymbol{\gamma}} \tag{2.11}$$

The BFGS update is:

$$\boldsymbol{H}^{(k+1)} = \boldsymbol{H} + \left(1 + \frac{\boldsymbol{\gamma}^T\boldsymbol{H}\boldsymbol{\gamma}}{\boldsymbol{\gamma}^T\boldsymbol{\delta}}\right)\frac{\boldsymbol{\delta}\boldsymbol{\delta}^T}{\boldsymbol{\gamma}^T\boldsymbol{\delta}} - \frac{\boldsymbol{\delta}\boldsymbol{\gamma}^T\boldsymbol{H} + \boldsymbol{H}\boldsymbol{\gamma}\boldsymbol{\delta}^T}{\boldsymbol{\gamma}^T\boldsymbol{\delta}} \tag{2.12}$$

Note that the BFGS update for $\boldsymbol{H}$ can be obtained from the DFP update for $\boldsymbol{B}$ by interchanging $\boldsymbol{\gamma}$ with $\boldsymbol{\delta}$ and similarly the DFP update for $\boldsymbol{H}$ can be obtained from the BFGS update for $\boldsymbol{B}$, via the same interchange. In that sense the BFGS and the DFP updates are called dual. In practice both the $\boldsymbol{B}$ and the $\boldsymbol{H}$ updates have been used successfully. There is an advantage to use the $\boldsymbol{B}$ update when one wishes to treat some of the problem parameters as constants, since then the obvious dimensional reduction is easily implemented. If factorization techniques are not used, (a bad choice, since the code will not be robust) the $\boldsymbol{H}$ update has a edge, since it only requires a matrix-vector multiplication and not the solution of a linear system of equations.

2.3.4 Line search for descent methods

Line searches are used in quasi-Newton and conjugate gradient methods. The idea of a line search algorithm is simple: given a descent direction $\boldsymbol{s}^{(k)}$, we take a step $\lambda^{(k)}$ in that direction, that yields an acceptable next iterate. For convenience we denote $f(\lambda) \equiv f(\boldsymbol{x}^{(k)} + \lambda^{(k)}\boldsymbol{s}^{(k)})$ and $f'(\lambda) \equiv \boldsymbol{s}^{(k)^T}\boldsymbol{g}^{(k)}(\boldsymbol{x}^{(k)} + \lambda^{(k)}\boldsymbol{s}^{(k)})$. Descent methods are known to converge [59, 161, 126] when λ is chosen to satisfy the *weak Wolfe–Powell* conditions:

$$f(\lambda) \leq f(0) + \lambda\rho f'(0) \tag{2.13}$$

and

$$f'(\lambda) \geq \sigma f'(0) \tag{2.14}$$

where $\rho \in (0, \frac{1}{2})$ and $\sigma \in (\rho, 1)$. In practice, we prefer to use the more stringent test

$$|f'(\lambda)| \leq -\sigma f'(0) \tag{2.15}$$

in place of eqn. (2.14), which along with Eq. (2.13) are called the *strong Wolfe–Powell* conditions.

We describe a line search algorithm which uses a sectioning scheme and that mainly follows Al–Baali and Fletcher [2]. In the sectioning scheme, sequences a_j, b_j, λ_j are generated. a_j is always the current best point (least f) that satisfies Eq. (2.13) but neither Eq. (2.14) nor (Eq. 2.15). λ_j is the current trial point. b_j either fails to satisfy Eq. 2.13, or $f(b_j) \geq f(a_j)$, or both. However the interval (a_j, b_j) will always bracket either an interval of acceptable points, or points for which $f(\lambda) \leq \overline{f}$, with $\overline{f}$ being a lower bound on f.

The line search is initialized with $a_1 = 0$, $b_1 = \infty$, $\overline{f} \leq f(0)$ and an estimation for $\lambda_1 > 0$. The j^{th} iteration is given below:

(1) Evaluate $f(\lambda_j)$

(2) If $f(\lambda_j) \leq \overline{f}$ then terminate

(3) If $f(\lambda_j) > f(0) + \lambda_j \rho f'(0)$ or $f(\lambda_j) \geq f(a_j)$ then

Choose $\lambda_{j+1} \in T(a_j, \lambda_j)$ using either a quadratic interpolating $f(a_j)$, $f'(a_j)$ and $f(\lambda_j)$, or a cubic interpolating $f(a_j)$, $f'(a_)$, $f(\lambda_j)$ and $f(b_j)$.

Set $a_{j+1} = a_j$, $b_{j+1} = \lambda_j$.

else

Evaluate $f'(\lambda_j)$

Test for termination. For the weak Wolfe–Powell conditions use Eq. (2.14), otherwise use Eq. (2.15).

Set $a_{j+1} = \lambda_j$

If $(b_j - a_j) f'(\lambda_j) < 0$ then

Choose $\lambda_{j+1} \in E(a_j, \lambda_j, b_j)$ using a cubic interpolating either $f(a_j)$, $f'(a_j)$, $f(\lambda_j)$ $f'(\lambda_j)$, or $f(a_j)$, $f(b_j)$, $f(\lambda_j)$, $f'(\lambda_j)$.

Set $b_{j+1} = b_j$

else

Choose $\lambda_{j+1} \in T(a_j, \lambda_j)$ using a cubic that interpolates $f(a_j)$, $f'(a_j)$, $f(\lambda_j)$, $f'(\lambda_j)$.

Set $b_{j+1} = a_j$

end if

end if

When interpolating, we use the truncation scheme defined by

$$T(a,b) = \begin{cases} [a + \tau_1(b-a), b - \tau_2(b-a)] & \text{if } a < b \\ [b + \tau_2(a-b), a - \tau_1(a-b)] & \text{if } b < a \end{cases} \tag{2.16}$$

where $0 < \tau_1 \leq \tau_2 \leq \frac{1}{2}$. When extrapolating, we define

$$E(a,\lambda,b) = \begin{cases} [\min(\tau_3,\lambda), \min(\tau_4,\lambda)] & \text{if } a < \lambda < b = \infty \\ [\lambda + \tau_5(b-a), b - \tau_6(b-a)] & \text{if } a < \lambda < b \\ [b + \tau_6(a-b), \lambda - \tau_5(a-b)] & \text{if } b < \lambda < a \end{cases} \tag{2.17}$$

where $1 < \tau_3 \leq \tau_4$ and $0 < \tau_5 \leq \tau_6 \leq \frac{1}{2}$.

2.3.5 Trust Region

The dogleg technique

Given a quadratic model:

$$f(\boldsymbol{x}+\boldsymbol{h}) \approx q(\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^T \nabla f(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{h}^T \nabla^2 f(\boldsymbol{x})\boldsymbol{h} = f(\boldsymbol{x}) + \boldsymbol{h}^T \boldsymbol{g} + \frac{1}{2}\boldsymbol{h}^T \boldsymbol{G}\boldsymbol{h}$$

the problem:

$$\min_{\boldsymbol{h}} \{q(\boldsymbol{h})\} \text{ subject to: } \|\boldsymbol{h}\| \leq R$$

is solved approximately by the following technique termed by Powell as the *dogleg* method [124]. This is one way to approximately solve the above constrained optimization problem inside the trust region defined by its radius R. Two points are calculated. The Cauchy point $\boldsymbol{x}_c = \boldsymbol{x} + \boldsymbol{h}_c$ and the Newton point: $\boldsymbol{x}_N = \boldsymbol{x} + \boldsymbol{h}_N$. The Cauchy point is the minimum along the gradient direction, i.e. $\boldsymbol{h}_c = -\lambda \boldsymbol{g}$ with $\lambda = \frac{\boldsymbol{g}^T\boldsymbol{g}}{\boldsymbol{g}^T\boldsymbol{G}\boldsymbol{g}}$, while the Newton step is given by: $\boldsymbol{h}_N = -\boldsymbol{G}^{-1}\boldsymbol{g}$. If $\|\boldsymbol{h}_N\| \leq R$ the Newton point is taken as the next trial iterate. Otherwise the first point where the piecewise linear trajectory: $\boldsymbol{x} \to \boldsymbol{x}_c \to \boldsymbol{x}_N$ intersects the sphere of radius R centered at $\boldsymbol{x}$, is taken as the next trial iterate.

Dennis & Mei [34] proposed a similar procedure termed *double dogleg*, that defines another point $\boldsymbol{x}_D = \boldsymbol{x} + \boldsymbol{h}_D$ with:

$$\boldsymbol{h}_D = \zeta \boldsymbol{h}_N, \quad \zeta = 0.8\gamma + 0.2, \quad \text{and} \quad \gamma = \frac{(\boldsymbol{g}^T\boldsymbol{g})^2}{(\boldsymbol{g}^T\boldsymbol{G}\boldsymbol{g})(\boldsymbol{g}^T\boldsymbol{G}^{-1}\boldsymbol{g})}$$

and a modified trajectory: $\boldsymbol{x} \to \boldsymbol{x}_c \to \boldsymbol{x}_D \to \boldsymbol{x}_N$.

The updating scheme of the trust region radius is given below:

(1) Calculate the ratio of the actual to the expected reduction: $r^{(k)} = \frac{f^{(k)} - f^{(k+1)}}{f^{(k)} - q(\boldsymbol{h}^{(k)})}$, where $f^{(k)}$ stands for $f(\boldsymbol{x}^{(k)})$ and $f^{(k+1)} = f(\boldsymbol{x}^{(k)} + \boldsymbol{h}^{(k)})$.

(2) Accept or reject the trial point according to:
If $r^{(k)} \leq 0$ then
$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}$, $f^{(k+1)} = f^{(k)}$
else
$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{h}^{(k)}$
end if

(3) if $r^{(k)} < 0.25$ then
$R^{(k+1)} = \|\boldsymbol{h}^{(k)}\|/4$
else if $r^{(k)} > 0.75$ and $\|\boldsymbol{h}^{(k)}\| = R^{(k)}$ then
$R^{(k+1)} = 2R^{(k)}$
else
$R^{(k+1)} = R^{(k)}$
end if

2.3.6 Sum of squares problems

Levenberg–Marquardt method for sum of squares

For the case where the objective function is a sum of squares, i.e.

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} f_i^2(\boldsymbol{x}) \equiv \boldsymbol{r}(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x})$$

with $\boldsymbol{r}^T(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x}))^T$, a special method has been proposed first by Levenberg [87] and later on by Marquardt [100]. Let $\boldsymbol{J}$ be the Jacobian matrix $\frac{\partial f_i(\boldsymbol{x})}{\partial x_j}$ and let $\boldsymbol{D}$ be a diagonal matrix. The quadratic approximation to $f(\boldsymbol{x}+\boldsymbol{h})$ is given by:

$$q(\boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{x})\boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T\boldsymbol{B}(\boldsymbol{x})\boldsymbol{h}$$

and is being minimized under the condition $\|\boldsymbol{D}\boldsymbol{h}\| \leq R$, where R is the radius of the trust region. The trust region in this case is a hyper–ellipsoid with semi–axis lengths $R/\boldsymbol{D}_{ii}$. Since $\boldsymbol{g} = 2\boldsymbol{J}^T\boldsymbol{r}$ and if the Gauss–Newton approximation is made i.e. $\boldsymbol{B}(\boldsymbol{x}) \approx 2\boldsymbol{J}^T\boldsymbol{J}$, we get using the Lagrange multiplier procedure:

$$[\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{D}^T\boldsymbol{D}]\boldsymbol{h} = -\boldsymbol{J}^T\boldsymbol{r} \quad \text{or} \quad \boldsymbol{h}(\lambda) = -[\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{D}^T\boldsymbol{D}]^{-1}\boldsymbol{J}^T\boldsymbol{r}$$

Initially we set $\boldsymbol{D}_{ii}^{(0)} = \|\frac{\partial \boldsymbol{r}(\boldsymbol{x}^{(0)})}{\partial x_i}\| \;\; \forall\; i = 1, 2, \ldots, N$. The k^{th} iteration of the algorithm is as:

(1) If $\|\boldsymbol{D}^{(k)}\boldsymbol{h}^{(k)}(0)\| \leq R^{(k)}$ then
Set $\boldsymbol{\delta}^{(k)} = \boldsymbol{h}^{(k)}(0)$
else
find a $\lambda^{(k)} > 0$ such that $\|\boldsymbol{D}^{(k)}\boldsymbol{h}^{(k)}(\lambda^{(k)})\| = R^{(k)}$
Set $\boldsymbol{\delta}^{(k)} = \boldsymbol{h}^{(k)}(\lambda^{(k)})$
end if
If $f(\boldsymbol{x}^{(k)} + \boldsymbol{\delta}^{(k)}) < f(\boldsymbol{x}^{(k)})$ then
Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{\delta}^{(k)}$, and calculate $\boldsymbol{J}^{(k+1)}$
else
Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}$ and $\boldsymbol{J}^{(k+1)} = \boldsymbol{J}^{(k)}$
end if

(2) Update $R^{(k)}$ to $R^{(k+1)}$ (the algorithm is similar to the one described for the dogleg case).

(3) Choose $\boldsymbol{D}^{(k+1)}$ as:
$\boldsymbol{D}_{ii}^{(k+1)} = \max\left\{\boldsymbol{D}_{ii}^{(k)}, \|\frac{\partial \boldsymbol{r}(\boldsymbol{x}^{(k+1)})}{\partial x_i}\|\right\} \quad \forall\, i = 1, 2, \ldots, N$

A very robust implementation of the above is described by Moré [105].

2.4 Methods that use second derivatives

2.4.1 Newton's Method

The idea behind Newton's method is the basis of many algorithms. Again the quadratic model 2.4 is employed and under the assumption that the initial point is near the minimum, i.e. $|\boldsymbol{h}| = |\boldsymbol{x}^* - \boldsymbol{x}^{(0)}|$ is small, we may determine $\boldsymbol{h}$ by solving the linear system $\boldsymbol{G}\boldsymbol{h} = -\boldsymbol{g}$. The algorithm is similar to the one described for the Quasi-Newton methods, replacing however the updates to the $\boldsymbol{B}$ matrix with the calculation of the Hessian matrix $\boldsymbol{G}$. While $\boldsymbol{G}$ is positive definite the above procedure is stable, however $\boldsymbol{G}$ may become singular or indefinite, in which case the above algorithm has to be modified to be operational. Many alternatives have been suggested in the literature under the name *Modified Newton* methods, [60],[58]. However the method by Gill and Murray [52], seems to perform better. Their method uses an $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$ Cholesky factorization for the Hessian, where $\boldsymbol{L}$ is a lower triangular matrix with all diagonal elements equal to one. This is possible when $\boldsymbol{G}$ is positive definite. When $\boldsymbol{G}$ is indefinite or singular, then a diagonal matrix is added to $\boldsymbol{G}$ suitably to ensure positive definiteness, and this modified $\boldsymbol{G}$ is factorized. The resulting linear system is then solved taking advantage of the existing factors. This method attains quadratic convergence near the minimum, a very desirable feature. A disadvantage of the method is the usually expensive calculation of the Hessian and its factorization at every iteration.

2.5 Termination Criteria

The issue when to terminate an algorithm, and therefore regard the current point as a minimizer, is important. Ideally the necessary conditions alone (**??**), could dictate the termination. However in practice, this will not work most of the time. For one thing, the gradient will very rarely if ever, become exactly equal to zero, due to the rounding errors that prevent exact arithmetic. In addition the optimality conditions (**??**), assume that the objective function is twice continuously differentiable, which actually may not be the case. Hence practical termination criteria had to be developed that deviate from the ideally expected. An obvious modification that has been frequently employed is:

$$|\nabla f(x)| \leq \epsilon$$

where ϵ is a small positive number. This rule is not scale invariant and hence is not recommended for general use. Some other rules, that can be used in combination are listed.

$$\begin{aligned} |x^{(k)} - x^{(k-1)}||\nabla f(x^{(k)})| &\leq \epsilon \quad \textit{(scale invariant)} \\ |x^{(k)} - x^{(k-1)}| &\leq \epsilon\ (1 + |x^{(k)}|\ ) \\ f(x^{(k-1)}) - f(x^{(k)}) &\leq \epsilon\ max(\ |f(x^{(k)})|,\ |f(x^{(k-1)})|\ ) \end{aligned}$$

For the Simplex method, an appropriate termination criterion is based upon:

$$\frac{1}{n+1}\sum_{i=0}^{n} |f_i - \bar{f}| \leq \epsilon$$

$$\text{where,} \quad \bar{f} = \frac{1}{n+1}\sum_{i=0}^{n} f_i$$

The tolerances (all denoted above by ϵ), should be chosen with care and taking in account the accuracy of the calculations. Additional checks may be performed to establish the quality of the approximate minimizer and possibly improve it, after the minimization algorithm has come to an end.

Chapter 3

An algorithm for convex quadratic programming subject to bound constraints

3.1 Summary

We present an algorithm for solving a quadratic programming problem with positive definite Hessian and bound constraints, that employs a Lagrange multiplier approach. The proposed method falls in the category of active set techniques. The algorithm, at each iteration, modifies the minimization parameters both in the primal space and in the dual space (Lagrange multipliers). The method may be profitably used on a number of problems from the fields of Physics, Chemistry, Computer Science and Engineering. Comparative results of numerical experiments are reported demonstrating the advantages of the proposed approach.

3.2 Introduction

The problem of minimizing a convex quadratic function subject to bound constraints appears quite frequently in applications. For instance, many problems in computational physics and engineering, are reduced to quadratic programming problems. Portfolio management can also be formulated as quadratic programming problem [120]. In the field of Artificial Intelligence, and especially in Support Vector Machines (SVM) an efficient quadratic solver is crucial for the training process [64, 113]. Also methods for calculating the radiation intensity in oncology treatment are formulated as quadratic optimization problems [18]. In Physics, Chemistry and Engineering, the resulting optimization problems are most often highly non-linear. Iterative optimization methods, model the objective function by truncating its Taylor expansion up to and including the quadratic term. Solving efficiently the recurring quadratic problems is crucial for the overall performance of

the optimization procedure. In Section 2.3.5 we present the role of the proposed method in the general setting of Sequential Quadratic Programming methods for non-linear programming. To be more specific Quasi-Newton (BFGS, DFP, SR1) or modified Newton methods that are among the most important non-linear optimizers, are implemented in the framework of trust region methods that require the solution of a quadratic problem at each iteration. The Quadratic Programming problem with simple bounds is stated as:

$$\begin{aligned} q(x) = \min_{x} \ & \frac{1}{2} x^T B x + x^T d, \\ \text{subject to: } & a_i \leq x_i \leq b_i, \forall i \in I = \{1, 2, \cdots, n\} \end{aligned} \tag{3.1}$$

where $x, d \in R^n$ and B is a symmetric, positive definite $n \times n$ matrix.

For the problem in Eq. (3.1) two major strategies exist in the literature, both of which require feasible steps to be taken.

The first one is the Active Set strategy [53, 40, 8] which generates iterates on a face of the feasible box until either a minimizer of the objective function is found or a point on the boundary of that face is reached. The basic disadvantage of this approach, especially in the large-scale case, is that constraints are added or removed one at a time, thus requiring a number of iterations proportional to the problem size. To overcome this, gradient projection methods [28, 8] were proposed. In that framework the active set algorithm is allowed to add or remove many constraints per iteration.

The second strategy [164, 66, 26] consists in treating the inequality constraints using interior point algorithms. In brief, an interior point algorithm consists of adding a series of parameterized barrier functions which are minimized using Newton's method. The major computational cost is due to the solution of a linear system, which provides a feasible search direction. Recently, D' Apuzzo et. al. [29] presented a parallel implementation of an interior-point method for box-constrained quadratic programming.

In the present article we propose an infeasible active set algorithm, which generates a finite number of iterations that are not necessarily descent. In each step we maintain the first order optimality condition along with the complementarity constraint, until primal and dual feasibility hold. Two closely related methods in the literature are the Projected Newton method [7] and the infeasible method of Kunisch and Rendl [80] that treats only upper bounds. In the first case the first order optimality condition is satisfied, primal feasibility is maintained throughout the iterations and a line search scheme is applied to guarantee convergence. In the second case the new iterate is uniquely determined by the active set. Hence note that the problem may be solved in at most 2^n iterations[1]. We recognize that bound constraints are a very special case of linear inequalities, which may in general have the form $Ax \geq b$, A being an $m \times n$ matrix and b is a vector $\in R^m$. Our investigation is also motivated by the fact that in the convex case, and under certain conditions a problem subject to inequality constraints can be transformed to a bound

[1] 2^n is the number of all possible active sets

constrained one, using duality, i.e.:

$$\begin{aligned} \min_{x \in R^n} \ & \frac{1}{2} x^T B x + x^T d \\ \text{subject to: } & Ax \geq b \end{aligned} \tag{3.2}$$

is equivalent to the dual:

$$\begin{aligned} \max_{y \in R^m} \ & -\frac{1}{2} y^T \tilde{B} y + y^T \tilde{d} \\ \text{subject to: } & y \geq 0 \end{aligned} \tag{3.3}$$

where $\tilde{B} = AB^{-1}A^T$ a positive definite matrix and $\tilde{d} = AB^{-1}d + b$. The dual problem in Eq. (3.3) is also a quadratic problem subject to bounds. Let y^* be the solution of the dual problem. We can then obtain, under certain circumstances, the solution to the initial problem of Eq. (3.2) as:

$$x^* = B^{-1}(A^T y^* - d) \tag{3.4}$$

The paper is organized as follows. The proposed algorithm is described in Section 3.3. In Section 3.4 we briefly present three different competing quadratic programming codes and comparison on five different test problem types is performed in Section 3.5. Finally, in Section 2.3.5 we present a trust region approach for nonlinear bound constrained problems that takes full advantage of the present quadratic programming method.

3.3 Solving the quadratic problem

For the problem in Eq. (3.1), we construct the associated Lagrangian:

$$L(x, \lambda, \mu) = \frac{1}{2} x^T B x + x^T d - \lambda^T (x - a) - \mu^T (b - x) \tag{3.5}$$

The KKT necessary conditions at the minimum $x^*, \lambda^*, \mu^* \in \mathbf{R}^n$ require that:

$$\begin{aligned} Bx^* + d - \lambda^* + \mu^* &= 0 \\ \lambda_i^* \geq 0, \quad \mu_i^* &\geq 0, \ \forall i \in I \\ \lambda_i^* (x_i^* - a_i) &= 0, \ \forall i \in I \\ \mu_i^* (b_i - x_i^*) &= 0, \ \forall i \in I \\ x_i^* &\in [a_i, b_i], \ \forall i \in I \end{aligned} \tag{3.6}$$

A solution to the above system (3.6), can be obtained through an active set strategy described in detail in Algorithm 1:

Algorithm 1 BOXCQP

Initially set: $k = 0$, $\lambda^{(0)} = \mu^{(0)} = 0$ and $x^{(0)} = -B^{-1}d$.
If $x^{(0)}$ is feasible, **Stop**, the solution is: $x^* = x^{(0)}$.
At iteration k, the quantities $x^{(k)}, \lambda^{(k)}, \mu^{(k)}$ are available.

1. Define the sets:

$$\begin{aligned} L^{(k)} &= \{i : x_i^{(k)} < a_i, \text{ or } x_i^{(k)} = a_i \text{ and } \lambda_i^{(k)} \geq 0\} \\ U^{(k)} &= \{i : x_i^{(k)} > b_i, \text{ or } x_i^{(k)} = b_i \text{ and } \mu_i^{(k)} \geq 0\} \\ S^{(k)} &= \{i : a_i < x_i^{(k)} < b_i, \text{ or } x_i^{(k)} = a_i \text{ and } \lambda_i^{(k)} < 0, \\ &\text{or} \quad x_i^{(k)} = b_i \text{ and } \mu_i^{(k)} < 0\} \end{aligned}$$

   Note that $L^{(k)} \cup U^{(k)} \cup S^{(k)} = I$

2. Set:

$$\begin{aligned} x_i^{(k+1)} &= a_i, \ \mu_i^{(k+1)} = 0, \quad \forall i \in L^{(k)} \\ x_i^{(k+1)} &= b_i, \ \lambda_i^{(k+1)} = 0, \quad \forall i \in U^{(k)} \\ \lambda_i^{(k+1)} &= 0, \ \mu_i^{(k+1)} = 0, \quad \forall i \in S^{(k)} \end{aligned}$$

3. Solve:

$$Bx^{(k+1)} + d = \lambda^{(k+1)} - \mu^{(k+1)}$$

   for the n unknowns:

$$\begin{aligned} &x_i^{(k+1)}, \ \forall i \ \in S^{(k)} \\ &\mu_i^{(k+1)}, \ \forall i \ \in U^{(k)} \\ &\lambda_i^{(k+1)}, \ \forall i \ \in L^{(k)} \end{aligned}$$

4. Check if the new point is a solution and decide to either stop or iterate.

   If $(x_i^{(k+1)} \in [a_i, b_i] \ \forall i \in S^{(k)}$ **and** $\mu_i^{(k+1)} \geq 0, \ \forall i \in U^{(k)}$
   and $\lambda_i^{(k+1)} \geq 0, \ \forall i \in L^{(k)})$ **Then**
   Stop, the solution is: $x^* = x^{(k+1)}$.
   Else
   set $k \leftarrow k + 1$ and iterate from **Step 1.**
   Endif

The solution of the linear system in Step 3 of Algorithm 3.3, needs further consideration. Let us rewrite the system in a componentwise fashion.

$$\sum_{j \in I} B_{ij} x_j^{(k+1)} + d_i = \lambda_i^{(k+1)} - \mu_i^{(k+1)},\ \forall i \in I \tag{3.7}$$

Since $\forall i \in S^{(k)}$ we have that $\lambda_i^{(k+1)} = \mu_i^{(k+1)} = 0$, hence we can calculate $x_i^{(k+1)},\ \forall i \in S^{(k)}$ by splitting the sum in Eq. (3.7) and taking into account Step 2 of the algorithm, i.e.:

$$\sum_{j \in S^{(k)}} B_{ij} x_j^{(k+1)} = -\sum_{j \in L^{(k)}} B_{ij} a_j - \sum_{j \in U^{(k)}} B_{ij} b_j - d_i,\ \forall i \in S^{(k)} \tag{3.8}$$

The submatrix $B_{ij},\ \text{with}\ i, j \in S^{(k)}$ is positive definite as can be readily verified, given that the full matrix B is. The calculation of $\lambda_i^{(k+1)},\ \forall i \in L^{(k)}$ and of $\mu_i^{(k+1)},\ \forall i \in U^{(k)}$ is straightforward and is given by:

$$\lambda_i^{(k+1)} = \sum_{j \in I} B_{ij} x_j^{(k+1)} + d_i,\ \forall i \in L^{(k)} \tag{3.9}$$

$$\mu_i^{(k+1)} = -\sum_{j \in I} B_{ij} x_j^{(k+1)} - d_i,\ \forall i \in U^{(k)} \tag{3.10}$$

Convergence analysis in the line of Kunisch and Rendl [80] may be followed also for our method. We numerically tested cases with thousands of variables and a wide spectra for the condition number of B ranging from 1.259 to 10^5. When B becomes nearly singular, then oscillation occurs as expected. (Note that for such cases the linear system $Bx = -d$ is ill conditioned). At this point corrective measures may be taken.

The main computational task of the algorithm above, is the solution of the linear system in Step 3. The size of the system may vary according to the size of the active set in each iteration. In our implementation we solve the linear system using either using a direct solver via Cholesky decomposition (Variant 1) or the conjugate gradient method (without any preconditioning) (Variant 2). We also provide the option to initially use an inaccurate-inexpensive conjugate gradient search to obtain a starting point, and then to switch back to Cholesky decomposition (Variant 3).

3.4 Other convex quadratic codes

There exist several Quadratic Programming codes in the literature. We have chosen to compare with three of them, specifically with QPBOX, QLD and QUACAN. These codes share several common features so that the comparison is both meaningful and fair. All codes are written in the same language (FORTRAN 77) so that different language overheads are eliminated. Also they are written by leading experts int he field of Quadratic Programming, so that their quality is guaranteed. Notice also that all codes are specific to the problem, and not of general purpose nature and are distributed freely through the World Wide Web, at the moment of this writing.

3.4.1 QPBOX

QPBOX [97] is a Fortran77 package for box constrained quadratic programs developed at IMM of the Technical University of Denmark. The bound constrained quadratic program is solved via a dual problem, which is the minimization of an unbounded, piecewise quadratic function. The dual problem involves a lower bound of λ_1, i.e the smallest eigenvalue of a symmetric, positive definite matrix, and is solved by Newton iteration with line search. (Downloadable from `http://www2.imm.dtu.dk/~hbn/Software/`).

3.4.2 QLD

This program [136] is due to K.Schittkowski of the University of Bayreuth, Germany and is a modification of code due to MJD Powell of the University of Cambridge. It is essentially an active set, interior point method and supports general linear constraints too.(Downloadable from `http://siconos.gforge.inria.fr/Documentation/Numerics\_Doxygen/ql0001\_8f-source.html`).

3.4.3 QUACAN

This program combines conjugate gradients and gradient projection techniques, as in the algorithm of Moré J.J. and Toraldo G. (1991). QUACAN [49] is specialized for convex problems subject to simple bounds. (Downloadable from `http://search.cpan.org/src/ELLIPSE/PDL-Opt-NonLinear-0.02/opti\_lib/box9903.f`).

3.5 Results of Numerical Experiments

To verify the effectiveness of the proposed approach we experimented with five different problem types, and measured cpu times to make a comparison possible. We have implemented BOXCQP in Fortan 77 and used a 64-bit AMD Opteron processor with Linux operating system and the GNU `g77` FORTRAN compiler.

In the subsections that follow we describe in detail the test problems used and we report the results of our experiments. In every experiment we have applied all the three variants of BOXCQP (see Section 3.3).

3.5.1 Random problems

The first set of experiments treats randomly generated problems. We generate problems following the general guidelines of [106]. The Hessian matrices B have the form

$$B = M^T M \text{ with } M = D^{\frac{1}{2}} Z \tag{3.11}$$

where

$$D = diag(d_1, \ldots, d_n) \text{ with } d_i = 10^{\frac{i-1}{n-1} ncond} \tag{3.12}$$

where $ncond$ is a positive real, controlling the condition number of B ($\kappa_2(B) = 10^{ncond}$) and Z is a Householder matrix,

$$Z = I - \frac{2}{z^T z} z z^T \tag{3.13}$$

The vectors d, a and b are created by the following procedure, which is controlled by two real numbers in $[0, 1]$, namely act_prob and up_low_prob:

Random problem creation

---

for $i = 0$ to n **do**
  $a_i \leftarrow \text{rand}(-1, 0)$
  $b_i \leftarrow \text{rand}(0, 1)$
for $i = 0$ to n **do**
  Get random $\xi_i \in [0, 1]$
  if $\xi_i \leq act\_prob$ **then**
    Get random $\tilde{\xi}_i \in [0, 1]$ {Add i to the active set.}
    if $\tilde{\xi}_i \leq up\_low\_prob$ **then**
      $x_i \leftarrow b_i$ {On upper bound.}
      $\mu_i \leftarrow \text{rand}(0, 1)$
      $\lambda_i \leftarrow 0$
    else
      $x_i \leftarrow a_i$ {On lower bound.}
      $\mu_i \leftarrow 0$
      $\lambda_i \leftarrow \text{rand}(0, 1)$
  else
    $x_i \leftarrow (a_i + b_i)/2$ {i in the interior.}
    $\mu_i \leftarrow 0$
    $\lambda_i \leftarrow 0$
Calculate $d \leftarrow -Bx + \lambda - \mu$ {From Eq. (3.6).}

We have created three classes of random problems:

(a) Problems that the solution has approximately 50% of the variables on the bounds with equal probability to be either on the lower or on the upper bound ($act\_prob = \frac{1}{2}$, $up\_low\_prob = 0.5$).

(b) Problems that the solution has approximately 90% of the variables on the bounds with equal probability to be on either the lower or on the upper bound ($act\_prob = \frac{9}{10}$, $up\_low\_prob = 0.5$).

(c) Problems that the solution has approximately 10% of the variables on the bounds with equal probability to be either on the lower or on the upper bound ($act\_prob = \frac{1}{10}$, $up\_low\_prob = 0.5$).

For every random problem class we have created Hessian matrices with three different condition numbers:

- Using $ncond = 0.1$ and hence, $\kappa_2(B) = 1.259$

- Using $ncond = 1$ and hence, $\kappa_2(B) = 10$

- Using $ncond = 5$ and hence, $\kappa_2(B) = 10^5$

The results for the three variants of BOXCQP against the other other quadratic codes for the classes (a), (b) and (c) and for three different condition numbers are shown in Tables 3.1, 3.2 and 3.3 respectively.

From the tables, we observe that our method performed worse only on the ill conditioned problems of class (c) (20 cases in Table 3.3 were $\kappa_2(B) = 10^5$). In all other cases, there exists a BOXCQP variant that outperforms all the tested codes. As a general observation we notice that Variant 2, performs better in the majority of the well conditioned problems, whereas Variants 1 and 3 that incorporate the Choleksy decomposition exhibit higher efficiency in the nearly ill conditioned problems. We also note that in the ill conditioned case where $act\_prob = 0.1$ (approximately 10% of the variables of the solution are on the bounds) QPBOX and QLD outperform all three variants of BOXCQP. Commend on Fig 3.5 3.6 3.6 Variant 2 scales very well with dimension for well conditioned problems $ncond = 0.1, ncond = 1$ followed by Variant 3. Variant 1 on the other hand performs better in the ill conditioned case. Seems that Variant 3 is a good compromise when you don't know in advanced the condition number.

As a rule of thumb we propose the usage of Variant 2 when the matrix B is well conditioned and Variant 1 when B is ill conditioned. In cases where we cannot afford to calculate the condition number Variant 3 may be appropriate.

3.5.2 Circus Tent problem

The circus tent problem is taken from Matlab's optimization demo as an example of large-scale quadratic programming with simple bounds. The problem is to build a *circus tent* to cover a square lot. The tent is elastic and is to be supported by five poles. The question is to find the shape of the tent at equilibrium, that corresponds to the minimum of the energy function. As we can see in Figure 3.1, the problem has only lower bounds imposed by the five poles and the ground. The surface formed by the elastic tent, is determined by solving the bound constrained optimization problem:

$$\begin{aligned} &\min_x q(x) = \frac{1}{2}x^T Bx + x^T d \\ &\text{subject to: } a_i \leq x_i \quad \forall i \in I = \{1, 2, \cdots, n\} \end{aligned} \tag{3.14}$$

where $q(x)$ corresponds to the energy function and H is a 5-point finite difference Laplacian over a square grid. It is obvious from Table 3.4 that variant 2 outperforms all other codes. Notice that matrix B exhibits large sparse patterns that favors CGR iterations .

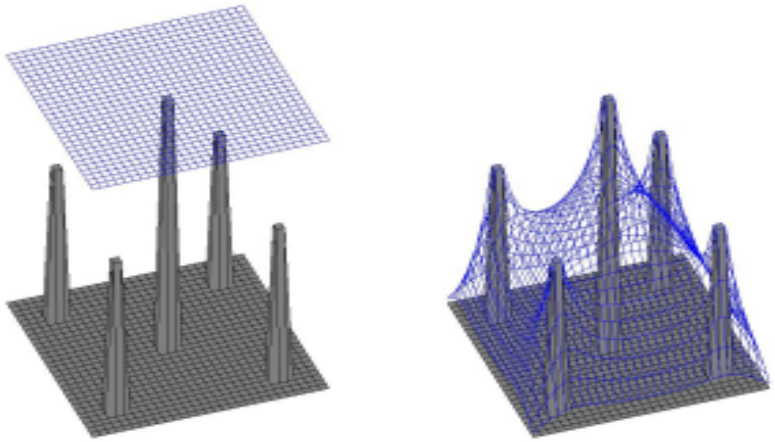


Figure 3.1: Circus tent problem.

3.5.3 Biharmonic Equation problem

We consider the problem of describing small vertical deformations of an horizontal, elastic membrane clamped on a rectangular boundary, under the influence of a vertical force. The membrane is constrained to remain below an obstacle. For an in depth discussion of this problem see [80]. The formulation of the problem is given by Eq.(3.15).

$$\begin{aligned} &\min_x \frac{1}{2}x^T Bx + x^T d \\ &\text{subject to: } x_i \leq b \quad \forall i \in I = \{1, 2, \cdots, n\} \end{aligned} \tag{3.15}$$

We see an example in Fig 3.2 of a membrane under the influence of a vertical force.

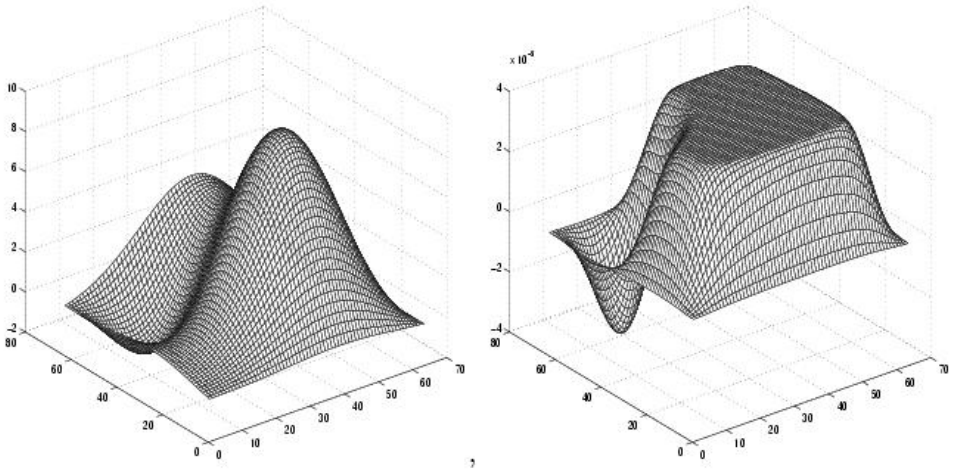


Figure 3.2: On the left we show the acting force, on the right is the final shape of the membrane.

It is obvious from Table 3.4 that variant 3 performs better than all the other codes.

3.5.4 Intensity Modulated Radiation Therapy

This problem arises in the field of radiotherapy and concerns the determination of the spatial distribution of the radiation, in a way that the patient's vital organs are minimally irradiated. Knowing the beam settings and the intensity profile, one can calculate the radiation dose. Inversely, when a desired dose is required, the proper intensity profile for given beam settings can be retrieved by solving a quadratic problem. The beam settings are successively modified in an effort to satisfy a set of clinical constraints, and hence the quadratic subproblem (shown in Eq. (3.16)), must be solved a large number of times [18].

$$\begin{aligned} &\min_x q(x) = \frac{1}{2} x^T B x + x^T d \\ &\text{subject to } x_i \geq 0 \quad \forall i \in I = \{1, 2, \cdots, n\} \end{aligned} \tag{3.16}$$

The results reported in Table 3.4, correspond to real world data, kindly provided by S. Breedveld (private communication). In this example, seven beams are combined resulting to a quadratic problem with 2342 parameters(see last line of Table 3.4).

3.5.5 Support Vector Classification

In this classification problem, the goal is to separate two classes using a hyperplane $h(y) = w^T y + \beta$, which is determined from available examples ($D = \{(y^1, t^1), (y^2, t^2), \ldots (y^l, t^l)\}$, $y \in R^n$, $t \in -1, 1$). Furthermore it is desirable to produce a classifier that will work well on unseen examples, i.e. it will generalize well. Consider the example in Fig. 3.3. There are many possible linear classifiers that can separate the data, but there is only one that maximizes the distance to the nearest data point of each class. This classifier is termed the optimal separating hyperplane and intuitively, one would expect that generalizes optimally.

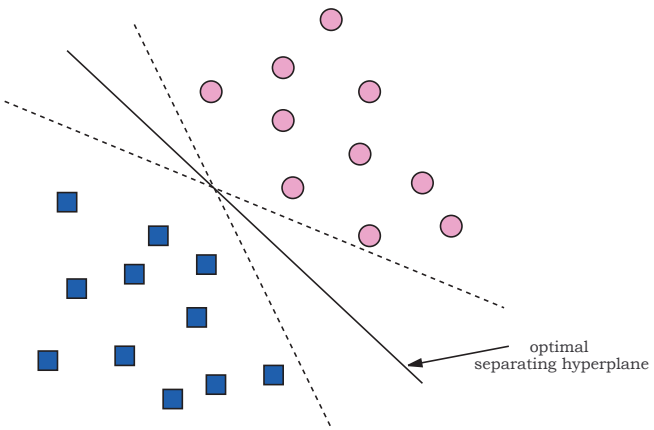


Figure 3.3: Optimal separating classifier.

The formulation of the maximum distance linear classifier (if we omit the constant

term β of the hyperplane equation[2]) is a convex quadratic problem with simple bounds on the variables. The resulting problem has the form:

$$\begin{aligned} &\min_x \frac{1}{2} x^T B x - x^T e \\ &\text{subject to: } 0 \leq x_i \leq C \quad \forall i \in I = \{1, 2, \cdots, n\} \end{aligned} \tag{3.17}$$

where $e \in R^l$ and with $e_i = 1$, $B_{ij} = t^i t^j K(y^i, y^j)$ and $K(y_1, y_2)$ is the kernel function performing the non-linear mapping into the feature space. The parameters $x \in R^l$ are Lagrange multipliers of an original quadratic problem, that define the separating hyperplane using the relation:

$$w^{*T} y = \sum_{i=1}^{l} x_i^* t^i K(y^i, y) \tag{3.18}$$

Hence the separating surface is given by:

$$h(y) = \text{sgn}(w^{*T} y) \tag{3.19}$$

In our experiments we used the CLOUDS [107] data set, which is a two-dimensional data set with two classes. We have constructed the problem in Eq. (3.18) using an RBF Kernel function $K(y_1, y_2) = exp(\frac{-||y_1 - y_2||^2}{2^2})$, and setting $C = 100$. The experiments conducted follow the procedure:

- Form the training set by extracting l examples from the dataset and let the rest examples (5000-l) form the test set.
- Construct the matrix B for the problem in Eq. (3.18)
- Apply each solver, obtain the corresponding separating surface and test-set error.

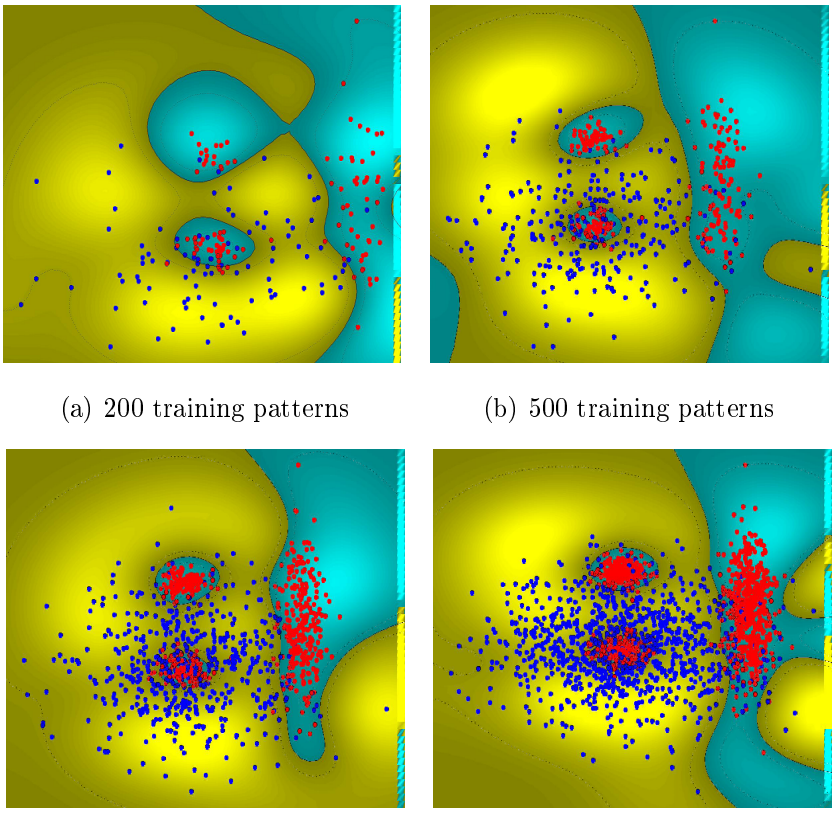
In these experiments the large condition number of matrix B leads to ill conditioned problems. To circumvent this, we added in the main diagonal of B a small positive term of order 10^{-2}. The resulting classification surfaces for $l = 200$, 500, 1000 and 2000 training examples from CLOUDS dataset are shown in Fig. 3.4.

The first 12 lines of Table 3.4 contain results for a varying number of training patterns. The addition of the $O(10^{-2})$ term in the main diagonal led to the creation of well conditioned matrices that could be efficiently solved by CGR algorithm.

3.6 Conclusions

The BOXCQP algorithm specialized to solve box-constrained convex quadratic problems, has been developed. We have presented a number of applications in Computer Science, Physics and Engineering where BOXCQP has been applied. In addition a trust region method for nonlinear problems is sketched, that takes advantage BOXCQP in order to

[2]Also known as explicit bias.



(a) 200 training patterns (b) 500 training patterns

(c) 1000 training patterns (d) 2000 training patterns

Figure 3.4: Examples of SVM classification.

efficiently solve unconstrained and bound constrained non-linear problems. From the experiments one observes the robustness of our method even in the case of nearly ill conditioned problems.

Notice that all the energy minimization problems presented in this work, exhibit large sparsity patterns. Sparsity can be exploited by using special linear solvers and further significant speed-up is expected.

Our software can be downloaded from `http://www.cs.uoi.gr/~voglis` both in a FORTRAN 77 and in a Matlab version.

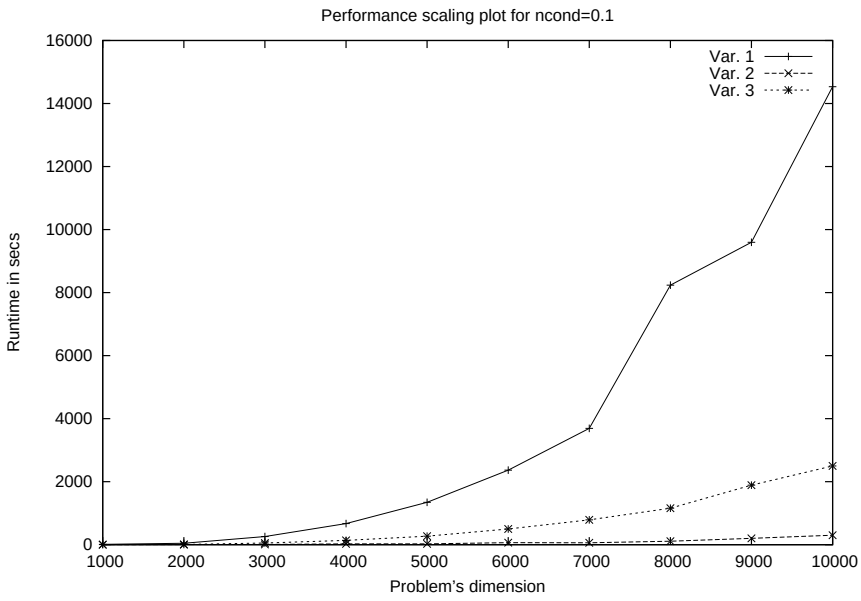


Figure 3.5: Plot for $act\_prob = 0.5$ and $ncond = 0.1$

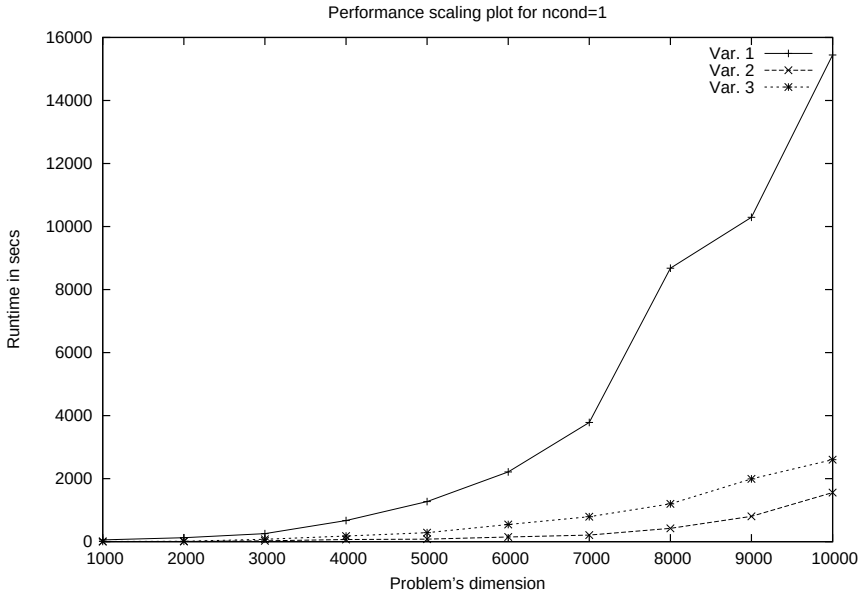


Figure 3.6: Plot for $act\_prob = 0.5$ and $ncond = 1$

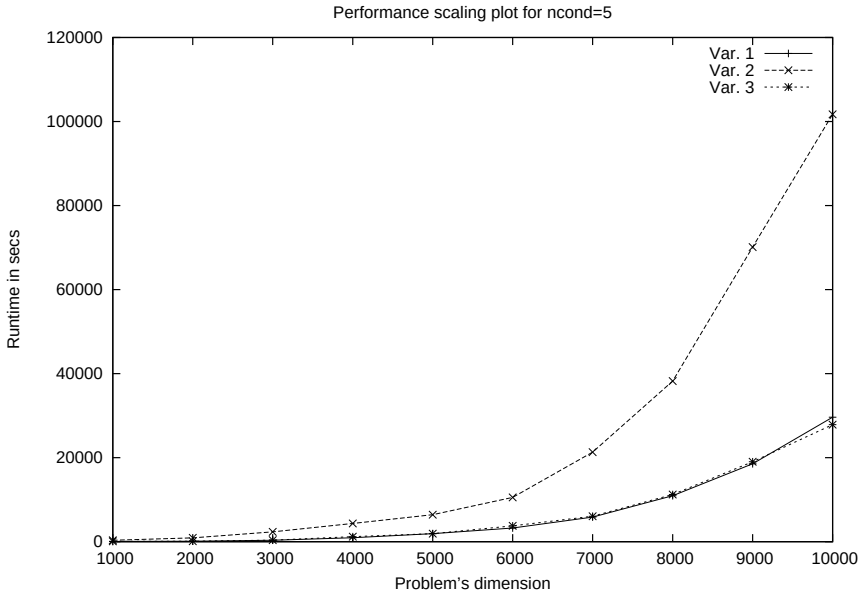


Figure 3.7: Plot for $act\_prob = 0.5$ and $ncond = 5$

Table 3.1: Random table results, $act\_prob = 0.5, up\_low\_prob = 0.5$

Prob. Name	Var.1	Var.2	Var.3	QUACAN	QPBOX	QLD
Random ($ncond = 0.1, n = 100, f^* = -103.70$)	0.00	0.00	0.00	0.00	0.00	0.010
Random ($ncond = 1, n = 100, f^* = -248.90$)	0.00	0.00	0.00	0.00	0.00	0.01
Random ($ncond = 5, n = 100, f^* = -259614$)	0.00	0.02	0.00	0.16	0.01	0.01
Random ($ncond = 0.1, n = 200, f^* = -183.32$)	0.02	0.00	0.003	0.00	0.06	0.08
Random ($ncond = 1, n = 200, f^* = -421.78$)	0.02	0.00	0.01	0.03	0.05	0.07
Random ($ncond = 5, n = 200, f^* = -608139$)	0.03	0.18	0.03	1.38	0.07	0.07
Random ($ncond = 0.1, n = 300, f^* = -394.03$)	0.08	0.01	0.01	0.02	0.19	0.25
Random ($ncond = 1, n = 300, f^* = -707.29$)	0.08	0.02	0.02	0.08	0.20	0.26
Random ($ncond = 5, n = 300, f^* = -1146677$)	0.12	0.75	0.12	5.328	0.25	0.25
Random ($ncond = 0.1, n = 400, f^* = -502.26$)	0.19	0.02	0.050	0.06	0.44	0.60
Random ($ncond = 1, n = 400, f^* = -607.83$)	0.22	0.08	0.11	0.25	0.47	0.61
Random ($ncond = 5, n = 400, f^* = -1679600$)	0.37	2.00	0.42	16.16	0.69	0.58
Random ($ncond = 0.1, n = 500, f^* = -762.53$)	0.40	0.05	0.09	0.10	0.86	1.23
Random ($ncond = 1, n = 500, f^* = -1133.68$)	0.4141	0.15	0.18	0.48	0.85	1.19
Random ($ncond = 5, n = 500, f^* = -1692549$)	0.63	4.04	0.87	48.75	1.24	1.25
Random ($ncond = 0.1, n = 600, f^* = -994.19$)	0.78	0.06	0.12	0.14	1.43	2.13
Random ($ncond = 1, n = 600, f^* = -1288.29$)	0.90	0.24	0.37	0.66	1.57	2.14
Random ($ncond = 5, n = 600, f^* = -2049820$)	1.16	9.72	1.27	48.69	1.97	2.14
Random ($ncond = 0.1, n = 700, f^* = -838.28$)	1.34	0.09	0.20	0.23	2.35	3.46
Random ($ncond = 1, n = 700, f^* = -1703.28$)	1.31	0.28	0.34	0.73	2.45	3.68
Random ($ncond = 5, n = 700, f^* = -2328669$)	2.49	20.45	2.84	164.35	3.92	3.45
Random ($ncond = 0.1, n = 800, f^* = -645.14$)	2.58	0.13	0.27	0.31	3.80	5.44
Random ($ncond = 1, n = 800, f^* = -1824.65$)	2.59	0.42	0.53	1.26	3.76	5.37
Random ($ncond = 5, n = 800, f^* = -2630417$)	3.58	32.21	3.72	108.13	5.76	5.47
Random ($ncond = 0.1, n = 900, f^* = -596.17$)	4.02	0.19	0.40	0.68	5.70	7.50
Random ($ncond = 1, n = 900, f^* = -1951.62$)	4.04	0.63	0.77	2.13	5.60	7.44
Random ($ncond = 5, n = 900, f^* = -2904251$)	5.16	46.01	5.87	145.91	7.39	7.64
Random ($ncond = 0.1, n = 1000, f^* = -1327.91$)	4.52	0.22	0.54	0.50	7.83	10.17
Random ($ncond = 1, n = 1000, f^* = -2677.47$)	4.58	0.66	0.95	1.68	7.93	10.00

continued on next page

continued from previous page						
Random ($ncond = 5, n = 1000, f^* = -3082720$)	6.82	72.48	8.19	143.93	10.02	9.93
Random ($ncond = 0.1, n = 1100, f^* = -1464.97$)	7.86	0.35	0.77	0.81	10.69	13.79
Random ($ncond = 1, n = 1100, f^* = -2061.31$)	7.98	1.01	1.45	3.50	10.31	13.57
Random ($ncond = 5, n = 1100, f^* = -4224564$)	10.84	85.98	12.03	213.99	14.21	13.77
Random ($ncond = 0.1, n = 1200, f^* = -1332.93$)	9.40	0.33	1.23	0.75	13.26	18.05
Random ($ncond = 1, n = 1200, f^* = -1978.65$)	9.02	0.96	2.29	3.32	13.49	19.19
Random ($ncond = 5, n = 1200, f^* = -4071507$)	11.08	90.54	11.95	187.50	16.90	19.31
Random ($ncond = 0.1, n = 1300, f^* = -2247.07$)	10.73	0.41	1.29	1.14	17.33	23.89
Random ($ncond = 1, n = 1300, f^* = -2698.07$)	11.67	1.46	2.91	4.43	17.73	24.35
Random ($ncond = 5, n = 1400, f^* = -1537.81$)	16.66	136.48	20.56	733.69	25.75	24.06
Random ($ncond = 0.1, n = 1400, f^* = -2247.07$)	12.03	0.43	1.57	1.14	20.94	30.16
Random ($ncond = 1, n = 1400, f^* = -2860.81$)	11.97	1.20	2.15	3.51	21.41	30.57
Random ($ncond = 5, n = 1400, f^* = -4446068$)	17.48	118.56	17.92	300.58	27.72	30.07
Random ($ncond = 0.1, n = 1500, f^* = -1287.82$)	17.70	0.50	2.16	1.14	25.30	36.80
Random ($ncond = 1, n = 1500, f^* = -2952.20$)	20.42	1.92	5.78	5.21	26.37	35.48
Random ($ncond = 5, n = 1500, f^* = -3811836$)	27.47	226.89	34.67	624.51	47.77	36.22
Random ($ncond = 0.1, n = 1600, f^* = -2321.86$)	23.34	0.61	2.68	1.76	29.76	48.56
Random ($ncond = 1, n = 1600, f^* = -3679.48$)	26.29	2.60	4.57	7.79	33.50	46.80
Random ($ncond = 5, n = 1600, f^* = -5524905$)	36.36	302.91	49.70	895.78	37.99	46.23
Random ($ncond = 0.1, n = 1700, f^* = -2994.18$)	34.09	0.89	4.37	2.48	37.61	51.78
Random ($ncond = 1, n = 1700, f^* = -3748.60$)	35.90	2.82	7.45	8.43	39.12	53.46
Random ($ncond = 5, n = 1700, f^* = -5223138$)	38.31	262.87	43.32	734.07	52.66	54.31
Random ($ncond = 0.1, n = 1800, f^* = -1354.26$)	29.42	0.68	4.43	2.13	43.82	64.54
Random ($ncond = 1, n = 1800, f^* = -3020.69$)	28.75	1.93	4.79	7.11	43.91	64.10
Random ($ncond = 5, n = 1800, f^* = -5748462$)	48.49	352.77	57.25	925.31	57.65	63.13
Random ($ncond = 0.1, n = 1900, f^* = -1466.46$)	47.30	0.90	5.20	2.10	52.67	76.41
Random ($ncond = 1, n = 1900, f^* = -4447.00$)	52.71	3.55	12.38	8.67	51.68	75.52
Random ($ncond = 5, n = 1900, f^* = -6772368$)	67.32	298.62	60.88	1043.96	72.19	75.50
Random ($ncond = 0.1, n = 2000, f^* = -1536.18$)	47.95	0.92	6.05	2.55	58.39	89.83
Random ($ncond = 1, n = 2000, f^* = -5177.91$)	47.75	2.75	7.32	7.57	60.00	91.10
Random ($ncond = 5, n = 2000, f^* = -6248198$)	74.03	395.21	70.79	711.86	103.15	89.13

Table 3.2: Random table results, $act\_prob = 0.9, up\_low\_prob = 0.5$

Prob. Name	Var.1	Var.2	Var.3	QUACAN	QPBOX	QLD
Random ($ncond = 0.1, n = 100, f^* = -213.04$)	0.00	0.01	0.01	0.00	0.00	0.00
Random ($ncond = 1, n = 100, f^* = -404.78$)	0.00	0.01	0.01	0.00	0.00	0.00
Random ($ncond = 5, n = 100, f^* = -494902$	0.24	0.01	0.01	0.00	0.01	0.00
Random ($ncond = 0.1, n = 200, f^* = -521.19$)	0.02	0.00	0.00	0.01	0.06	0.10
Random ($ncond = 1, n = 200, f^* = -972.38$)	0.02	0.01	0.01	0.01	0.06	0.11
Random ($ncond = 5, n = 200, f^* = -1236934$	0.03	0.10	0.02	2.20	0.06	0.10
Random ($ncond = 0.1, n = 300, f^* = -382.97$)	0.07	0.00	0.01	0.02	0.19	0.34
Random ($ncond = 1, n = 300, f^* = -891.98$)	0.07	0.02	0.02	0.04	0.20	0.34
Random ($ncond = 5, n = 300, f^* = -1614883$	0.08	0.10	0.05	2.45	0.19	0.33
Random ($ncond = 0.1, n = 400, f^* = -767.40$)	0.16	0.02	0.02	0.05	0.46	0.80
Random ($ncond = 1, n = 400, f^* = -1589.16$)	0.16	0.05	0.05	0.11	0.44	0.82
Random ($ncond = 5, n = 400, f^* = -2559084$	0.19	0.64	0.19	10.93	0.49	0.81
Random ($ncond = 0.1, n = 500, f^* = -1011.44$)	0.35	0.04	0.04	0.10	0.88	1.79
Random ($ncond = 1, n = 500, f^* = -1837.85$)	0.37	0.15	0.15	0.38	0.84	1.70
Random ($ncond = 5, n = 500, f^* = -2567167$	0.40	1.04	0.50	43.83	0.89	1.69
Random ($ncond = 0.1, n = 600, f^* = -874.34$)	0.70	0.05	0.05	0.15	1.59	2.93
Random ($ncond = 1, n = 600, f^* = -2851.60$)	0.64	0.13	0.13	0.35	1.48	3.16
Random ($ncond = 5, n = 600, f^* = -3219702$	0.77	2.29	0.78	35.44	1.57	2.99
Random ($ncond = 0.1, n = 700, f^* = -1718.10$)	1.49	0.09	0.09	0.27	2.53	4.79
Random ($ncond = 1, n = 700, f^* = -2322.62$)	1.53	0.32	0.33	1.10	2.74	4.84
Random ($ncond = 5, n = 700, f^* = -3907450$	1.31	3.78	1.23	104.96	2.49	5.08
Random ($ncond = 0.1, n = 800, f^* = -1847.63$)	2.39	0.11	0.11	0.26	3.77	7.53
Random ($ncond = 1, n = 800, f^* = -2810.93$)	2.42	0.35	0.36	0.96	3.82	7.47
Random ($ncond = 5, n = 800, f^* = -4336447$)	2.52	9.36	2.10	63.45	4.09	7.40
Random ($ncond = 0.1, n = 900, f^* = -1926.40$)	2.93	0.14	0.14	0.40	5.51	11.00
Random ($ncond = 1, n = 900, f^* = -2586.61$)	3.78	0.49	0.49	1.48	5.55	10.30
Random ($ncond = 5, n = 900, f^* = -4953068$)	4.00	11.63	3.21	129.98	5.79	10.33
Random ($ncond = 0.1, n = 1000, f^* = -1327.91$)	4.52	0.22	0.54	0.49	7.83	10.17
Random ($ncond = 1, n = 1000, f^* = -3738.74$)	3.57	0.43	0.43	1.45	7.27	15.08
Random ($ncond = 5, n = 1000, f^* = -5054535$)	3.91	9.30	3.07	180.95	8.53	15.12
Random ($ncond = 0.1, n = 1100, f^* = -1464.97$)	7.85	0.34	0.77	0.81	10.68	13.79
Random ($ncond = 1, n = 1100, f^* = -3223.46$)	5.68	0.66	0.66	2.62	9.66	20.50
Random ($ncond = 5, n = 1100, f^* = -5699765$)	7.86	19.25	6.08	300.41	11.82	18.90
Random ($ncond = 0.1, n = 1200, f^* = -1332.93$)	9.40	0.32	1.22	0.74	13.25	18.05
Random ($ncond = 1, n = 1200, f^* = -3052.69$)	8.40	0.73	0.73	3.01	12.95	25.86
Random ($ncond = 5, n = 1200, f^* = -6273425$)	7.69	19.61	5.51	209.04	13.56	27.15
Random ($ncond = 0.1, n = 1300, f^* = -2247.07$)	10.73	0.41	1.29	1.14	17.33	23.88
Random ($ncond = 1, n = 1300, f^* = -5120.61$)	9.76	0.96	0.96	3.52	16.73	34.33
Random ($ncond = 5, n = 1300, f^* = -6959699$)	10.34	26.22	8.07	365.77	17.86	33.73
Random ($ncond = 0.1, n = 1400, f^* = -1537.81$)	12.03	0.43	1.56	1.14	20.94	30.15
Random ($ncond = 10, n = 1400, f^* = --5818.67$)	12.53	1.29	1.30	3.32	20.75	40.12
Random ($ncond = 5, n = 1400, f^* = -6708835$)	13.08	55.20	11.63	363.79	21.79	39.71
Random ($ncond = 0.1, n = 1500, f^* = -1287.82$)	17.69	0.49	2.15	1.14	25.30	36.80
Random ($ncond = 1, n = 1500, f^* = 4595.18$)	20.42	1.59	1.60	4.72	25.96	49.28
Random ($ncond = 5, n = 1500, f^* = -8134587$)	21.22	68.82	16.76	387.54	26.71	49.82
Random ($ncond = 0.1, n = 1600, f^* = -2315.79$)	23.34	0.61	2.68	1.76	29.76	48.55
Random ($ncond = 1, n = 1600, f^* = -7025.33$)	23.88	2.06	2.07	7.24	33.57	63.22
Random ($ncond = 5, n = 1600, f^* = -8357410$)	22.53	64.27	22.67	663.67	33.521	69.35
Random ($ncond = 0.1, n = 1700, f^* = -2994.18$)	34.09	0.89	4.36	2.48	37.61	51.78
Random ($ncond = 1, n = 1700, f^* = -5597.19$)	30.40	1.95	1.95	5.84	37.00	73.40
Random ($ncond = 5, n = 1700, f^* = -8926662$)	30.31	106.51	24.51	488.59	45.261	88.21
Random ($ncond = 0.1, n = 1800, f^* = -4272.84$)	28.56	0.61	0.62	1.59	51.00	88.60

continued on next page

continued from previous page						
Random ($ncond = 1, n = 1800, f^* = -7183.86$)	28.57	1.65	1.66	5.11	43.85	88.02
Random ($ncond = 5, n = 1800, f^* = -8926662$)	30.31	106.52	24.51	488.60	45.26	88.21
Random ($ncond = 0.1, n = 1900, f^* = -6047.49$)	42.79	0.69	0.71	2.29	53.38	106.33
Random ($ncond = 1, n = 1900, f^* = -6443.84$)	42.80	2.48	2.48	9.87	49.56	106.76
Random ($ncond = 5, n = 1900, f^* = -9768911$)	44.13	97.03	29.84	979.92	54.03	105.90
Random ($ncond = 0.1, n = 2000, f^* = -5199.87$)	39.84	0.59	0.61	2.14	73.93	128.85
Random ($ncond = 1, n = 2000, f^* = -7383.81$)	40.05	2.36	2.41	8.78	62.69	128.56
Random ($ncond = 5, n = 2000, f^* = -10262511$)	42.04	147.56	32.04	613.54	67.91	128.35

Table 3.3: Random table results, $act\_prob = 0.1, up\_low\_prob = 0.5$

Prob. Name - Parameters	Var.1	Var.2	Var.3	QUACAN	QPOX	QLD
Random ($ncond = 0.1, n = 100, f^* = -33.75$)	0.00	0.00	0.00	0.00	0.01	0.00
Random ($ncond = 1, n = 100, f^* = -61.16$)	0.00	0.00	0.00	0.01	0.01	0.01
Random ($ncond = 5, n = 100, f^* = -198582$)	0.00	0.06	0.01	0.12	0.01	0.00
Random ($ncond = 0.1, n = 200, f^* = -61.26$)	0.03	0.00	0.02	0.01	0.06	0.03
Random ($ncond = 1, n = 200, f^* = -137.74$)	0.03	0.01	0.02	0.03	0.06	0.03
Random ($ncond = 5, n = 200, f^* = -198507$)	0.06	0.82	0.10	1.00	0.072	0.04
Random ($ncond = 0.1, n = 300, f^* = -98.13$)	0.12	0.01	0.06	0.01	0.22	0.12
Random ($ncond = 1, n = 300, f^* = -335.72$)	0.11	0.03	0.06	0.07	0.21	0.14
Random ($ncond = 5, n = 300, f^* = -294620$)	0.23	1.27	0.18	3.43	0.23	0.13
Random ($ncond = 0.1, n = 400, f^* = -81.61$)	0.28	0.03	0.14	0.05	0.52	0.30
Random ($ncond = 1, n = 400, f^* = -239.87$)	0.29	0.10	0.19	0.20	0.51	0.29
Random ($ncond = 5, n = 400, f^* = -371506$)	0.73	13.43	1.15	11.81	0.57	0.29
Random ($ncond = 0.1, n = 500, f^* = -165.39$)	0.58	0.05	0.27	0.09	0.98	0.61
Random ($ncond = 1, n = 500, f^* = -250.77$)	0.69	0.17	0.45	0.39	0.99	0.57
Random ($ncond = 5, n = 500, f^* = -503874$)	1.25	22.10	2.33	24.34	1.04	0.57
Random ($ncond = 0.1, n = 600, f^* = -120.53$)	1.12	0.08	0.52	0.12	1.70	1.03
Random ($ncond = 1, n = 600, f^* = -420.14$)	1.08	0.24	0.59	0.55	1.71	1.07
Random ($ncond = 5, n = 600, f^* = -646953$)	2.69	36.03	3.69	37.16	1.84	1.05
Random ($ncond = 0.1, n = 700, f^* = -215.51$)	2.05	0.12	0.92	0.28	2.83	1.72
Random ($ncond = 1, n = 700, f^* = -608.39$)	3.42	0.67	2.23	1.13	2.91	1.75
Random ($ncond = 5, n = 700, f^* = -866604$)	4.74	73.42	7.17	71.99	3.05	1.64
Random ($ncond = 0.1, n = 800, f^* = -379.52$)	3.96	0.20	1.67	0.31	4.58	2.97
Random ($ncond = 1, n = 800, f^* = -555.21$)	4.18	0.60	2.08	1.13	4.69	2.73
Random ($ncond = 5, n = 800, f^* = -874662$)	9.41	113.31	13.11	84.98	5.02	2.87
Random ($ncond = 0.1, n = 900, f^* = -162.00$)	6.42	0.27	2.79	0.52	6.89	3.85
Random ($ncond = 1, n = 900, f^* = -702.73$)	6.28	0.87	2.99	1.73	6.91	3.83
Random ($ncond = 5, n = 900, f^* = -1288215$)	14.06	159.45	18.67	144.42	7.42	3.77
Random ($ncond = 0.1, n = 1000, f^* = -385.19$)	8.10	0.33	4.12	0.49	9.67	4.92
Random ($ncond = 1, n = 1000, f^* = -744.63$)	7.80	0.93	4.17	1.99	9.58	5.23
Random ($ncond = 5, n = 1000, f^* = -947787$)	21.76	150.66	23.29	137.16	10.21	4.78
Random ($ncond = 0.1, n = 1100, f^* = -359.04$)	12.62	0.41	5.45	0.93	12.45	7.01
Random ($ncond = 1, n = 1100, f^* = -815.00$)	9.65	1.08	4.58	2.60	11.98	7.34
Random ($ncond = 5, n = 1100, f^* = -1398369$)	27.49	262.46	36.10	206.25	13.12	7.21
Random ($ncond = 0.1, n = 1200, f^* = -473.98$)	15.27	0.50	7.10	0.66	15.82	9.64
Random ($ncond = 1, n = 1200, f^* = -757.89$)	15.40	1.42	7.73	3.18	15.69	9.55
Random ($ncond = 5, n = 1200, f^* = -1719411$)	38.69	332.99	51.92	208.28	16.41	9.46
Random ($ncond = 0.1, n = 1300, f^* = -387.00$)	21.73	0.65	9.41	1.39	34.45	11.82
Random ($ncond = 1, n = 1300, f^* = -982.13$)	21.78	1.85	10.21	4.12	19.54	12.06
Random ($ncond = 5, n = 1300, f^* = -1549329$)	41.39	481.25	71.13	316.36	20.80	11.74
Random ($ncond = 0.1, n = 1400, f^* = -475.6$)	24.19	0.69	12.07	1.28	25.21	14.47
Random ($ncond = 1, n = 1400, f^* = -993.08$)	24.13	1.92	12.59	3.87	25.08	14.61
Random ($ncond = 5, n = 1400, f^* = -2063165$)	51.78	570.86	93.98	254.72	26.55	14.44
Random ($ncond = 0.1, n = 1500, f^* = -311.40$)	35.39	0.82	15.43	1.14	29.92	18.17
Random ($ncond = 1, n = 1500, f^* = -1076.26$)	34.83	2.37	15.71	6.19	31.23	19.00
Random ($ncond = 5, n = 1500, f^* = -1791434$)	80.21	628.68	111.98	510.94	32.73	18.67
Random ($ncond = 0.1, n = 1600, f^* = -285.54$)	42.09	1.09	18.83	2.06	39.03	25.64
Random ($ncond = 1, n = 1600, f^* = -1112.70$)	60.71	4.21	38.91	8.60	39.03	25.60
Random ($ncond = 5, n = 1600, f^* = -1794845$)	80.80	482.62	109.59	765.19	41.66	25.64
Random ($ncond = 0.1, n = 1700, f^* = -552.04$)	52.12	1.13	22.32	1.88	45.16	27.56
Random ($ncond = 1, n = 1700, f^* = -1238.63$)	74.48	4.33	45.94	7.83	45.62	28.54
Random ($ncond = 5, n = 1700, f^* = -1845689$)	125.11	559.45	133.35	630.73	47.11	27.14
Random ($ncond = 0.1, n = 1800, f^* = -579.5$)	56.08	1.13	28.06	1.67	51.92	33.74

continued on next page

continued from previous page						
Random ($ncond = 1, n = 1800, f^* = -1122.75$)	54.58	3.15	27.59	6.78	52.20	33.69
Random ($ncond = 5, n = 1800, f^* = -2751580$)	130.07	819.08	180.85	876.60	55.17	34.36
Random ($ncond = 0.1, n = 1900, f^* = -566.86$)	76.20	1.37	34.12	3.00	60.27	40.06
Random ($ncond = 1, n = 1900, f^* = -1194.64$)	75.6	3.92	35.20	9.03	60.39	40.63
Random ($ncond = 5, n = 1900, f^* = -2139161$)	145.81	902.77	162.77	712.32	62.87	39.40
Random ($ncond = 0.1, n = 2000, f^* = -621.46$)	78.98	1.36	37.10	1.91	69.26	49.50
Random ($ncond = 1, n = 2000, f^* = -1443.17$)	114.84	4.57	74.21	8.00	69.67	49.85
Random ($ncond = 5, n = 2000, f^* = -2589665$)	189.95	1191.16	269.69	670.03	73.12	50.21

Prob. Name	Var.1	Var.2	Var.3	QUACAN	QPBOX	QLD
SVM ($n = 100$), $f^* = -167.79$	0.00	0.00	0.00	0.01	0.01	0.00
SVM ($n = 200$), $f^* = -384.11$	0.0430	0.0352	0.0391	0.1523	0.0703	0.0742
SVM ($n = 300$), $f^* = -545.54$	0.1367	0.0977	0.1016	0.3945	0.2305	0.2539
SVM ($n = 400$), $f^* = -736.00$	0.3672	0.2891	0.3359	2.0039	0.5352	0.6289
SVM ($n = 500$), $f^* = -933.94$	0.7031	0.6133	0.7070	4.6914	1.0508	1.2734
SVM ($n = 600$), $f^* = -1073.77$	1.1797	0.8398	0.9727	6.5430	1.8516	2.3633
SVM ($n = 700$), $f^* = -1222.33$	2.2656	1.5078	1.8516	14.3320	3.0273	3.8125
SVM ($n = 800$), $f^* = -1323.44$	3.3789	1.8750	2.2539	21.7461	4.6836	6.1953
SVM ($n = 900$), $f^* = -1431.59$	5.6680	3.3984	3.8438	27.1602	7.3281	8.2031
SVM ($n = 1000$), $f^* = -1539.77$	7.2578	4.2930	5.0117	34.0078	10.3945	11.3281
SVM ($n = 2000$), $f^* = -2849.68$	68.7852	22.0078	36.2461	256.2266	77.2969	104.3086
SVM ($n = 3000$), $f^* = -4490.68$	263.3477	63.9688	151.4023	1068.9766	264.5586	354.4297
Tent ($n = 100$), $f^* = 0.0168$	0.0078	0.0039	0.0039	0.0039	$N.C$	0.0039
Tent ($n = 400$), $f^* = 0.3162$	0.322	0.132	0.217	$N.C$	$N.C$	0.248
Tent ($n = 900$), $f^* = 0.4442$	5.570	1.453	3.273	$N.C$	$N.C$	2.77
Tent ($n = 1600$), $f^* = 0.5023$	48.3008	9.5742	29.1133	$N.C$	$N.C$	20.5352
Tent ($n = 3600$), $f^* = 0.5455$	557.74	55.74	284.05	$N.C$	$N.C$	246.04
Tent ($n = 4900$), $f^* = 0.5540$	1333.51	150.49	696.21	$N.C$	$N.C$	617.58
Biharm ($n = 100$), $f^* = -0.0001$	0.0030	0.0030	0.0020	0.0040	0.0120	0.0130
Biharm ($n = 400$), $f^* = -0.0004$	0.1958	0.2090	0.1880	0.6450	0.6382	0.7539
Biharm ($n = 900$), $f^* = -0.0008$	4.2788	3.1328	2.9180	18.5229	10.4912	9.2886
Biharm ($n = 1600$), $f^* = -0.0015$	23.3280	17.8920	15.3110	119.6610	82.1220	60.8680
Biharm ($n = 2500$), $f^* = -0.0023$	106.1869	77.2411	60.5740	775.0340	333.9110	222.7870
Biharm ($n = 3600$), $f^* = -0.0033$	308.7246	271.4639	186.8857	2988.0826	1071.3447	684.5688
Biharm ($n = 4900$), $f^* = -0.0045$	816.13	705.52	484.45	8282.04	3067.21	1837.66
IMRT ($n = 2342$), $f^* = 0.0563$	54.22	33.11	40.56	85.11	67.88	73.22

Table 3.4: CPU times (secs). ($N.C$: No convergence)

Chapter 4

A Rectangular Trust Region Approach for Unconstrained and Bound Constrained Nonlinear Optimization

4.1 Summary

A trust region algorithm for unconstrained and bound constrained nonlinear optimization problems is presented. The trust region is a rectangular hyperbox in contrast with the commonly used hyperellipsoid. The resulting quadratic subproblems are solved approximatelly by an adaptation of Powell's dogleg method for rectangular trust regions and a the novel quadatratic programming algorithm presented in Chapter 3. Comparative results of numerical experiments are reported.

4.2 Introduction

Non-linear optimization plays an important role in many fields of science and engineering, in the industry, as well as in a plethora of practical problems. Frequently the optimization parameters are constrained inside a range imposed by the nature of the problem at hand. Developing methods for bound constrained optimization is hence quite useful. We refer to [27] (pp. 10–12) for a list of application areas. The most efficient optimization methods are based on Newton's method where a quadratic model is adopted as a local approximation to the objective function. Two general approaches have been followed. One uses a line–search along a properly selected descent direction, while the other permits steps of restricted size in an effort to maintain the reliability of the quadratic approximation. The approaches in this second class, bear the generic name Trust-Region techniques. In this article we deal with a method of that type.

We develop a method that adopts a rectangular shape for the trust region. This geometry has the obvious advantage of the linearity of the subproblem constraints and

in addition allows effortless adaptation to bound constrained problems. The emerging quadratic subproblems are of the sort:

$$\min_s \frac{1}{2} s^T B s + s^T g \quad \text{subject to:} \quad a_i \leq s_i \leq b_i \tag{4.1}$$

a modification of Powell's [125] dogleg technique is developed to obtain an approximate solution and an exact technique based on quadratic algorithm in Chapter 3.

We embed this scheme in a quasi–Newton framework that uses a positive definite approximation to the Hessian matrix. This renders the problem in Eq.4.1 a strictly convex one, and hence the dogleg technique is applicable.

In Section 2, we describe in brief the trust region class of algorithms along the lines of Conn, Gould and Toint [27]. In Sections 3 and 4 we present the proposed methodology along with our experimental results. Finally our conclusions are layed out in Section 5.

4.3 Trust Region Methods

Trust region methods fall in the category of sequential quadratic programming. The algorithms in this class are iterative procedures in which the objective function $f(x)$ is represented by a quadratic model inside a suitable neighborhood (the trust region) of the current iterate, as implied by the Taylor series expansion. This local model of $f(x)$ at the k^{th} iteration can be written as:

$$f(x_k + s) \approx m_k(s) = f(x_k) + g_k^T s + \frac{1}{2} s^T B_k s \tag{4.2}$$

where $g_k = \nabla f(x_k)$ and B_k is a symmetric approximation to $\nabla^2 f(x_k)$.

The trust region may be defined by:

$$\mathbf{T}_k = \{x \in \Re^n \mid ||x - x_k|| \leq \Delta_k\} \tag{4.3}$$

It is obvious that different choices for the norm lead to different trust region shapes. The Euclidean norm $||\cdot||_2$, corresponds to a hypershpere, while the $||\cdot||_\infty$ norm defines a hyperbox.

Given the model and the trust region, we seek a step s_k with $||s_k|| \leq \Delta_k$, such that the model is sufficiently reduced in value. Using this step we compare the reduction in the model to that in the objective function. If they agree to a certain extend, the step is accepted and the trust region is either expanded or remains the same. Otherwise the step is rejected and the trust region is contracted. The basic trust region algorithm is sketched in Alg. 4.7

Algorithm 4.7 Basic trust region

S0: Pick the initial point and trust region parameter x_0 and Δ_0, and set $k = 0$.

S1: Construct a quadratic model:
$m_k(s) \approx f(x_k + s)$

S2: Calculate s_k with $||s_k|| \leq \Delta_k$, so as to sufficiently reduce m_k.

S3: Compute the ratio of actual to expected reduction, $r_k = \frac{f(x_k)-f(x_k+s_k)}{m_k(0)-m_k(s_k)}$. This value will determine if the step will be accepted or not and the update for Δ_k.

S4: Increment $k \leftarrow k + 1$ and repeat from S1.

4.4 Dogleg approximate solution

As mentioned in the introduction, our algorithm is a modification of Powell's dogleg method suitable for rectangular trust regions. The dogleg path is defined as:

$$s(a) = \begin{cases} aC & \text{for } 0 \leq a \leq 1 \\ C + (a-1)(N-C) & \text{for } 1 \leq a \leq 2 \end{cases}$$

where $C = -\frac{g_k^T g_k}{g_k^T B_k g_k} g_k$ is the Cauchy step, and $N = -H_k^{-1} g_k$ is the Newton step, that is the unconstrained minimizer of m_k. In Fig. 4.1 we show the dogleg path for the cases of the $||\cdot||_\infty$ and the $||\cdot||_2$ norm. The quadratic model $m_k(s(a))$, decreases monotonically as a increases assuming that B_k is *positive definite*. In the original paper, the dogleg path was truncated as soon as it intersected with the trust region boundary. We distinguish the three following cases:

Case 1:	$N \in T_k$
Case 2:	$C \in T_k$ and $N \notin T_k$
Case 3:	$C \notin T_k$ and $N \notin T_k$

In our algorithm cases 1 and 2 are treated the same way as in Powell's original paper[125]. However in case 3, we prefer a slightly different approach. Instead of taking the maximum feasible step along C ($PC = bC,\ b \leq 1$) which is the case in the original algorithm, we proceed further towards N in the direction $N - PC$ until a bound is encountered. In Fig.4.2 we show such a case when the trust region is a hyperbox. The definition of the dogleg path under this modification is:

$$s(a) = \begin{cases} aC & \text{for } 0 \leq a \leq b \\ bC + (a-b)(N-bC) & \text{for } b \leq a \leq 1+b \end{cases}$$

where $b = \frac{||PC||_2}{||C||_2} \in [0, 1]$. It can be trivially shown that along this path $m_k(s(a))$ monotonically decreases, reaching so a lower value for the model.

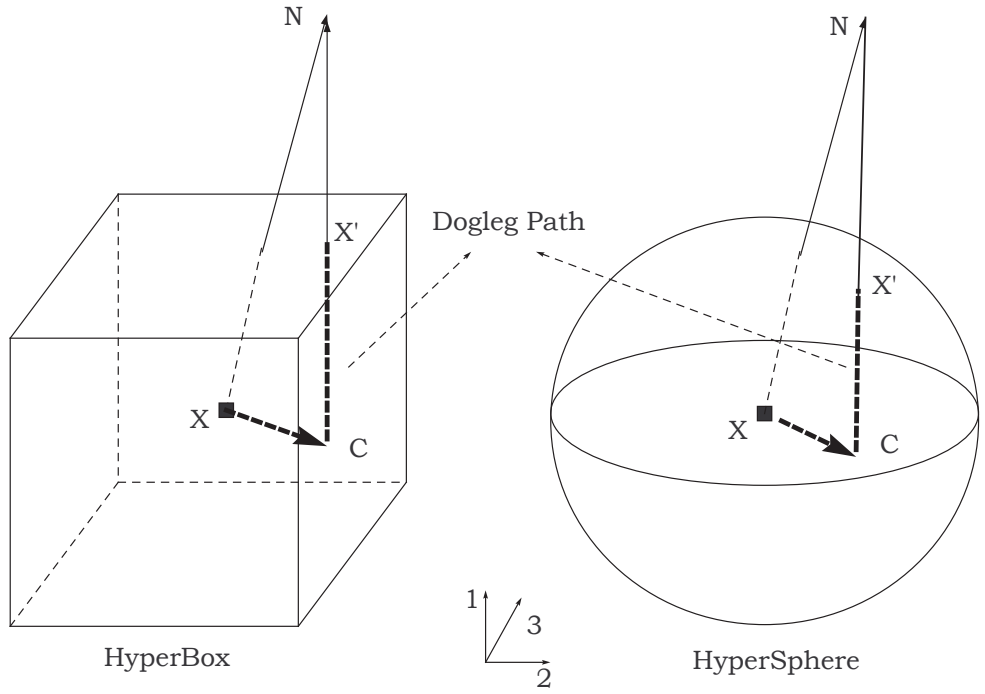


Figure 4.1: Dogleg path

We wish to apply our method to the more general problem:

$$\min_{x \in R_n} f(x) \text{ subject to: } l_i \leq x_i \leq u_i \tag{4.4}$$

This covers both unconstrained and bound constrained problems.

We employ BFGS updates to guarantee the positive definiteness of the approximation B_k, to the Hessian matrix. We construct the model $m_k(s)$ as described in Section 2, and we omit the constant term $f(x_k)$ in Eq. 4.11.

The trust region at the k^{th} iteration is defined as:

$$\mathbf{T}_k = \{x \in \Re^n \mid ||x - x_k||_\infty \leq \Delta_k\} \tag{4.5}$$

and thus the dogleg step must be constrained by:

$$||s_k||_\infty \leq \Delta_k \tag{4.6}$$

in other words:

$$-\Delta_k \leq \max_k(s_k) \leq \Delta_k \tag{4.7}$$

From Eq. 4.7, and the fact that the new point $x_k + s_k$ must be feasible, the subproblem can be restated as:

$$\min_{s \in R^n} m_k(s) = \tfrac{1}{2} s^T B_k s + s^T g_k$$
$$\max[l_i - x_i, -\Delta_k] \leq s_i \leq \min[u_i - x_i, \Delta_k]$$

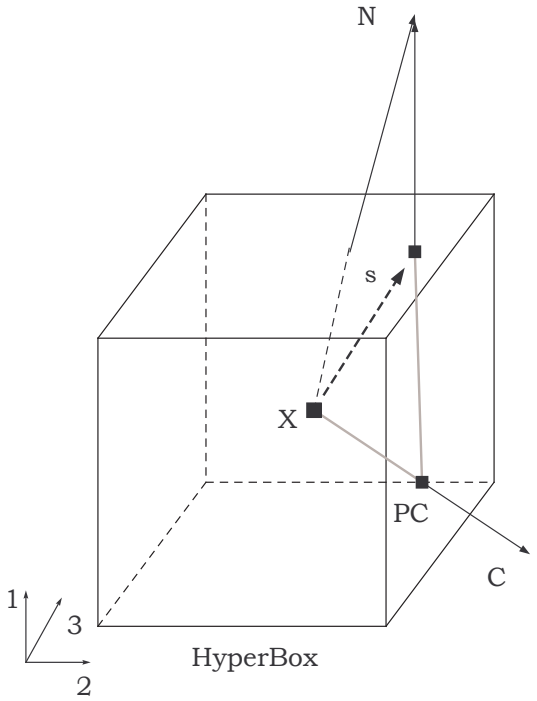


Figure 4.2: Our approach in Case 3

It is worth mentioning that when the original problem involves bound constraints, the trust region shape is a hyperectangle. When no bounds are present the trust region is just a hypercube.

Special care must be taken when an iterate x_k reaches a bound. We define the *active set* at a point x, as the set of indices:

$$\begin{aligned} A(x) \;=\; & \left\{ i \mid x_i = u_i \text{ and } \frac{\partial f}{\partial x_i} < 0 \right\} \cup \\ & \left\{ i \mid x_i = l_i \text{ and } \frac{\partial f}{\partial x_i} > 0 \right\} \end{aligned} \tag{4.8}$$

When $A(x_k) \neq \emptyset$ the dogleg step s_k that is computed from the quadratic subproblem may lead outside the feasible region and hence no progress can be achieved. To deal with this situation, we reduce the dimension of the subproblem by excluding the minimization parameters that belong to the active set. Let m the number of parameters in the active set. The dimension of the subproblem is reduced to $n - m$. In Fig.4.3, we present a case that progress would have been impossible without the reduction.

Our algorithm is presented in Alg. 4.8.

4.4.1 Experimental results

In order to investigate the behavior of the DOGBOX algorithm, we have performed a substantial amount of numerical testing. We have attempted to solve 35 unconstrained and bound constrained test problems taken from the More collection [106].

Algorithm 4.8 DOGBOX

S0: Pick the initial point and trust region parameter x_0 and Δ_0, and set $k = 0$

S1: If active constrains exist, reduce the subproblem's dimension. $\tilde{B}_k$ and $\tilde{g_k}$ are reduced quantities.

S2: Construct the quadratic model around x_k:

$$m_k(s) = 1/2\tilde{s}^T \tilde{B}_k \tilde{s} + \tilde{s}^T \tilde{g}_k$$
$$\max_i[l_i - x_i, -\Delta] \leq \tilde{s}_i \leq \min_i[u_i - x_i, \Delta]$$

S3: Calculate dogleg step $\tilde{s}_k$

if $N = -\tilde{B}_k^T \tilde{g}_k$ is feasible then
  $\tilde{s}_k = N$
else
  if $C = -\frac{\tilde{g}_k^T \tilde{g}_k}{\tilde{g}_k^T \tilde{B}_k \tilde{g}_k} \tilde{g}_k$ is feasible then
    find the maximum α such that
    $C + \alpha * (N - C) \ \in T_k$
    $\tilde{s}_k = C + \alpha * (N - C)$
  else
    find the maximum β such that
    $PC \equiv \beta C \in T_k$
    find the maximum α such that
    $PC + \alpha * (N - PC) \ \in T_k$
    $\tilde{s}_k = PC + \alpha * (N - PC)$
  end if
end if

S4: Using the reduced step $\tilde{s}_k$, calculate the full space step s_k and the ratio r_k.

S5: Choose the new point x_{k+1} according to:

if $r_k \leq 0.1$ then
  $x_{k+1} = x_k$
else
  $x_{k+1} = x_k + s_k$
endif

S6: Update trust region Δ_k according to:

if $r_k < 0.25$ then
  $\Delta_{k+1} = ||s_k||/4$
else if $r_k > 0.75$ and $||s_k|| = \Delta_k$ then
  $\Delta_{k+1} = 2\Delta_k$
else
  $\Delta_{k+1} = \Delta_k$
endif

S7: Increment $k \leftarrow k + 1$ and repeat from S1.

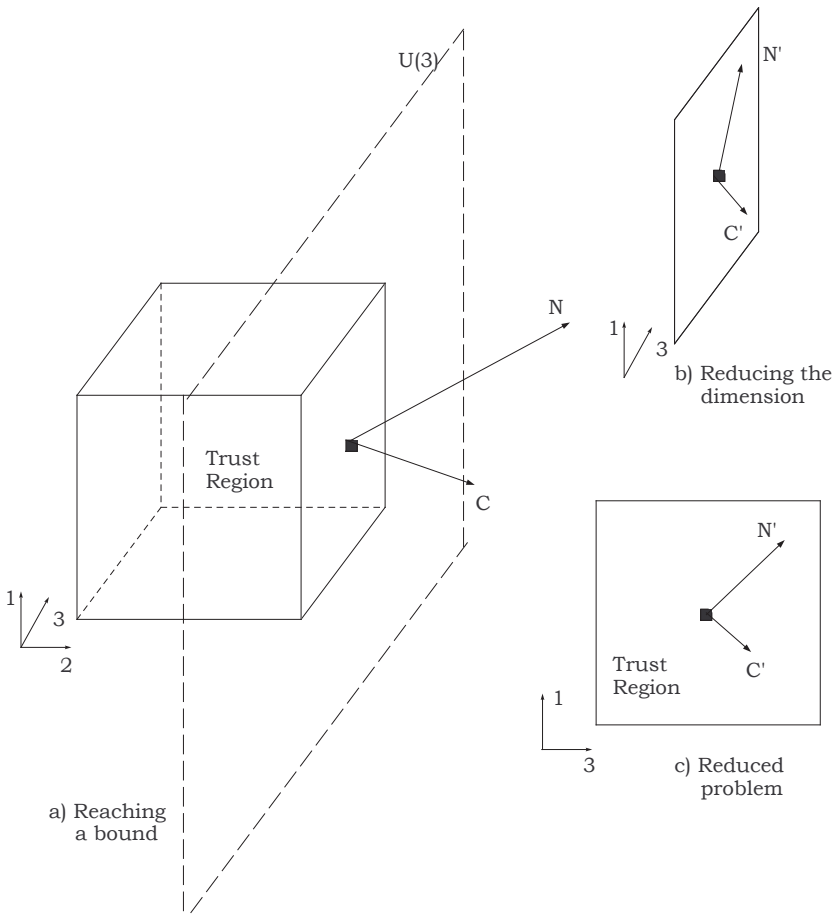


Figure 4.3: Bound handling

The implementation was written in double precision FORTRAN 77, and was incorporated in the Merlin Optimization Environment [114].

In the unconstrained case we compare our hyperbox-dogleg method to the originally proposed dogleg that is implemented in Merlin (command *TRUST*). We start the minimization from the points recommended by More (Test Points 1 and 2). Both methods use BFGS updates to approximate the Hessian matrix and use exactly the same scheme to treat the trust region. The stopping criteria are identical as well. The aim of these experiments is to verify that, in the unconstrained case, our method is as effective as the original one proposed by Powell. The results are shown in Table 4.1, were the number of iterations ("*It.*"), the function calls ("*FC*") and the gradient calls ("*GC*") are reported for each method. In this table, "$*$" denotes that the two methods ended up in different minima, and hence any comparison is meaningless.

For the bound constrained tests, the bounds were generated by the following two schemes, were x stands for the initial starting points recommended by More.

$$(1-r)x \leq x \leq (1+r)x, \quad x \in R^n, \ 0 < r < 1 \tag{4.9}$$

$$x - c \leq x \leq x + c, \quad x, c \in R^n \tag{4.10}$$

Care was taken that in our experiments the unconstrained minimum was feasible in some, but not in all, cases. In the bound constrained case, we compare our method against Merlin's *TRUST* method and the well known *Tolmin*[127] algorithm which is also included in the Merlin distribution. The results of the two bound constrained tests are shown in Table 4.2 for Eq.4.9 and Table 4.3 for Eq.4.10. We should point out that the symbol "−" in these tables means that the method did not converge to the solution.

The presented results for the unconstrained case, offer a useful insight about the behavior of our algorithm. It seems that our method performs better (although marginally) than the original dogleg-trust region method in the majority of the test problems. We can infer that our slight modification in the dogleg path, is responsible for that.

In the bound constrained case results, we witness a dramatic improvement when we compare *TRUST* to our implementation. This is expected due to the hyberbox nature of our approach, that helps dealing with bounds in a straightforward way. Another conclusion that can be drawn is that our method behaves similarly to *Tolmin* in most cases, and overall perfoms slightly better.

Table 4.1: Unconstrained case

Problem	Test Point 1						Test Point 2					
	TRUST			DOGBOX			TRUST			DOGBOX		
Name	It.	FC	GC	It.	FC	GC	It.	FC	GC	It.	FC	GC
ROSEN	40	47	41	37	44	38	26	31	27	27	34	28
FRE-ROT	13	40	13	14	34	14	14	40	14	14	40	14
BRO-B-S	34	43	35	34	43	35	37	50	37	37	50	38
BEA	19	20	19	18	19	18	16	19	16	18	19	20
JEN-SAM	1	7	2	1	7	2	1	17	2	1	17	2
HEL-VAL	33	43	34	30	38	30	*	*	*	*	*	*
BARD	23	42	23	20	39	20	23	41	23	22	40	22
GAUS	7	19	7	7	18	8	15	15	16	13	14	14
GULF	1	2	1	1	2	1	2	22	2	2	22	2
BOX3	37	39	38	39	40	42	52	57	53	51	57	52
POW-SIN	67	71	68	88	89	94	92	97	93	71	74	72
WOOD	36	44	36	37	46	37	24	30	25	34	43	35
KOW-OSB	33	49	33	34	49	34	41	56	41	42	62	42
BRO-DEN	37	65	37	41	69	41	42	69	42	49	83	49
OSB1	67	91	67	69	92	69	111	142	111	101	133	101
BIG-E6	44	62	44	46	69	46	41	57	41	40	58	40
OSB2	66	89	66	61	89	61	49	75	49	40	63	40
WATS	159	177	159	131	156	131	180	216	180	188	225	188
X-ROS	92	107	92	104	123	104	95	115	95	98	121	98
X-POW-S	204	218	204	221	247	231	254	274	254	204	221	204
PENI	202	226	202	172	217	172	57	81	57	38	61	38
PENII	203	241	203	270	300	271	259	300	260	253	300	254
VAR-DIM	15	21	15	25	31	25	23	28	23	24	29	24
TRIG	34	48	34	30	46	30	36	50	36	39	54	39
BR-A-LIN	19	36	19	18	34	18	1	1	1	1	1	1
DISC-INT	29	30	29	33	35	33	29	29	29	34	37	35
LIN-FR	3	5	4	2	3	2	3	4	3	2	3	2
LIN-R1	3	25	3	3	25	3	3	27	3	3	25	3
LIN-R10	3	24	3	4	28	4	5	28	5	4	27	4
CHEB	38	55	38	40	63	40	150	186	150	106	144	106

Table 4.2: Constrained case (1)

Problem	Test Point 1								Test Point 2							
	TRUST			DOGBOX			TOLMIN		TRUST			DOGBOX			TOLMIN	
Name	It.	FC	GC	It.	FC	GC	FC	GC	It.	FC	GC	It.	FC	GC	FC	GC
ROSEN	6	39	6	2	2	2	3	2	5	11	6	2	2	2	3	2
FRE-ROT	39	84	39	2	2	2	3	2	1	2	1	2	2	2	3	2
POW-B-S	11	29	11	2	2	2	3	2	13	32	13	3	3	3	5	4
BROW-B-S	8	65	8	3	48	3	37	36	6	63	6	3	3	3	4	3
BEAL	46	93	46	3	3	3	4	3	1	2	1	3	3	3	4	3
JEN-SAM	1	2	1	3	3	3	5	4	1	13	2	3	3	3	6	5
GAUS	15	16	15	7	18	8	14	15	56	73	56	9	9	9	31	32
MEYE	63	117	63	20	47	20	25	24	-	-	-	12	12	12	23	22
GULF	50	100	50	6	6	6	8	7	50	97	50	10	10	10	8	7
BOX3	5	5	6	4	4	4	5	4	7	32	7	4	4	4	5	4
POW-SI	-	-	-	4	4	4	5	4	-	-	-	3	3	3	4	3
KOW-OSB	68	84	68	13	13	13	20	19	58	105	58	7	7	7	8	7
BRO-DEN	1	9	2	3	3	3	7	6	1	12	2	3	3	3	5	4
OSB1	66	115	66	250	339	250	19	18	-	-	-	11	11	11	16	15
BIG-EX	53	70	53	10	11	10	19	18	30	46	30	16	32	16	27	26
OSB2	73	91	73	33	53	33	59	58	58	76	58	14	30	14	22	21
WATS	1	0	0	0	0	0	0	0	1	3	2	21	21	21	42	41
X-ROSE	7	33	7	2	2	2	3	2	6	40	6	2	2	2	3	2
X-POW-S	-	-	-	6	6	6	6	5	-	-	-	3	3	3	4	3
PEN1	2	36	2	5	5	5	6	5	1	2	1	5	5	5	6	5
PEN2	50	97	50	5	5	5	10	9	90	136	90	5	5	5	7	6
VAR-DIM	22	82	22	10	10	10	11	10	20	70	20	10	10	10	11	10
TRIG	61	78	61	19	36	19	33	32	53	99	53	11	11	11	13	12
BR-A-LIN	8	41	8	3	3	3	4	3	0	0	0	0	0	0	0	0
DISC-BOUN	-	-	-	20	35	20	39	38	0	0	0	0	0	0	0	0
LIN-FR	46	90	46	2	2	2	3	2	45	89	45	2	2	2	3	2
LIN-R1	1	5	2	11	11	11	12	11	1	7	2	11	11	11	12	11
LIN-R10	1	4	2	9	9	9	10	9	1	6	2	9	9	9	10	9
CHEB	49	69	49	44	66	44	60	59	74	143	74	52*	96	52	86	85

4.5 Boxcqp exact solution

We present in this section a trust region method for non-linear optimization with bound constraints, where the trust region is a hyperbox, in contrast with the usual hypersphere or hyperellipsoid shapes. The rectangular trust region is natural for problems with bound constraints, because even when there is an overlap with the feasible region, its geometry is preserved. Trust region methods fall in the category of sequential quadratic programming. These algorithms are iterative and the objective function $f(x)$ (assumed to be twice continuously differentiable), is approximated in a proper neighborhood of the current iterate (the trust region), by a quadratic model. Namely, at the k^{th} iteration the model is given by:

$$f(x^k + s) \approx m^{(k)}(s) = f(x^{(k)}) + s^T g^{(k)} + \frac{1}{2} s^T B^{(k)} s \tag{4.11}$$

where $g^{(k)} = \nabla f(x^{(k)})$ and $B^{(k)}$ in the case of Newton's method is a positive definite modification of the Hessian, while in the case of quasi-Newton methods is a positive definite matrix produced by a low rank relevant update.

Table 4.3: Constrained case (2)

Problem Name	Test Point 1								Test Point 2							
	TRUST			DOGBOX			TOLMIN		TRUST			DOGBOX			TOLMIN	
	It.	FC	GC	It.	FC	GC	FC	GC	It.	FC	GC	It.	FC	GC	FC	GC
ROSEN	24	60	24	14	17	14	17	16	26	66	26	5	5	5	11	10
FREU-ROT	16	44	16	9	9	9	28	27	42	99	42	3	3	3	5	4
BROW-B-S	7	64	7	3	44	3	15	14	14	78	14	3	3	3	4	3
BEAL	-	-	-	8	24	8	20	19	1	2	1	2	2	2	3	2
JEN-SAM	-	-	-	27	54	27	55	54	-	-	-	20	50	20	67	66
GAUS	9	9	9	7	18	7	14	13	49	52	41	14	28	15	33	32
MEYE	62	116	62	12	24	12	27	26	79	170	79	47	60	47	22	21
BOX3	6	35	6	4	4	4	5	4	7	35	7	5	5	5	6	5
POW-SI	63	94	63	17	37	17	45	44	-	-	-	10	41	10	23	22
KOW-OSB	36	52	36	33	43	33	48	47	41	57	41	44	64	44	46	45
BRO-DEN	-	-	-	5	5	5	10	9	-	-	-	5	5	5	8	7
OSB1	76	102	76	70	93	70	103	102	300	300	300	91	124	91	99	98
BIG-EX	31	47	31	21	38	21	31	30	28	45	28	17	34	17	36	35
OSB2	84	106	84	54	78	54	91	90	53	69	53	19	37	19	39	38
X-ROSE	34	72	34	36	53	36	41	40	77	130	77	11	40	11	42	41
X-POW-S	79	118	79	32	61	32	56	55	-	-	-	9	34	9	27	26
PEN1	1	2	1	7	7	7	13	12	1	2	1	5	5	5	6	5
VAR-DIM	202	258	202	1	2	1	3	2	-	-	-	18	19	18	37	36
TRIG	28	41	28	32	47	32	53	52	*	*	*	*	*	*	*	*
BR-A-LIN	-	-	-	17	35	17	34	33	1	0	0	1	0	0	0	0
DISC-BOUN	27	30	27	33	35	33	46	45	32	34	32	34	37	35	50	49
DISC-INT	25	25	25	25	25	25	31	30	27	27	27	26	26	26	33	32
BROY-TRI	60	78	60	64	98	64	48	47	27	69	27	12	12	12	30	29
BROY-BAN	88	119	88	68	109	68	88	87	26	76	26	11	11	11	26	25
LIN-FR	48	93	48	2	2	2	3	2	47	92	47	2	2	2	3	2
LIN-R1	-	-	-	12	12	12	21	20	1	9	2	11	11	11	12	11
LIN-R10	-	-	-	10	10	10	19	18	1	8	2	9	9	9	10	9
CHEB	44	66	44	42	66	42	53	52	*	*	*	*	*	*	*	*

The trust region may be defined by:

$$\mathbf{T}^{(k)} = \{x \in \Re^n \mid ||x - x^{(k)}|| \leq \Delta^{(k)}\} \tag{4.12}$$

It is obvious that different choices for the norm lead to different trust region shapes. The Euclidean norm $||\cdot||_2$, corresponds to a hypershpere, while the $||\cdot||_\infty$ norm defines a hyperbox.

Given the model and the trust region, we seek a step $||s^{(k)}|| \leq \Delta^{(k)}$, that minimizes $m^{(k)}(s)$. We compare the actual reduction $\delta f^{(k)} = f(x^{(k)}) - f(x^{(k)} + s^{(k)})$, to the model reduction $\delta m^{(k)} = m^{(k)}(0) - m^{(k)}(s^{(k)})$. If they agree to a certain extend, the step is accepted and the trust region is either expanded or remains the same. Otherwise the step is rejected and the trust region is contracted. The code for this method is to be added to the upcoming version 4.0 of Merlin Optimization Environment [114]. The basic trust region algorithm is sketched in Algorithm 2.

Algorithm 2 Basic trust region

1. Pick the initial point and trust region parameter $x^{(0)}$ and $\Delta^{(0)}$, and set $k = 0$.

2. Construct a quadratic model:

$$f(x^k + s) \approx m^{(k)}(s) = f(x^{(k)}) + s^T g^{(k)} + \frac{1}{2} s^T B^{(k)} s$$

3. Minimize $m^{(k)}(s)$ and hence determine $||s^{(k)}|| \leq \Delta^{(k)}$

4. Compute the ratio of actual to expected reduction: $r^{(k)} = \frac{\delta f^{(k)}}{\delta m^{(k)}}$, and update the trust region, following the strategy of J. E. Dennis, R. B. Schnabel (1996) (Appendix A, page 338).

5. Increment $k \leftarrow k + 1$ and repeat from 1.

Consider the bound constrained problem:

$$\min_x f(x), \quad \text{subject to: } l_i \leq x_i \leq u_i$$

(The unconstrained case is obtained by letting $u_i = -l_i \to \infty$.)
Let $x^{(k)}$ be the k-th iterate of the trust region algorithm.

Hence step 3 of Algorithm 2 becomes:

$$\begin{aligned} \min_s \; & m^{(k)}(s) = s^T g^{(k)} + \frac{1}{2} s^T B^{(k)} s \\ \text{subject to: } & \max(l_i - x_i^{(k)}, -\Delta^{(k)}) \leq s_i \leq \min(u_i - x_i^{(k)}, \Delta^{(k)}) \end{aligned} \tag{4.13}$$

In the unconstrained case, our experiments (that used a BFGS update), showed similar performance to spherical trust region implementations. For the bound constrained case our method is obviously superior, since it maintains the simplicity of the rectangular trust region, where our efficient quadratic solver is applicable [155]. Note that the trust region formed by the intersection of a sphere with the rectangular box defined by the parameter bounds, is not easy to treat. Concluding, further numerical tests showed that our method performs similarly to active set methods with line–search.

4.5.1 Experimental results

Table 4.4: Unconstrained case

Problem Name	Test Point 1						Test Point 2					
	TRUST			TRUSTQP			TRUST			TRUSTQP		
	It.	FC	GC	It.	FC	GC	It.	FC	GC	It.	FC	GC
ROSEN	40	47	41	37	44	38	26	31	27	27	34	28
FRE-ROT	13	40	13	14	34	14	14	40	14	14	40	14
BRO-B-S	34	43	35	34	43	35	37	50	37	37	50	38
BEA	19	20	19	18	19	18	16	19	16	18	19	20
JEN-SAM	1	7	2	1	7	2	1	17	2	1	17	2
HEL-VAL	33	43	34	30	38	30	*	*	*	*	*	*
BARD	23	42	23	20	39	20	23	41	23	22	40	22
GAUS	7	19	7	7	18	8	15	15	16	13	14	14
GULF	1	2	1	1	2	1	2	22	2	2	22	2
BOX3	37	39	38	39	40	42	52	57	53	51	57	52
POW-SIN	67	71	68	88	89	94	92	97	93	71	74	72
WOOD	36	44	36	37	46	37	24	30	25	34	43	35
KOW-OSB	33	49	33	34	49	34	41	56	41	42	62	42
BRO-DEN	37	65	37	41	69	41	42	69	42	49	83	49
OSB1	67	91	67	69	92	69	111	142	111	101	133	101
BIG-E6	44	62	44	46	69	46	41	57	41	40	58	40
OSB2	66	89	66	61	89	61	49	75	49	40	63	40
WATS	159	177	159	131	156	131	180	216	180	188	225	188
X-ROS	92	107	92	104	123	104	95	115	95	98	121	98
X-POW-S	204	218	204	221	247	231	254	274	254	204	221	204
PENI	202	226	202	172	217	172	57	81	57	38	61	38
PENII	203	241	203	270	300	271	259	300	260	253	300	254
VAR-DIM	15	21	15	25	31	25	23	28	23	24	29	24
TRIG	34	48	34	30	46	30	36	50	36	39	54	39
BR-A-LIN	19	36	19	18	34	18	1	1	1	1	1	1
DISC-INT	29	30	29	33	35	33	29	29	29	34	37	35
LIN-FR	3	5	4	2	3	2	3	4	3	2	3	2
LIN-R1	3	25	3	3	25	3	3	27	3	3	25	3
LIN-R10	3	24	3	4	28	4	5	28	5	4	27	4
CHEB	38	55	38	40	63	40	150	186	150	106	144	106

Table 4.5: Constrained case (1)

Problem	Test Point 1								Test Point 2							
	TRUST			TRUSTQP			TOLMIN		TRUST			TRUSTQP			TOLMIN	
Name	It.	FC	GC	It.	FC	GC	FC	GC	It.	FC	GC	It.	FC	GC	FC	GC
ROSEN	6	39	6	2	2	2	3	2	5	11	6	2	2	2	3	2
FRE-ROT	39	84	39	2	2	2	3	2	1	2	1	2	2	2	3	2
POW-B-S	11	29	11	2	2	2	3	2	13	32	13	3	3	3	5	4
BROW-B-S	8	65	8	3	48	3	37	36	6	63	6	3	3	3	4	3
BEAL	46	93	46	3	3	3	4	3	1	2	1	3	3	3	4	3
JEN-SAM	1	2	1	3	3	3	5	4	1	13	2	3	3	3	6	5
GAUS	15	16	15	7	18	8	14	15	56	73	56	9	9	9	31	32
MEYE	63	117	63	20	47	20	25	24	-	-	-	12	12	12	23	22
GULF	50	100	50	6	6	6	8	7	50	97	50	10	10	10	8	7
BOX3	5	5	6	4	4	4	5	4	7	32	7	4	4	4	5	4
POW-SI	-	-	-	4	4	4	5	4	-	-	-	3	3	3	4	3
KOW-OSB	68	84	68	13	13	13	20	19	58	105	58	7	7	7	8	7
BRO-DEN	1	9	2	3	3	3	7	6	1	12	2	3	3	3	5	4
OSB1	66	115	66	250	339	250	19	18	-	-	-	11	11	11	16	15
BIG-EX	53	70	53	10	11	10	19	18	30	46	30	16	32	16	27	26
OSB2	73	91	73	33	53	33	59	58	58	76	58	14	30	14	22	21
WATS	1	0	0	0	0	0	0	0	1	3	2	21	21	21	42	41
X-ROSE	7	33	7	2	2	2	3	2	6	40	6	2	2	2	3	2
X-POW-S	-	-	-	6	6	6	6	5	-	-	-	3	3	3	4	3
PEN1	2	36	2	5	5	5	6	5	1	2	1	5	5	5	6	5
PEN2	50	97	50	5	5	5	10	9	90	136	90	5	5	5	7	6
VAR-DIM	22	82	22	10	10	10	11	10	20	70	20	10	10	10	11	10
TRIG	61	78	61	19	36	19	33	32	53	99	53	11	11	11	13	12
BR-A-LIN	8	41	8	3	3	3	4	3	0	0	0	0	0	0	0	0
DISC-BOUN	-	-	-	20	35	20	39	38	0	0	0	0	0	0	0	0
LIN-FR	46	90	46	2	2	2	3	2	45	89	45	2	2	2	3	2
LIN-R1	1	5	2	11	11	11	12	11	1	7	2	11	11	11	12	11
LIN-R10	1	4	2	9	9	9	10	9	1	6	2	9	9	9	10	9
CHEB	49	69	49	44	66	44	60	59	74	143	74	52*	96	52	86	85

Table 4.6: Constrained case (2)

Problem Name	Test Point 1								Test Point 2							
	TRUST			TRUSTQP			TOLMIN		TRUST			TRUSTQP			TOLMIN	
	It.	FC	GC	It.	FC	GC	FC	GC	It.	FC	GC	It.	FC	GC	FC	GC
ROSEN	24	60	24	14	17	14	17	16	26	66	26	5	5	5	11	10
FREU-ROT	16	44	16	9	9	9	28	27	42	99	42	3	3	3	5	4
BROW-B-S	7	64	7	3	44	3	15	14	14	78	14	3	3	3	4	3
BEAL	-	-	-	8	24	8	20	19	1	2	1	2	2	2	3	2
JEN-SAM	-	-	-	27	54	27	55	54	-	-	-	20	50	20	67	66
GAUS	9	9	9	7	18	7	14	13	49	52	41	14	28	15	33	32
MEYE	62	116	62	12	24	12	27	26	79	170	79	47	60	47	22	21
BOX3	6	35	6	4	4	4	5	4	7	35	7	5	5	5	6	5
POW-SI	63	94	63	17	37	17	45	44	-	-	-	10	41	10	23	22
KOW-OSB	36	52	36	33	43	33	48	47	41	57	41	44	64	44	46	45
BRO-DEN	-	-	-	5	5	5	10	9	-	-	-	5	5	5	8	7
OSB1	76	102	76	70	93	70	103	102	300	300	300	91	124	91	99	98
BIG-EX	31	47	31	21	38	21	31	30	28	45	28	17	34	17	36	35
OSB2	84	106	84	54	78	54	91	90	53	69	53	19	37	19	39	38
X-ROSE	34	72	34	36	53	36	41	40	77	130	77	11	40	11	42	41
X-POW-S	79	118	79	32	61	32	56	55	-	-	-	9	34	9	27	26
PEN1	1	2	1	7	7	7	13	12	1	2	1	5	5	5	6	5
VAR-DIM	202	258	202	1	2	1	3	2	-	-	-	18	19	18	37	36
TRIG	28	41	28	32	47	32	53	52	*	*	*	*	*	*	*	*
BR-A-LIN	-	-	-	17	35	17	34	33	1	0	0	1	0	0	0	0
DISC-BOUN	27	30	27	33	35	33	46	45	32	34	32	34	37	35	50	49
DISC-INT	25	25	25	25	25	25	31	30	27	27	27	26	26	26	33	32
BROY-TRI	60	78	60	64	98	64	48	47	27	69	27	12	12	12	30	29
BROY-BAN	88	119	88	68	109	68	88	87	26	76	26	11	11	11	26	25
LIN-FR	48	93	48	2	2	2	3	2	47	92	47	2	2	2	3	2
LIN-R1	-	-	-	12	12	12	21	20	1	9	2	11	11	11	12	11
LIN-R10	-	-	-	10	10	10	19	18	1	8	2	9	9	9	10	9
CHEB	44	66	44	42	66	42	53	52	*	*	*	*	*	*	*	*

Chapter 5

A Hybrid Local Search Method for Neural–Network Training

5.1 Introduction

In this section our approach to develop a local search method suitable for supervised training of feed-forward artificial neural networks, with one hidden layer and sigmoidal activation functions. The resulting Sum-of-Squares objective function is minimized using a hybrid technique that switches between the Gauss–Newton approach in the small residual case, and Newton's method (Section 2.4.1)in case where large residuals are detected. This is done in the spirit of Fletcher[43] where instead of Newton's method, a variable metric method (BFGS)(Section 2.3.3) was preferred in order to avoid the calculation of the Hessian matrix, which in the general case is both costly and cumbersome. In the special case that we consider here, the Hessian matrix can be expressed analytically and calculated efficiently by taking advantage of the properties of the sigmoidal activation function and its derivatives.

5.1.1 Problem Description

Artificial Neural Network (ANN) training is a subject of central interest due to the widespread involvement of ANNs in a variety of scientific as well as practical tasks, such as data fitting, modelling, classification, pattern recognition, solution of differential equations etc. The "back–propagation" technique, that has been widely used mainly due to the simplicity of its implementation, is far from being satisfactory. Its main shortcomings, i.e. the oscillatory behavior and the sensitivity to round–off that causes premature termination, were early recognized. As a consequence, several alternative approaches employing efficient and robust optimization methods have been tried out. Among the various optimization techniques, Newton's method is the one with the most desirable properties. However it requires the computation of the Hessian matrix which may be inaccurate and very expensive if performed numerically, or very complicated to be expressed analytically.

Hence due to this extra burden, Newton's method is not the preferred method in a host of practical applications. Training ANNs is an optimization problem, where the objective function can be cast as a sum of squared terms. This structure can be exploited to devise efficient approaches. In cases where the value of the objective function at the minimum is close to zero, an excellent approximation for the Hessian matrix exists that does not involve second derivatives. This is the well known "Gauss–Newton" approach. When however the value at the minimum is far from zero then the Hessian is not well approximated with severe consequences on the convergence of the approach. Taking the above into account, a hybrid method has been developed by Fletcher[43], which at run time detects, according to a simple criterion, the problem category, and switches appropriately to either the Gauss-Newton or to the BFGS Quasi–Newton method. Quasi–Newton methods do not use second order derivatives; instead they maintain at each iteration a positive definite approximation for the Hessian via an updating scheme, using gradient information only. Nowadays Quasi–Newton are considered the most succesful general purpose optimization methods and are being widely used. In this article we focus on the special problem of ANN training and we use a modified Newton instead of BFGS in the proposed framework of Fletcher[43]. We derive analytical closed-form expressions for the Hessian of this problem and present the sigmoidal properties that can be exploited to make the implementation efficient.

Let $N(x,p)$ denote an ANN with input vector x and weights p. In our case this will be a perceptron with one hidden layer with sigmoidal units and linear output activation, i.e.

$$N(x,p) = \sum_{i=1}^{h} p_{i(n+2)-(n+1)} \sigma \left(\sum_{k=1}^{n} p_{i(n+2)-(n+1)+k} x_k + p_{i(n+2)} \right) \tag{5.1}$$

where:

- $x_i,\ \forall i = 1, \cdots, n$ are the components of the input vector $x \in R^{(n)}$.
- $p_i,\ \forall i = 1, \cdots, h(n+2)$ are the components of the weight vector p.
- h, denotes the number of hidden units.
- $\sigma(z) \equiv (1 + exp(-z))^{-1}$ is the sigmoid used as activation.

The training of the ANN to existing data is performed by minimizing the following "Error function":

$$f(p) = \frac{1}{2} \sum_{K=1}^{M} r_K^2 \equiv \frac{1}{2} \sum_{K=1}^{M} [N(x_K, p) - y_K]^2 \tag{5.2}$$

Useful expressions are its gradient:

$$g \equiv \nabla_p f(p) = \sum_{K=1}^{M} r_K \nabla_p r_K = J^T r \tag{5.3}$$

and the Hessian:

$$G \equiv \nabla_p^2 f(p) = J^T J + \sum_{K=1}^{M} r_K \nabla_p^2 r_K \tag{5.4}$$

Where J is the Jacobian given by: $J_{K,j} = \frac{\partial r_K}{\partial p_j}$

5.1.2 Description of the algorithm

Nonlinear least-squares problems are among the most commonly occurring and important applications, such as neural network training. Let $r : \Re^n \to \Re^m$, with $m \geq n$ be a nonlinear mapping. The problem is:

Find a local minimum x^* of

$$f(x) = \frac{1}{2}\sum_{i=1}^{m}[r_i(x)]^2 = \frac{1}{2}r(x)^T r(x)$$

Assuming that $r_i(x)$ is twice differentiable function then the derivatives of f are given by:

$$\mathbf{g}(x) = \nabla f(x) = J(x)\mathbf{r}(x) \tag{5.5}$$

$$G(x) = \nabla^2 f(x) = J^T(x)J(x) + \sum_{i=1}^{m} r_i(x)\nabla^2 r_i(x) \tag{5.6}$$

where $J(x)$ is the Jacobian matrix with elements $J_{ij} = \frac{\partial r_i(x)}{x_j}$ and $G(x)$ is the Hessian matrix, with elements $G_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. Throughout this paper we will use the notation J_k, G_k, g_k and f_k for $J(x_k)$, $G(x_k)$, $g(x_k)$ and $f(x_k)$ respectively.

Many methods have been suggested for solving such problems. In this work we only consider Newton-like methods with line search. These methods have the form of the minimization algorithm 5.9.

Within this framework, different methods correspond to different choices for the matrix B_k. Two well known methods which have been extensively studied and constitute the basis for several others are the damped *modified Newton* method (see Section 2.4.1)for general nonlinear optimization and the damped *Gauss-Newton* method(see Section **??**) for nonlinear least squares problems.

Combining the methods

In comparing GN and Newton methods, the GN is generally preferred for zero residual problem (ZRP) that is when $\mathbf{r}(x^*) = 0$, whereas Newton-like methods are preferred for large residual problems (LRP) or when J_k looses rank.

Usually is not known beforehand whether a problem will turn out to have small or large residuals at the solution. It seems reasonable, therefore, to consider *hybrid algorithms*, which would behave like Gauss-Newton if the residuals turn out to be small (and take advantage of the cost savings associated with these methods) but switch to Newton like

Algorithm 5.9 Newton-like + Line Search Framework

Let x_k be the current estimate of x^*

S1. [*Test for convergence*] If the conditions for convergence are satisfied, the algorithm terminates with $x^* = x_k$ as solution

S2. [*Compute search direction*] Compute a non-zero vector p_k, by solving

$$B_k p_k = -g_k \tag{5.7}$$

where B_k is some positive definite approximation of the hessian $\nabla^2 f_k$. This property ensures that p_k is a descent direction.

S3. [*Compute step length*] Compute a positive scalar a_k, called step length, which satisfies the two conditions

$$f(x_k + a_k p_k) \leq f_k + \rho a_k g_k^T p_k \tag{5.8}$$

$$|g(x_k + a_k s_k)^T s_k| \leq -\sigma g_k^T s_k \tag{5.9}$$

$\rho \in (0, 1)$ and $\sigma \in (\rho, 1)$ known as Wolfe conditions.

S4. [*Update the estimate of the minimum*] Set $x_{k+1} \leftarrow x_k + a_k p_k, k \leftarrow k + 1$ and go back to step S1.

---

steps if the residuals at the solution are large (with the cost of approximating or computing second order derivatives).

We use Fletcher's criterion to switch between the GN approximation ($J_k^T J_k$) and a positive definite modification of the full Hessian.

In this way the method will asymptotically take Newton steps for a LRP and GN steps for ZRP.

Following Fletcher, the quantity

$$\lim_{k \to \infty} \frac{f_k - f_{k+1}}{f_k} = \begin{array}{ll} 0 & \text{for the LRP,} \\ 1 & \text{for the ZRP.} \end{array}$$

Therefore this quantity defines a straightforward criterion that can be used to switch the minimizing procedure from GN to modified Newton.

We replace the second step of the minimization algorithm with the following

---

S2. Compute search direction.

$$\text{Set } B_k = \begin{cases} \nabla^2 f_k & \text{if } f_{k-1} - f_k / f_{k-1} < \epsilon, \\ J_k^T J_k & \text{otherwise.} \end{cases}$$

Solve $B_k p_k = -g_k$ to get the search direction p_k

5.1.3 Hessian Calculation

Using the mapping $i = l(n+2) - (n+1) + m$ and $j = r(n+2) - (n+1) + s$ where $l, r = 1 \dots h$ and $m, s = 0 \dots n+1$ we can write $\frac{\partial^2 N(x_K, p)}{\partial p_i \partial p_j}$ in a form

$$\frac{\partial^2 N(x_K, p)}{\partial p_{l(n+2)-(n+1)+m} \partial p_{r(n+2)-(n+1)+s}} \tag{5.10}$$

For simplicity we denote $Y_j = \sum_{k=1}^{n} p_{j(n+2)-(n+1)+k} x_k + p_{j(n+2)}$. Using the above notation we can derive an analytic formula for the Hessian matrix of a feedforward artificial neural network $N(x, p)$. The resulting formula is displayed in Table 5.1.

Table 5.1: Analytic Hessian calculation

$l = r$	$m = 0$	$s = 0$	0
		$s = 1 \dots n$	$\sigma'(Y_j) x_s$
		$s = n+1$	$\sigma'(Y_j)$
	$m = 1 \dots n$	$s = 0$	$\sigma'(Y_j) x_m$
		$s = 1 \dots n$	$p_{l(n+2)-(n+1)} x_m x_s \sigma''(Y_j)$
		$s = n+1$	$p_{l(n+2)-(n+1)} x_m \sigma''(Y_j)$
	$m = n+1$	$s = 0$	$\sigma'(Y_j)$
		$s = 1 \dots n$	$p_{l(n+2)-(n+1)} x_s \sigma''(Y_j)$
		$s = n+1$	$p_{l(n+2)-(n+1)} \sigma''(Y_j)$
$l \neq r$	$m = 0 \dots n+1$	$s = 0 \dots n+1$	0

5.2 Experimental results

In order to compare the convergence speed of our hybrid method, to other well known algorithms we have contacted a series of experiments using the Merlin Optimization environment[114]. The Merlin testbed provides a set of powerful and robust minimization routines, that guarantee its effectiveness.

We have tested our method against five other minimization procedures namely the Quasi-Newton (Tolmin[127]), the Gauss-Newton with line search, the Hybrid BFGS–Gauss-Newton[1],the damped modified Newton and the conjugate gradients with Polack–Ribiere updates.

The strategy that we followed was to start the minimization in the neighborhood of a local minimum and calculate the iterations and function calls that each method performed in order to reach it. In this way we have a clear view of the convergence rate.

In the tables (5.2) and (5.3) we present the results for a training problem with input dimension $n = 2$ hidden nodes $h = 5$ and training data $M = 100$. This problem falls into the LRP category because the minimum reached is non-zero. Each table displays the results for two different minima of the same training problem.

[1] As it was presented in the original paper of Fletcher

Table 5.2: LRP: Minimum No 1

Method	**Iterations**	**Function calls**
Hybrid Newton	126	357
Hybrid BFGS	335	652
Newton	719	1000
Gauss-Newton	1000	3000
Tolmin	174	252
Conjugate Gradient	1480	6000
Minimum value		18.486

Table 5.3: LRP: Minimum No 2

Method	**Iterations**	**Function calls**
Hybrid Newton	20	64
Hybrid BFGS	33	100
Newton	35	57
Gauss-Newton	150	301
Tolmin	74	107
Conjugate Gradient	77	302
Minimum value		19.266

On the other hand, the results for a ZRP case are shown in the tables (5.4) and (5.5)This training problem has input dimension $n = 10$ hidden nodes $h = 10$ and training data $M = 100$. Again we present two results for different local minima.

Table 5.4: ZRP: Minimum No 1

Method	**Iterations**	**Function calls**	**Minimum reached**
Hybrid Newton	115	275	0
Hybrid BFGS	363	487	0
Newton	316	364	0
Gauss-Newton	257	296	0
Tolmin	513	697	0
Conjugate Gradient	2318	10000	1.0768
Minimum value			0

Finally we present comparative results using the well known More's test set of functions. We compared Merlin's BFGS and Levenberg-Marquardt implementations to the proposed hybrid approach. The number of gradient calls is equal to the number of iteration for each case.

Table 5.5: ZRP: Minimum No 2

Method	**Iterations**	**Function calls**	**Minimum reached**
Hybrid Newton	599	1000	0.0961
Hybrid BFGS	623	713	0.0101
Newton	759	1000	2.3282
Gauss-Newton	734	1000	0.4374
Tolmin	710	1000	0.0733
Conjugate Gradient	965	4000	1.2228
Minimum value			0

5.3 Conclusion

We have presented a novel hybrid method focused in Artificial Neural Network training. Our proposal combines the hybrid ideas of Fletcher[43] and an efficient way to compute analytically second order derivatives. Our preliminary results are promising, however more extensive experimentation should be contacted.

We are currently investigating online schemes for exact Hessian calculation, in order to increase the speed of our method. We also search for a better way to distinguish between LRP and SRP. Another extension is to calculate derivatives for alternative ANN architectures and use the proposed algorithm for their training process.

Test name	BFGS	LEVE	Hybrid Newton
	Func Eval/Iter	Func Eval/Iter	Func Eval/Iter(Gauss Steps)
ROSENBROCK	1.986 *10-16 80/12	0.000 10//3	3.958*10-16 16/2(1)
FREUDENSTEIN AND ROTH	7.183 *10-16 77/14	7.888 *10-31 28//9	48.984 64/10(2)
POWELL BADLY SCALED	Acc Stop	1.232 * 10ỏ32 202/59	1.102 * 10-8 23/4(2)
BROWN BADLY SCALED	7,17244E+11 40/2	2.549 * 10-29 50/17	1.139*10-14 1198/125(63)
BEALE	1.359*10-18 161/24	0.452 5545/1700	0.452 10016/1269(1)
JENNRICH AND SAMPSON	2020 38/1	259.58 74/24	2020 57/10(8)
HELICAL VALEY	3.024*10-34 183/24	1.271*10?57 30-8	5.933*10-38 110/13(10)
BARD	8.214*10-3 142/18	8.214*10-3 46/9	8.214*10-3 148/17(6)
GAUSSIAN	1.128*10-8 256/32	0.564 21/5	1.128*10-8 240/27(9)
MEYER	Acc Stop	87.94 28813/6708	8477691 10006/1005(6)
GULF	3.849*10?2 0 Iterations The gradient criterion is satisfied		
BOX 3-D	1.036*10-24 102/12	2.773*10-32 92/20	5.718*10-22 118/5(4)
POWELL SINGULAR	7.263*10?24 712/75	1.609*10-63 367/70	7.222*10-32 1471/73(62)
WOOD	1.187*10-17 587/62	0.000 36/7	1.187*10-17 169/18(13)
KOWALIK AND OSBORNE	3.075*10-4 665/68	1.027*10?3 569/110	1.027*10?3 10001/833(5)
BROWN AND DENNIS	85822.22 254/21	85822.22 358/69	85822.22 207/18(5)
OSBORNE 1	1.106 50/4	1.106 51/7	1.106 60/5(1)
BIGGS EXP6	0.306 228/17	0.180 16730/2307	Acc Stop
OSBORNE 2	1.790 549/17	1.790 171/14	1.790 466/7(1)
WATSON	2.829*10-13 7963/184	2.836*10-3 8210/39	868908 424/6(2)
EXTENDED ROSENBROCK	1.998*10-15 8976/406	0.000 46/5	1.987*10-15 231/10(9)
EXTENDED POWELL SINGULAR	3.705*10-16 3739/147	3.112*10-68 952/72	7.928*10-32 4166/87(70)
PENALTY I	2.249*10-5 1886/195	2.249*10-5 179/32	2.249*10-5 918/90(10)
PENALTY II	9.376*10-6 12825/1351	9.376*10-6 160/29	Acc Stop
VARIABLY DIMENSIONED	2.674*10-30 715/33	0.000 155/14	0.000 85/3(2)
TRIGONOMETRIC	4.224*10-5 1749/81	8.788*10-4 277/23	2.795*10-5 1120/48(17)
BROWN ALMOST LINEAR	1.316*1013 50/0/1	1.000 179/16	9.478*10-30 682/25(21)
DISCRETE BOUNDARY PR	9.358*10-21 751/35	2.503*10-33 90/9	8.962*10-21 203/9(7)
DISCRETE INTEGRAL EQ	1.116*10-22 574/27	3.229*10-33 90/9	1.034*10-22 222/10(8)
BROYDEN TRIDIAGONAL	0.974 1721/78	1.521 583/51	1.349 545/23(4)
BROYDEN BANDED	2.680 2122/96	12.593 455/40	3.469*10-18 382/18(11)
LINEAR, FULL RANK	9.999 49/2	10.000 58/5	9.999 95/3(1)
LINEAR, RANK 1	4.634 105/4	4.633 202/18	4.634 158/4(1)
LINEAR, RANK1, ZERO	6.135 120/5	6.135 155/14	6.135 53/2(1)
CHEBYQUAD	Acc Stop	3.286*1014 1000004/88833	68370621 344/14(12)
LARGE EXTENDED ROSENBROCK	2.595 1357498/6704	0.000 406/5	1.999*10-14 10121/47(33)
EXTENDED BEALE	5.935*10+16 230/0/1	22.600 298869/0/2953	27.658 1000017/4895(23)
EXTENDED WOOD	2.962*10-16 356326/1768	0.000 1213/12	2.916*10-16 4059/17(12)

Table 5.6: Comparative results for the More's test set

Chapter 6

Parallelizing derivatives

6.1 Summary

We present a software library for numerically estimating first and second order partial derivatives of a function by finite differencing. Various truncation schemes are offered resulting in corresponding formulas that are accurate to order $O(h)$, $O(h^2)$, and $O(h^4)$, h being the differencing step. The derivatives are calculated via forward, backward and central differences. Care has been taken that only feasible points are used in the case where bound constraints are imposed on the variables. The Hessian may be approximated either from function or from gradient values. There are three versions of the software: a sequential version, an OpenMP version for shared memory architectures and an MPI version for distributed systems (clusters). The parallel versions exploit the multiprocessing capability offered by computer clusters, as well as modern multicore systems and due to the independent character of the derivative computation, the speed up scales almost linearly with the number of available processors/cores.

6.2 Introduction

Estimating derivatives is a common subtask in many applications. For example the majority of optimization methods employ the gradient and/or the Hessian of the objective function [111, 78, 53, 35, 40], while for the solution of nonlinear systems the Jacobian matrix [35, 40] is required.

There are several methods for calculating derivatives. Methods that manipulate symbolically analytic expressions and provide the derivative in closed form [162, 102], are exact but of limited applicability. Automatic differentiation (AD) [62] is a promising alternative that has been rather recently developed. Given the source code that implements a function, the AD software creates the code for its derivative exploiting repeated application of the chain rule. Although this is a powerful technique, the complexity of the process is considerable and sometimes the gains are marginal.

In a growing number of real world applications in science and engineering, the underlying functions are represented by large and complicated computer codes and the user may find it difficult or almost impossible to follow the original program (if it is available in source form) and develop the corresponding code for the derivative. An alternative is offered by finite differencing, where the derivatives are approximated from function values at suitably chosen points. The corresponding formulae may be derived in a number of ways, and a detailed classification is given in [11]. Recently Khan and Ohba [79] reported formulas based on Taylor expansion, suitable for highly oscillating functions and Li [89] extended this method by employing equally and unequally spaced points to estimate derivatives of arbitrary order.

Derivative estimation via finite differencing is simple but computationally expensive, since it requires a number of function evaluations. Moreover there are several applications where the time for a single function call is substantial. For example, the determination of stable molecular conformations via "molecular mechanics" [22, 129], ab–initio quantum mechanical structure calculations [129, 76], construction of potentials for the atomistic simulation of materials [165, 115], the construction of nuclear forces [84], phase–shift analysis from nucleon–nucleon scattering data [132], the solution of partial differential equations via Galerkin type of methods [81, 82, 83] and all cases where the function's value is a result of a simulation. In such cases parallelism could play a key role in accelerating the process. Assuming that we have an ample number of available processors, the cost for the derivative calculation may become almost equal to that of a single function call if parallel processing is employed.

In the literature there exist several software packages for estimating derivatives numerically. In the GSL library [50] the authors provide a five-point rule for central differences and four-point rules for forward/backward differences using an adaptive scheme to achieve the desired accuracy. NAG library [121] provides subroutine `D04AAF` that calculates derivatives up to 14^{th} order for a univariate function using an extended Neville's algorithm [96, 95]. In the book of Mathews [101] three subroutines are offered for differentiating univariate functions, one of which takes in account upper and lower variable bounds. The `numDeriv` package [51] differentiates multivariate functions using Richardson extrapolation and calculates the Jacobian and Hessian matrices. Package `DIFF` [112] differentiates univariate functions up to the third order using Neville's process. Mathematica [162] provides a command (`ND`) for differentiation up to any order, using Richardson's extrapolation to the limit. Package `LNIDIF` [157] calculates first and second order derivatives having non-uniformly spaced points, out to three dimensions. IMSL library provides subroutine `DERIV` [75] for calculating up to third order derivatives of univariate functions. From the above only [51] supports multivariate functions, while [157] is limited to three dimensional functions. Note that the implementation of these packages is sequential, hence they cannot exploit the advantage offered by the architecture of distributed or parallel systems. This capability is important and that is why even automatic differentiation packages, such as [21], have followed the parallel implementation path.

The software described in this article, supports multivariate functions, respects variable bounds, offers several prescribed accuracy levels, and is implemented using state of the art parallel techniques in the framework of MPI [48] and OpenMP [30] platforms. The rest of the paper is organized as follows. In Section 2 we provide derivative formulas. In Section 3 we briefly sketch the parallelization strategy followed, and in Section 4 we describe the interface of the subroutines included in the library. Installation instructions are given in Section 5. Finally in Section 6 we present and analyze numerical experiments that concern both the accuracy and the efficiency of the software. In the software's distribution extensive test runs are included.

6.3 Derivative formulae

In the following, we present derivative formulae using forward differences (FD), backward differences (BD), and central differences (CD), for several levels of accuracy. The formulae for FD and BD are common. FD use positive while BD use negative differencing step. The reason for this variety is to handle the bound constraints case, i.e. when $x_i \in [a_i, b_i]$. Consider the case where the derivative at the bound $x_i = a_i$ is desired. Then the proper formula to use is from the forward difference class, since otherwise infeasible $x-$values will be used, which may lead to numerical errors. (Imagine for example, a quantity under the square root sign that becomes negative when x is infeasible). The software takes into account bound constraints, in the sense that only feasible points are used to evaluate the derivatives, and given the level of the desired accuracy, the proper formula is automatically employed. In the case where there are no bounds or when the bounds are not violated, the default selection is the FD formula for $O(h)$ and the CD formulae for $O(h^2)$ and $O(h^4)$.

6.3.1 First order derivatives

The formulae of this section are used to evaluate the gradient of a scalar function and the Jacobian of a vector function. Each formula uses a different stepsize that is determined by approximately minimizing the total (truncation and roundoff) error [53]. In what follows we denote by η the relative error in the calculation of f, that defaults to the machine precision in absence of such information.

1. Accurate to order $O(h)$
   $h = \pm\sqrt{\eta}\max\{1, |x|\}$
   FD/BD formula:
   $$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h} \tag{6.1}$$
2. Accurate to order $O(h^2)$
   $h = \pm\eta^{1/3}\max\{1, |x|\}$

(a) CD formula:

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x-h)}{2h} \tag{6.2}$$

(b) FD/BD formula:

$$\frac{df(x)}{dx} \approx \frac{4f(x+h) - 3f(x) - f(x+2h)}{2h} \tag{6.3}$$

3. Accurate to order $O(h^4)$
   $h = \pm\eta^{1/5} \max\{1, |x|\}$

   (a) CD formula:

$$\frac{df(x)}{dx} \approx \frac{1}{3}\left(4\frac{f(x+h) - f(x-h)}{2h} - \frac{f(x+2h) - f(x-2h)}{4h}\right) \tag{6.4}$$

   (b) FD/BD formula:

$$\begin{aligned}\frac{df(x)}{dx} \approx \frac{1}{21}\Bigg(64\frac{f(x+h) - f(x)}{h} - 56\frac{f(x+2h) - f(x)}{2h} \\ +14\frac{f(x+4h) - f(x)}{4h} - \frac{f(x+8h) - f(x)}{8h}\Bigg)\end{aligned} \tag{6.5}$$

The gradient of a function $f(x),\ x \in R^N$ is calculated as:

$$\frac{\partial f(x)}{\partial x_i} = \frac{df(x + \alpha e_i)}{d\alpha}|_{\alpha=0},\ \forall i = 1, 2, \ldots, N$$

where e_i is the unit vector in the i^{th} direction. The gradient is coded using all formulae (6.1) to (6.5).

The Jacobian of a vector function $F^T(x) = (f_1(x), f_2(x), \ldots, f_M(x))$ is given by:

$$J_{ti} = \frac{\partial f_t(x)}{\partial x_i} = \frac{df_t(x + \alpha e_i)}{d\alpha}|_{\alpha=0}$$

with $t = 1, 2, \ldots, M$ and $i = 1, 2, \ldots, N$. Formulae up to $O(h^2)$ are implemented (only formulae (6.1), (6.2) and (6.3)).

6.3.2 Second order derivatives

The library offers two options for the estimation of the Hessian elements. If the gradient $g_i(x) = \frac{\partial f(x)}{\partial x_i}$ is available, the Hessian may be estimated by differencing the gradient, otherwise it is estimated using function values.

Using the gradient

1. Accurate to order $O(h_i) + O(h_j)$
   $h_k = \pm\sqrt{\eta}\max\{1, |x_k|\}$ for $k = i, j$
   FD/BD formula:

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{2}\left(\frac{g_i(x + h_j e_j) - g_i(x)}{h_j} + \frac{g_j(x + h_i e_i) - g_j(x)}{h_i}\right) \tag{6.6}$$

2. Accurate to order $O(h_i^2) + O(h_j^2)$
   $h_k = \pm\eta^{1/3}\max\{1, |x_k|\}$ for $k = i, j$

   (a) FD/BD formula:

$$\begin{aligned}\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{2}\Bigg(&\frac{4g_i(x + h_j e_j) - g_i(x + 2h_j e_j) - 3g_i(x)}{2h_j} + \\ &\frac{4g_j(x + h_i e_i) - g_j(x + 2h_i e_i) - 3g_j(x)}{2h_i}\Bigg)\end{aligned} \tag{6.7}$$

   (b) CD formula:

$$\begin{aligned}\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{2}\Bigg(&\frac{g_i(x + h_j e_j) - g_i(x - h_j e_j)}{2h_j} + \\ &\frac{g_j(x + h_i e_i) - g_j(x - h_i e_i)}{2h_i}\Bigg)\end{aligned} \tag{6.8}$$

Using function values

1. Accurate to order $O(h_i) + O(h_j)$
   $h_k = \pm\eta^{1/3}\max\{1, |x_k|\}$ for $k = i, j$
   FD/BD formula:

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{h_i h_j}\left(f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)\right) \tag{6.9}$$

2. Accurate to order $O(h_i h_j)$
   $h_k = \pm\eta^{1/4}\max\{1, |x_k|\}$ for $k = i, j$

   (a) FD/BD formula:
   Off–diagonal elements:

$$\begin{aligned}\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{4h_i h_j}\Big(&9f(x) + 16f(x + h_i e_i + h_j e_j) \\ &+ f(x + 2h_i e_i + 2h_j e_j) \\ &- 4f(x + h_i e_i + 2h_j e_j) - 4f(x + 2h_i e_i + h_j e_j) \\ &- 12f(x + h_i e_i) - 12f(x + h_j e_j) \\ &+ 3f(x + 2h_i e_i) + 3f(x + 2h_j e_j)\Big)\end{aligned} \tag{6.10}$$

Diagonal elements:

$$\frac{\partial^2 f(x)}{\partial x_i^2} \approx \frac{1}{h_i^2}\left(2f(x) - f(x+3h_ie_i) + 4f(x+2h_ie_i) - 5f(x+h_ie_i)\right) \quad (6.11)$$

(b) CD formula:
Off–diagonal elements:

$$\begin{aligned}\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{4h_ih_j}\big(&f(x+h_ie_i+h_je_j) + f(x-h_ie_i-h_je_j)\\ &-f(x+h_ie_i-h_je_j) - f(x-h_ie_i+h_je_j)\big)\end{aligned} \quad (6.12)$$

Diagonal elements:

$$\frac{\partial^2 f(x)}{\partial x_i^2} \approx \frac{1}{h_i^2}\left(f(x+h_ie_i) + f(x-h_ie_i) - 2f(x)\right) \quad (6.13)$$

(c) Mixed CD and FD(BD) formula:

$$\begin{aligned}\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{1}{4h_ih_j}\big(&4\left(f(x+h_ie_i+h_je_j) - f(x+h_ie_i-h_je_j)\right)\\ &-3\left(f(x+h_je_j) - f(x-h_je_j)\right)\\ &-\left(f(x+2h_ie_i+h_je_j) - f(x+2h_ie_i-h_je_j)\right)\big)\end{aligned} \quad (6.14)$$

6.4 Parallelization strategy

For the shared-memory parallelization of the Numerical Differentiation Library (NDL) we have used OpenMP, the standard programming model for a wide range of parallel platforms including small-scale SMPs and emerging multi-core processors. OpenMP defines a portable programming interface based on directives, i.e. annotations that enclose loops and sections of code. In addition, it provides a means of seamless parallelization of NDL, as it allows the construction of a parallel program as a natural extension of its sequential counterpart. To achieve optimal load balance and speedup in NDL, we exploit its inherent nested parallelism [147], that results from the multiple function evaluations performed at each coordinate direction. Nested parallelism is a major feature of OpenMP that allows multiple levels of parallelism to be active simultaneously. Nowadays, several research and commercial OpenMP compilers support more than one level of parallelism.

The parallelization of the software library on multiprocessor clusters has been based on the master-worker programming paradigm, a fundamental approach for parallel and distributed computing. In NDL, task parallelism is possible due to the independent function evaluations assigned by the master to the workers. For each function evaluation, the master provides the input vector x to the worker and receives the computed function value $f(x)$. The parallel version of NDL has been coded using the LWRPC library [65], a

runtime environment which is part of the software and provides a lightweight framework for executing task parallelism on top of MPI. LWRPC is a flexible and portable runtime library that implements a two-level thread model (where user-level threads run on top of kernel-level threads [23]) on clusters of multiprocessors, trying to exploit the shared-memory hardware whenever this is available.

It provides transparent data movement and load balancing and allows for static and dynamic scheduling of tasks. Furthermore, it supports multiple levels of parallelism and enables the same code to run efficiently on both distributed and shared memory multiprocessors.

In LWRPC, a task is represented with a data structure, called *work descriptor*. Tasks are distributed to the available nodes and eventually executed on top of user-level threads. The same approach has also been followed for the master which is the primary task. An MPI process runs on each cluster node and utilizes one or more kernel threads that execute these tasks. Moreover, task submission and management is performed completely asynchronously by means of a special per-node server thread. There are ready queues where tasks are submitted for execution. The submission of a work descriptor to a local queue is always performed through hardware shared memory, otherwise appropriate messages are sent to the server thread of the remote node. Each work descriptor (i.e. task) is associated with an owner node. If a task finishes on its owner node, its parent is notified directly through shared memory. Otherwise, a message is sent to the owner node and this notification is performed by the server thread of that node.

When the application is executed on shared memory machines, the runtime library, and accordingly the application, operates exclusively through the available hardware. However, whenever a task is inserted on a remote node, its data has to be sent explicitly. In this case, we associate each work descriptor with the arguments (data) of its corresponding function, similarly to the Remote Procedure Call protocol [145]. For each argument, the user specifies its MPI data type and number of elements, an intent attribute, and optionally a reduction operator. These MPI-specific details are the only references made to message passing programming at the user level. Note that, the explicit data movement is performed transparently to the user.

The parallel routines that NDL exports to MPI programs are designed to be called by all MPI processes that participate in the program execution, similarly to the MPI collective communication routines. This design adheres to the SPMD (Single Program Multiple Data) execution model that MPI supports by default. When an NDL parallel routine is invoked, the execution model switches to master-worker and the thread of the process with rank 0 becomes responsible for distributing the tasks to the workers and gathering the results. In NDL, each task corresponds to a function evaluation at a given point. Before returning from the library call to the user program, the execution model switches back to SPMD and the results are broadcast from the master to the rest of the MPI processes. This is schematically illustrated in Fig. 6.1.

In the case of second order derivatives we use a nested parallelization model that is

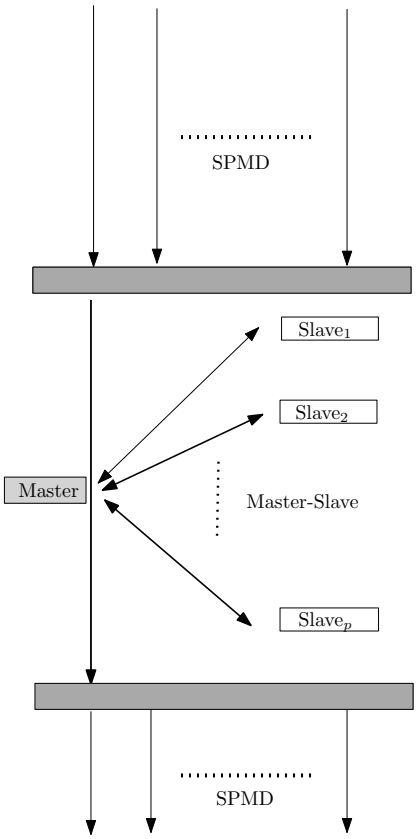


Figure 6.1: Library's Programming Model

provided by the LWRPC library. By nested we mean that each element of the Hessian is calculated, as a first level job, by a single worker. That leads to $(N^2 + N)/2$ first level jobs. Every first level job is responsible to perform function/gradient evaluations, second level jobs, according to the desired accuracy and the bounds. By inspecting the formulae of the previous section, it is readily deduced that the required number of second level jobs ranges between two and nine.

6.5 User interface

We have implemented subroutines for calculating gradients, Hessians and Jacobians. In all cases we provide serial, OpenMP–parallel and MPI–parallel subroutines. Every subroutine has a standard and an advanced interface. The advanced interface allows the user to specify bounds on the variables and an estimate for the relative accuracy of the function evaluation. It also returns the number of function calls, error codes and optionally issues verbose output.

6.5.1 Naming conventions

We use the following naming convention for the subroutines:

```
xNDLyz ( arguments )
```

where `x`, denotes the type of parallelization and can be:

empty: for the serial version

O: for the OpenMP–parallel version

P: for the MPI–parallel version

`y` denotes the order of accuracy and can be:

G: for the gradient

H: for the Hessian

J: for the Jacobian

`z` denotes the interface type correspondingly and can be:

empty: for the standard interface

A: for the advanced interface

where by *empty* is meant that the symbol is missing.

For example subroutine `PNDLGA` stands for the MPI–parallel code (`P`) for the gradient calculation (`G`) using the advanced interface (`A`). Also, `NDLJ` is the serial subroutine to calculate the Jacobian matrix using the standard interface.

6.5.2 Common arguments

The provided subroutines share a number of common arguments which are described bellow.

`X`	(input)	Array containing the point at which the calculation is desired.
`N`	(input)	The dimensionality of the function.
`XL`	(input)	Array containing the lower bounds on the variables.
`XU`	(input)	Array containing the upper bounds on the variables.
`FEPS`	(input)	The user's estimate for the relative precision of the function evaluation. If FEPS=0 it is reset to the machine's precision.
`IORD`	(input)	Requested order of accuracy. Possible values for gradients are 1, 2, 4 and for Hessians and Jacobians 1, 2.
`IPRINT`	(input)	Controls the amount of printout from the routine. Note that all output appears on the standard output device. Possible values are:

0 No printout at all.

1 Fatal error messages are printed.

2 Warning messages are printed.

3 Detailed information is printed (the formula that was used, differentiation steps and the resulting derivatives vector).

`NOC` (output) Number of calls to the function being differenciated.

`IERR` (output) Error indicator. Possible values are:

0 No errors at all.

1 Improper IORD value.

2 The supplied N is less than 1.

3 Some of the supplied upper bounds (XU) are less than the corresponding lower bounds (XL).

4 Some of the supplied variables are out of bounds.

5 The value of FEPS is incorrect (less than 0 or greater than 1).

6 The supplied IPRINT is incorrect.

7 N exceeds the maximum allowed value (MAXN). MAXN must be increased appropriately and the library must be reconfigured.

8 There is not enough internal storage. MAXN must be increased to match the number of squared terms (M) and the library must be reconfigured.

9 The number of squared terms (M) is less than 1.

6.5.3 Gradient calculation

Given a multidimensional function (F), these routines return the gradient vector (G) by applying a numerical differentiation formula according to the desired order of accuracy (IORD). The user provided function F must be declared as:

```
FUNCTION F ( X, N )
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION X(N)
```

Standard interface:

```
SUBROUTINE NDLG (F,X,N,IORD,G)
```

Advanced interface:

```
SUBROUTINE NDLGA (F,X,N,XL,XU,FEPS,IORD,IPRINT,G,NOC,IERR)
```

`F` (input) The function to be differentiated.

`G` (output) Array containing the resulting gradient vector

6.5.4 Jacobian calculation

Given a multidimensional function that is written as a sum of squared terms (residuals):

$$F(x) = \sum_{i=1}^{M} f_i(x)^2$$

these routines return the Jacobian matrix (FJ) by applying a numerical differentiation formula according to the desired order of accuracy (IORD). The user provided subroutine must declared as:

```
SUBROUTINE RSD ( X, N, M, F )
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION X(N), F(M)
```

Standard interface:

```
SUBROUTINE NDLJ (RSD,X,N,M,IORD,FJ,LD)
```

Advanced interface:

```
SUBROUTINE NDLJA (RSD,X,N,M,XL,XU,FEPS,IORD,IPRINT,FJ,LD, NOC,IERR)
```

`RSD` (input) A subroutine that returns the residuals.

`M` (input) The number of squared terms.

`LD` (input) Leading dimension of matrix FJ.

`FJ` (output) The Jacobian matrix. $J_{ij}(x) = \dfrac{\partial f_i(x)}{\partial x_j}$

6.5.5 Hessian calculation

Using gradients

Given a routine (GRD) that evaluates analytically the first partial derivatives of a function, these routines return the Hessian matrix (HES) by applying a numerical differentiation formula according to the desired order of accuracy (IORD). The user provided subroutine GRD must be declared as:

```
SUBROUTINE GRD ( X, N, G )
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION X(N), G(N)
```

Standard interface:
`SUBROUTINE NDLHG (GRD,X,N,IORD,HES,LD)`
Advanced interface:
`SUBROUTINE NDLHGA (GRD,X,N,XL,XU,FEPS,IORD,IPRINT,HES,LD,NOC,IERR)`

`GRD`	(input)	A subroutine that returns the gradient vector (G), given the values of the variables (X).
`LD`	(input)	Leading dimension of matrix HES.
`HES`	(output)	Array containing the resulting Hessian matrix. Note that only the lower triangular part (plus the diagonal elements) is returned.

Using function values

Given a multidimensional function (F), these routines return the Hessian matrix (HES) by applying a numerical differentiation formula according to the desired order of accuracy (IORD). The user provided function F must be declared as:

```
FUNCTION F ( X, N )
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION X(N)
```

Standard interface:
`SUBROUTINE NDLHF (F,X,N,IORD,HES,LD)`
Advanced interface:
`SUBROUTINE NDLHFA (F,X,N,XL,XU,FEPS,IORD,IPRINT,HES,LD,NOC,IERR)`

`F`	(input)	The function to be differentiated.
`LD`	(input)	Leading dimension of matrix HES.
`HES`	(output)	Array containing the resulting Hessian matrix. Note that only the lower triangular part (plus the diagonal elements) is returned.

6.6 Installation instructions and sample program

In this section we describe in brief how to configure and install the software and we provide a basic sample program for the MPI–parallel case.

6.6.1 Installation instructions

The software is distributed as a `tar.gz` file and can be uncompressed and extracted by issuing

`gunzip ndl-1.0.tar.gz tar -xvf ndl-1.0.tar.gz`

A directory called `ndl-1.0` will be created with three subdirectories `serial`, `openmp` and `mpi` containing the serial, OpenMP–parallel and MPI–parallel distributions respectively. Any of the three distributions can be installed by entering the corresponding directory and executing the following steps:

1. Configure the package:

   `./configure`

   In the case of OpenMP one must specify the Fortran compiler with the appropriate options. For example:

   `./configure F77=gfortran FFLAGS=-fopenmp ./configure F77=ifort FFLAGS=-openmp`

   Other configuration choices include the specification of the installation directory:

   `./configure --prefix=<install-dir>`

   the definition of the desired C and Fortran compilers:

   `./configure CC=<path-to-mpicc> F77=<path-to-mpif77>`

   and the definition of the maximum problem dimension:

   `./configure --with-maxn=<num>`

   Further help for the configuration parameters can be obtained by entering:

   `./configure --help`

2. Build the library:

   `make`

   For each distribution a library file will be created, `libndl.a` for the serial version, `libpndl.a` for the OpenMP–parallel version and `libondl.a` for the MPI–parallel version. This step also compiles the provided test run code.

3. Install the library:

   `make install`

   By default the NDL library files will be placed in `/usr/local/lib`.

6.6.2 Sample program

We present a sample program (file `osample.f`) that uses the OpenMP version of our library in order to calculate the gradient of the function $f(x_1, x_2) = x_1 \cos(x_2) + x_2 \cos(x_1)$ using order $O(h^2)$.

```
        program otest
        implicit double precision (a-h, o-z)
        parameter (n=2)
        dimension x(n), g(n)
        external f
C
        x(1) = 1.0d0
        x(2) = 1.1d0
        iord = 2
        call ondlg ( f, x, n, iord, g )
        print *, 'point ', (x(i), i=1, n)
        print *, 'gradient ', (g(i), i=1, n)
        end
c-------------------------------------------------------------
        function f(x,n)
        implicit double precision (a-h, o-z)
        dimension x(n)
        f = x(1)*cos(x(2))+x(2)*cos(x(1))
        end
```

The above sample program can be compiled and executed using:

```
$ gfortran -o osample osample.f -londl $ export OMP_NUM_THREADS=2
$ ./osample
```

Command `export OMP_NUM_THREADS=<num>` sets the number of worker threads in the OpenMP runtime library [30]. The MPI version of the above a sample program is presented bellow (file `psample.f`).

```
        program ptest
        implicit double precision (a-h, o-z)
        parameter (n=2)
        dimension x(n), g(n)
        external f
        include 'mpif.h'
        call mpi_init(mpierror)
c   ...
```

```
        x(1) = 1.0d0
        x(2) = 1.1d0
        iord = 2
        call pndlg ( f, x, n, iord, g )
c   ...
        call mpi_comm_rank (mpi_comm_world,irank,ierr)
        if (irank.eq.0) then
           print *, 'point ', (x(i), i=1, n)
           print *, 'gradient ', (g(i), i=1, n)
        endif
c   ...
        call mpi_finalize(mpierror)
        end
c-------------------------------------------------------------
        function f(x,n)
        implicit double precision (a-h, o-z)
        dimension x(n)
        f = x(1)*cos(x(2))+x(2)*cos(x(1))
        end
```

The above sample program can be compiled and executed using:

```
$ mpif77 -o psample psample.f -lpndl $ mpirun -n 2  ./psample
```

6.7 Performance results

In Table 6.1 we report the relative error defined as:

$$err^{(k)} = \frac{|f^{(k)}(x) - f_{fd}^{(k)}|}{\max(1, |f^{(k)}(x)|)}$$

where $f^{(k)}(x)$ is the exact k^{th} order derivative and $f_{fd}^{(k)}$ is a finite difference approximation to it. We used the five test functions listed in the first column. The relative error has been calculated at eleven equidistant points in $[-1, 1]$. In columns 2–4 we report the maximum (among these points) $err^{(1)}$, for the $O(h), O(h^2)$ and $O(h^4)$ formulae. Correspondingly in columns 5–6 we report the maximum $err^{(2)}$ for the $O(h)$ and $O(h^2)$ formulae.

In order to measure the performance of the parallel-NDL implementation we have conducted extensive tests for both OpenMP and MPI–parallel versions. We have performed two sets of experiments:

E1: We used a 500-dimensional test function (without imposing bounds on the variables) and we calculated the gradient with $O(h^4)$ precision. That leads to a total of 2000 function evaluations. We have arranged for function evaluation time to be 1 ms, 10 ms and 100 ms respectively, via appropriate artificial delays.

Table 6.1: Relative errors in several example functions.

	First order			Second order	
Function	$O(h)$	$O(h^2)$	$O(h^4)$	$O(h)$	$O(h^2)$
$\sin x$	1.5E-08	2.0E-11	1.1E-13	1.0E-05	5.3E-09
e^x	2.0E-08	2.4E-11	1.9E-13	1.4E-05	5.4E-09
$x^2 \sin x$	4.7E-08	1.0E-10	6.7E-13	6.0E-05	2.5E-08
$xe^{-2x} + \sin 3x$	1.5E-07	2.2E-10	5.1E-12	2.2E-04	1.2E-07
$x^7 + 2x^5 - 5x$	5.3E-07	2.7E-09	6.2E-11	8.2E-05	1.4E-07

E2: We used a 20-dimensional test function (without imposing bounds on the variables) and we calculated the gradient with $O(h^4)$ precision. In this setting the total number of function evaluations is 80. The computational cost for each function call was again set to be 1 ms, 10 ms and 100 ms respectively.

We measure the speedup s defined as $s = T_1/T_p$, where T_1 is the time required for execution on one processor and T_p is the real time required when running on p processors.

6.7.1 MPI–parallel

Our experiments are performed on a 200 node Hewlett-Packard XC cluster system. Each node has 2 AMD Opteron–248 processors and 4GB main memory, while the nodes are interconnected with Gigabit Ethernet.

In Fig. 6.2 we present the results from experiment *E1*. The solid line represents the ideal speedup. For the 1 ms and 10 ms test functions where the communication times are comparable to execution times, the speedup is reduced to approximately 35% and 70% away from the ideal, while the speedup for the 100 ms function almost coincides with the ideal as it was expected. The results for experiment *E2* are presented in Fig. 6.3, where similar behavior is observed. The steps in the curves for the 10 ms and 100 ms functions are due to the way the required 80 function evaluations are distributed to the available processors. If the number of processors p divides exactly the number of tasks n_t, then all p processors are employed for $\frac{n_t}{p}$ cycles. Otherwise p processors are fully utilized for $[\frac{n_t}{p}]$ cycles, and one more cycle is needed employing $n_t \ mod \ p$ processors.

6.7.2 OpenMP–parallel

The same experiments were conducted on a shared-memory multiprocessor system equipped with 4 Dual-Core 3.0GHz Intel Xeon processors and 4GB main memory, running 64-bit Debian Linux. In Fig. 6.4 we present the results only for the 1 ms test function since the others (10 ms, 100 ms test functions) almost coincide with the ideal speedup. We notice a 19% reduction from the ideal speedup when $N = 20$.

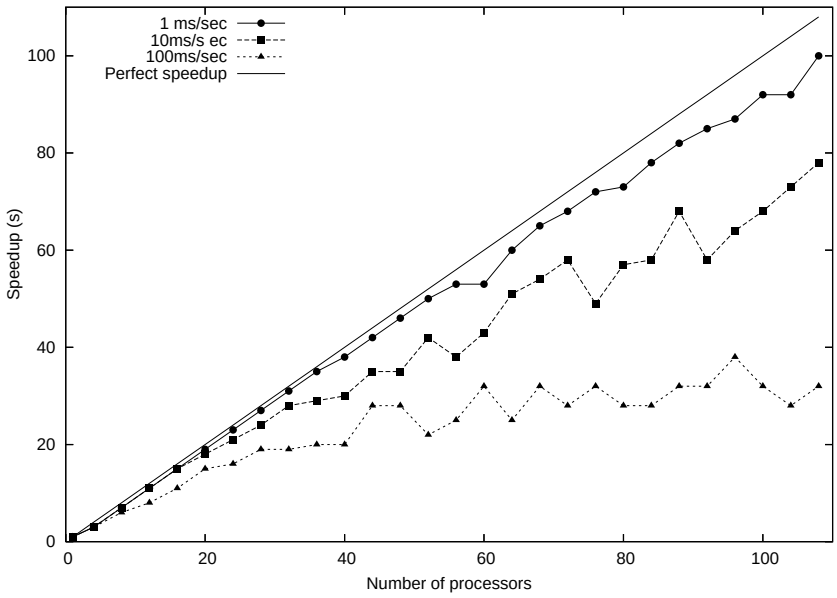


Figure 6.2: Speedup for experiment E1 ($N = 500$)

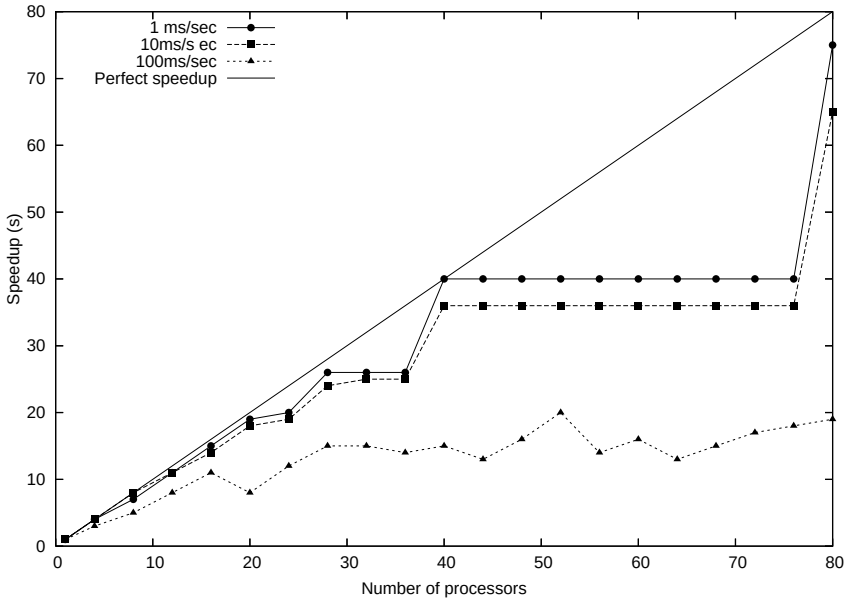


Figure 6.3: Speedup for experiment E2 ($N = 20$)

6.8 Test run description

Extensive test runs for the serial and parallel versions of the library were performed and are available with the distribution. The user is advised to repeat these runs in order to validate the installation. The relevant files are located in a subdirectory named `test`.

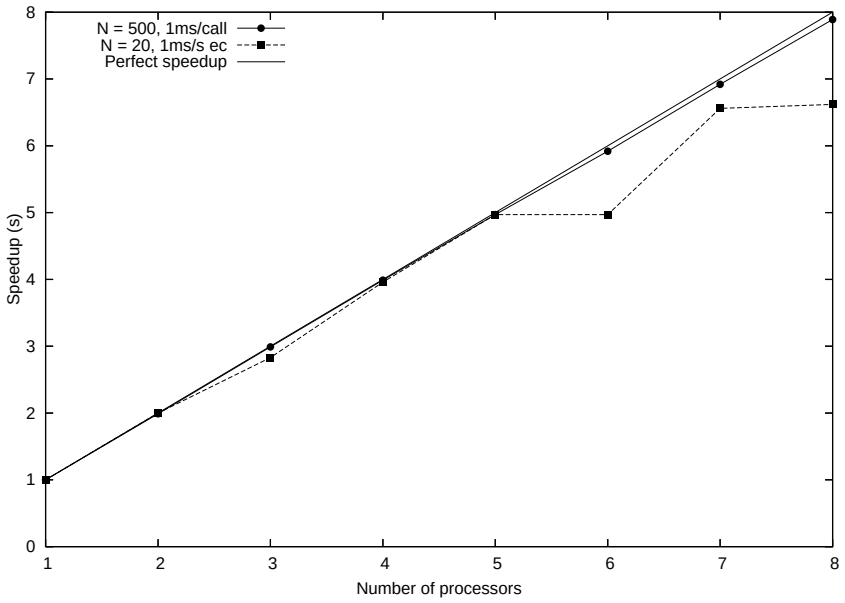


Figure 6.4: OpenMP implementation speedup

Acknowledgments

The authors acknowledge the computational resources generously provided by the Center for Scientific Simulation of the University of Ioannina.

Part III

Global Optimization

Chapter 7

Survey on Stochastic Global Optimization

The global Optimization problem (P) can be stated as:

$$\begin{aligned} &\min f(x) \\ &\text{subject to } x \in S \end{aligned}$$

where f is a continuous function on S and $S \subseteq R^n$ is a compact body. Some of the methods we will describe require additional assumptions on the objective function f or the feasible region S;we will note them wherever necessary. We will not consider very specialized subclasses of problems. However, under these weak conditions we know that the optimal solution value

$$f^* \equiv \max_{x \in S} f(x) \tag{7.1}$$

exists and is attained, i.e. the set

$$S^* \equiv \{x \in S : f(x) = f^*\} \tag{7.2}$$

is nonempty.

It is well-known that the global optimization problem (P) is inherently unsolvable in a finite number of steps. This can be verified as follows[]. For any continuously differentiable function f, any point x and any neighborhood o of x, there exists a function f' such that $f + f'$ is continuously differentiable, $f + f'$ equals f for all points outside B, and the global optimum of $f + f'$ is x ($f + f'$ is an indentation of f). Thus, for any point x, one cannot verify with certainty that it is not the global optimum without evaluating the function in at least one point in every neighborhood o of x. As o can be chosen arbitrarily small, it follows that any method designed to solve the global optimization problem would require an unbounded number of steps. Thus, generally, we shall not be able to find a point in S^* in finite time. Usually we will therefore consider the global optimization problem solved if we have found a point in

$$B_\epsilon(S^*) \equiv \{x \in S : ||x - x^*|| \leq \epsilon \text{ for some } x^* \in S^*\} \tag{7.3}$$

or in the level set

$$S_\epsilon \equiv \{x \in S : f(x) \geq f^* - \epsilon\} \tag{7.4}$$

for some $\epsilon > 0$ [Dixon].

7.1 Introduction

In this chapter we will discuss stochastic methods for solving the following general global optimization problem (P). Stochastic methods will be understood to be methods that contain some stochastic elements. This means that either the outcome of the method is itself a random variable (see Sections 7.3, 7.2 and 7.4), or the objective function is considered to be a realization of a stochastic process. Therefore, we will have to sacrifice the possibility of an absolute guarantee of success. Instead, we will usually aim at proving that, as the effort increases to infinity, an element of $B_\epsilon(S^*)$ or S_ϵ will be found with probability one.

Section 7.3 discusses the so-called *two-phase methods*, i.e. methods which use both random sampling (the global phase) and local optimization (the local phase). In Section 7.2 random search methods are described, which leads naturally to the class of Simulated Annealing algorithms in Section 7.4.

7.2 Random Search Methods

The class of random search methods consists of algorithms which generate a sequence of points in the feasible region following some prespecified probability distribution, or sequence of probability distributions. The most basic algorithms from this class proceed by generating points from a single probability distribution, i.e. the points are independently and identically distributed random variables. Alternatively, the distribution from which a point in the sequence is generated can be updated adaptively, i.e. depending on the iteration number and on previous iteration points.

The algorithms discussed in this section are of a conceptual nature, in the sense that at this point there does not exist an efficient implementation of these algorithms. However, the theoretical results that can be obtained for these algorithms are interesting in itself. Moreover, as we will see, they have a potential for inspiring (or theoretically supporting) more practical algorithms for global optimization.

7.2.1 Pure Random Search

he simplest algorithm from the class of random search methods is the Pure Random Search algorithm, which consists of generating a sequence of i.i.d. uniform points in the feasible region S, while keeping track of the best point that is found. This algorithm offers a probabilistic asymptotic guarantee in the sense that the global maximum will be found

with probability one as the sample size grows to infinity. It is interesting to note that this convergence result continues to hold if we replace the uniform distribution by any distribution whose support has a nonempty intersection with S. An interesting question is

Algorithm 7.10 Pure Random Search (PRS)

Step 0. Set $n = 1,\ y_0 = -\infty$.
Step 1. Generate a point x from the uniform distribution over S.
Step 2. If $f(x) > y_{n-1}$, then set $y_n = f(x)$ and $x_n = x$. Otherwise, set $y_n = y_{n-1}$ and $x_n = x_{n-1}$.
Step 3. Increment n and return to Step 1.

whether this algorithm has any advantage over its deterministic counterpart grid search, in which the function is evaluated in each point of a regular grid over S. One obvious advantage of PRS is that it can be implemented adapiively, i.e. the number of points generated does not need to be decided in advance. Another advantage is that the term regular grid is not well-defined for some arbitrary number of points $\Xi^{\cdot}$ over an arbitrary set S. The two algorithms have been more extensively analyzed by e.g. [Anderssen and Bloomfield]. The result of their analysis is that the points of the random sample cover S more efficiently (according to some probabilistic criterion) than grid points do, at least if the dimension of the problem is not too low.

7.2.2 Random Search

Let $\{\mu_n\}_{n=0}^{\infty}$ be a sequence of probability distributions on R^d. Then consider the following conceptual algorithm: The map D with domain $S \times R^d$ and range S satisfies the following

Algorithm 7.11 Random Search

Step 0. Set $n = 0$ and choose $x_0 \in S$.
Step 1. Generate y_{n+1} from the distribution μ_n.
Step 2. Set $x_{n+1} = D(x_n, y_{n+1})$, increment n and return to Step 1.

condition:

$$f(D(x,y)) \leq f(x) \text{ and if } y \in S,\ f(D(x,y)) \leq f(y)) \tag{7.5}$$

This condition ensures that the sequence $\{f(x_n)\}_{n=0}^{\infty}$ is monotone increasing with probability one. In fact, if:

$$\prod_{n=0}^{\infty}(1 - \mu(A)) = 0 \tag{7.6}$$

for all sets A that $\phi(A) > 0$, this sequnce will converge to f^* with probability one. If the random search algorithm is adaptive, i.e. if we allow the distribution μ_n to depend on the values of $x_0, ..., x_n$, then the convergence issue becomes more complicated. In the remainder of this section we will discuss two classes of adaptive random search algorithms for which convergence to the global optimum is obvious.

7.2.3 Pure Adaptive Search

The Pure Adaptive Search (PAS) algorithm differs from the PRS algorithm in that it forces improvement in each iteration. Because of this feature the PAS algorithm fits in the general class of random search algorithms discussed above. In PAS, an iteration point is generated from the uniform distribution on the subset of points that are improving with respect to the previous iteration point. More formally, the algorithm reads: [Zabinksy

Algorithm 7.12 Pure Adaptive Search (PAS)

Step 0. Set $n = 0$ and choose $y_0 = \infty$.
Step 1. Generate x_{n+1} uniformly distributed in $S_{n+1} = \{x \in S : f(x) < y_n\}$.
Step 2. Set $y_{n+1} = f(x_{n+1})$. Increment n and return to Step 1.

and Smith] provide a theorem that states that for a large class of global optimization problems with convex feasible regions in R^d, exists an upper bound on the expected number of iterations to achieve a solution arbirtary close to a global optimum.

Unfortunately, in practice, we encounter the following difficulties with directly implementing the Pure Adaptive Search algorithm:

1. Constructing the $i$mproving region

$$S_n = \{x \in S : f(x) > f(x_{n-1})\}$$

2. Generating a point uniformly distributed in S_n.

One way to avoid these difficulties is by using the acceptance-rejection method for generating points in S_n. That is, generate points uniformly in the feasible region S until we find a point that is in S_n.

7.2.4 Adaptive Search

In the Adaptive Search framework, points should be generated from the Boltzmann distribution π_T, with density function

$$g_T(x) \propto e^{f(x)/T}$$

where T is a "small"positive number. This is appropriate because for small T the distribution π_T will"concentrate near the global maximum". An important advantage of this algorithm is that sampling is done from the feasible region S, instead of from a nested set of smaller level sets of f. This avoids the two difficulties of PAS listed in the previous section. The price that has to be paid for this is that the distribution from which we have to sample changes during the course of the algorithm.

The number of trial points necessary in Step 1 can be influenced by an appropriate choice of the parameter T in Step 2 (where this choice will depend on the particular

Algorithm 7.13 Adaptive Search (AS)

Step 0. Set $n = 0$, $T_0 = \infty$ and $y_0 = \infty$.

Step 1. Generate x from the distribution π_{T_n} over S. If $f(x) < y_n$, set $x_{n+1} = x$. Otherwise, repeat from Step 1.

Step 2. Set $y_{n+1} = f(x_{n+1})$ and set the parameter $T_{n+1} = \tau(y_{n+1})$, where τ is an R^+-valued nonincreasing function. Increment n and return to Step 1.

shape of the distributions π_T). In accordance with the usual convention we will refer to the parameter T as the temperature parameter. A particular choice of temperature paB˜rameters $\{T_{nn=0}\}^{\infty}$ is then called a cooling schedule. The proposed cooling schedule for Adaptive Search is the following: given the current best function value y_n, choose the temperature T_n in such a way that a random variable generated from π_{T_n} has a better function value than y_n with probability at least $1\beta\alpha$. In this case the expected number of points that have to be generated at each temperature is at most $\frac{1}{1-\alpha}$. This analysis leads to the cooling schedule:

$$\tau(y) = \frac{f^* - y}{\gamma_{1-\alpha}(\nu, 1)} \tag{7.7}$$

where $\gamma_{1-\alpha}(\nu, 1)$ is the critical value of a gamma distributed random variable with parameters ν and 1 at level $1\beta a$.

7.2.5 Controlled Random Search (CRS)

There are several versions of this method that is based upon heuristics. We describe here a modification of Price's [128] algorithm, similar but not identical to the one described in [17]. The method seeks for one global minimum in a given domain S. Here the feasible domain S is considered to be a rectangular hyperbox. The algorithm has been designed for problems where the objective function is affected by the presence of noise and its gradient is not analytically available. Such problems, in the case of local optimization are treated with reasonable success by the irregular Simplex method [109], described in section 2.2.2. CRS is inspired in part by the tactic followed in that method, i.e. maintaining a population of points and performing operations such as reflection with respect to a centroid, etc.

Note that if more than one global minima exist, this method will locate only one of them.

The steps of the procedure are given by: CRS is simple to code and hence it has been frequently used in several applications. There exist several versions of the algorithm in the literature; however the main idea is common to all of them and no major performance differences have been observed.

Algorithm 7.14 Controlled Random Search

Step 0. Set $M > d+1$,$\epsilon = 10^{-6}$, $\omega = 1000$
Set $k = 0$, $S^k = \{x_1^k, x_2^k, \ldots, x_M^k\}$
Evaluate $f_i^k = f(x_i^k)$ for $i = 1, 2, \ldots, , M$

Step 1. $f_{max}^k = \max\{f_i^k\}$, and let the corresponding point be denoted as x_{max}^k.
$f_{min}^k = \min\{f_i^k\}$, and the corresponding point is denoted as x_{min}^k.
If $f_{max}^k - f_{min}^k \leq \epsilon$, polish x_{min}^k via a local search procedure and STOP.

Step 2. Choose at random $N+1$ points $\{x_{i_0}^k, x_{i_1}^k, \ldots, x_{i_N}^k\}$ from S^k.
Calculate the weighted centroids:

$$c_w^k = \sum_{j=1}^{N} w_j^k x_{ij}^k, \quad f_w^k = \sum_{j=1}^{N} w_j^k f(x_{ij}^k)$$

where:

$$w_j^k = \frac{n_j^k}{\sum_{j=1}^{N} n_j^k}, \quad n_j^k = \frac{1}{f(x_{ij}^k) - f_{min}^k + \phi^k}, \quad \phi^k = \omega \frac{(f_{max}^k - f_{min}^k)^2}{f_{max}^0 - f_{min}^0}$$

Calculate a trial point $\overline{x}^k$as:

$$\overline{x}^k = (x_{i_0}^k - c_w^k) \frac{f(x_{i_0}^k) - f_w^k}{f_{max}^k - f_{min}^k + \phi^k} + \Delta_w^k$$

where $\Delta_w^k = 2c_w^k - x_{i_0}^k$ if $f_w^k \leq f(x_{i_0}^k)$ and $\Delta_w^k = 2x_{i_0}^k - c_w^k$ if $f_w^k > f(x_{i_0}^k)$.
If $\overline{x}^k \notin S$ repeat Step 2.
Compute $f(\overline{x}^k)$.

Step 3. If $f(\overline{x}^k) \geq f_{max}^k$ then
Calculate the success rate (the fraction of function evaluations that led to a new lower upper bound).
If success rate $> 50\%$ then
Set $S^{k+1} = S^k$, $k = k+1$ and goto Step 2.
Calculate $y^k = \frac{c_w^k + x_{i_N}^k}{2}$, compute $f_y = f(y^k)$
If $f_y \geq f_{max}^k$ then
Set $S^{k+1} = S^k$, $k = k+1$ and goto Step 2.
Set $S^{k+1} = S^k \bigcup \{y^k\} - \{x_{max}^k\}$, $k = k+1$ and GOTO Step 1.

Step 4. Set $S^{k+1} = S^k \bigcup \{\overline{x}^k\} - \{x_{max}^k\}$.
Increment: $k = k+1$ and goto Step 1.

7.3 Two-phase Methods

In this section we will discuss stochastic methods in which two phases can be distinguished. Firstly, we have a global phase, in which the function is evaluated in a number of randomly sampled points. Secondly, in the local phase these sample points are manipulated, e.g.

by means of local searches, to yield a candidate global minimum. Most of these methods can be viewed as variants of the so-called *Multistart* algorithm. The global phase of this algorithm consists of generating a sample of points from a uniform distribution over S. In the local phase a local search procedure L is applied to each of these points, yielding various local minima. For a more extensive review of two-phase methods we refer to [Rinnooy Kan and Timmer]. Some essential properties of the local search procedure L will be presented in Section YY. The careful selection and implementation of the local search plays a very important role in two-phase methods and became a basic direction of our research.

7.3.1 Multistart

The simplest way to make use of a local search procedure L occurs in a method known as Multistart. This method is obviously much more attractive than Pure Random Search.

Algorithm 7.15 Multistart

Step 0. Set $n = 1,\ y_0 = -\infty$.

Step 1. Generate a point x from the uniform distribution over S and apply L to x yielding x'.

Step 2. If $f(x') > y_{n-1}$, then set $y_n = f(x')$ and $x_n = x'$. Otherwise, set $y_n = y_{n-1}$ and $x_n = x_{n-1}$.

Step 3. Increment n and return to Step 1.

However, the procedure is still lacking in efficiency. The main reason for this is that it will inevitably find each local maximum several times. Since local searches are the most time consuming part of the procedure, L should ideally be invoked no more than once in every *region of attraction*, where the region of attraction of the local maximum x_i^* is defined as the set of points in S starting from which L will converge to x^*. The clustering methods and the Multi Level Single Linkage algorithms discussed in the following two subsections have been designed with this objective in mind.

7.3.2 Clustering Methods

The basic idea behind clustering methods is to start from a uniform sample from S, to create groups of mutually "close" points, and to start L no more than once in each of those groups. Two ways to create such groups from the initial sample have been proposed. The first, called *reduction* [Becker and Lago]), only retains a fraction γ of the sample consisting of the points with the highest function values. The second, called *concentration* [Torn], transforms the sample by allowing one or a few steepest descent steps from every point.

The basic framework for identifying clusters of points that result from the sample obtained by one of the above methods is always the same. Clusters are formed in a stepwise fashion, starting from a seed point, which may be the unclustered point with the

highest function value or the local maximum found by applying L to this point. Points are then added to the cluster through application of a clustering rule.

In the remainder we will always use the reduction method as a starting point for clustering. The reason for this is that with this method, within each cluster the points will be uniformly distributed. Moreover, the clusters will correspond to connected components of level sets. These two properties provide a powerful tool in determining statistically correct clustering rules. The two most popular clustering rules are either density clustering or Single Linkage clustering.

Density Clustering

A stepwise description of the density clustering algorithm is: The critical distance $r_i(x)$,

Algorithm 7.16 Density Clustering

Step 0. Set $k = 1,\ X^* = \emptyset$.

Step 1. Generate Ξ points, $x_{(k-i)N+i}, \ldots, X_{kN}$, from the uniform distribution over S, and determine the reduced sample consisting of the γkN best points from the sample $\Xi_1, \ldots, x_{kN}$. Set $i = 1$ and $j = 1$.

Step 2. If all reduced sample points have been assigned to a cluster, go to Step 4. If $j < |X^*|$ then choose the j-th local minimum in X^* as the next seed point and go to Step 3. If $j > |X^*|$, then apply L to the unclustered reduced sample point $\bar{x}$ with the lowest function value. If the resulting local maximum x^* is an element of X^*, then assign $\bar{x}$ to the cluster initiated by x^* and return to Step 2. If $x^* \notin X^*$, then add x^*, to X^* and let x^* be the next seed point.

Step 3. Add all unclustered reduced sample points which are within distance $r_i(x^*)$ of the seed point x^*, to the cluster initiated by x^*. If no point has been added to the cluster for this specific value of $r_i(x^*)$, then increment j and return to Step 2, else increment i and repeat Step 3.

Step 4. Increment k and return to Step 1.

is chosen by [Rinnooy Kan and Timmer] to be equal to

$$r_i(x) = \frac{1}{\sqrt{\pi}} \left(i\Gamma(1 + \tfrac{d}{2}) \cdot m(S) \cdot \frac{\zeta ln(kN)}{kN} \right)^{1/d} \tag{7.8}$$

where H(x) denotes the Hessian at the point x and ζ is some positive constant. The idea behind this implementation is that the level set of the function f in the neighborhood of a local maximum is approximated by an ellipsoid, or, in other words, the function f is locally approximated by a quadratic function. Hence, the success of the method depends on how well this approximation is. A method which does not a priori fix the shape of the clusters is the Single Linkage clustering method.

Single Linkage Clustering

In the Single Linkage method [Rinnoy Kan and Timmer] the clusters are formed sequentially. Again, each cluster is initiated by a seed point. After a cluster C is initiated, we find an unclustered point x such that

$$d(x, C) = \min_{y \in C} ||x - y||$$

is minimal This point is than added to C, after which the procedure is repeated until d(x, C) exceeds some critical value r*. Experiments suggest that Single Linkage clustering approximates the level sets more accurately than density clustering. A stepwise description of the Single Linkage method is: The suggested value for the critical distance in Step 3

Algorithm 7.17 Single Linkage

Step 0. Set $k = 1,\ X^* = \emptyset$.

Step 1. Generate Ξ points, $x_{(k-i)N+i}, \ldots, X_{kN}$, from the uniform distribution over S, and determine the reduced sample consisting of the γkN best points from the sample $\Xi_1, \ldots, x_{kN}$. Set $j = 1$.

Step 2. If all reduced sample points have been assigned to a cluster, go to Step 4. If $j < |X^*|$ then choose the j-th local minimum in X^* as the next seed point and go to Step 3. If $j > |X^*|$, then select as the next seed point the unclusterd reduced sample point $\bar{x}$ with the lowest function value. Apply L to $\bar{x}$ to find a local maximum x^*, and add x^*, to X^*

Step 3. Add all unclustered reduced sample points which are within distance r_k of a point already in the cluster initiated by the seed point selected in Step 2 to the cluster. Increment j and return to Step 2.

Step 4. Increment k and return to Step 1.

is:

$$r_k = \frac{1}{\sqrt{\pi}} \left(\Gamma(1 + \tfrac{1}{d}) \cdot m(S) \cdot \frac{\zeta ln(kN)}{kN} \right)^{1/d} \tag{7.9}$$

his choice guarantees that, if $\zeta > 2$, the probability that a local search is started by Single Linkage in iteration k tends to zero with increasing k. Moreover, if $\zeta > 4$, then, even if the sampling continues forever, the total number of local searches ever started by Single Linkage is finite with probability one. This attractive property theoretically proves the efficiency of the method in terms of number of local searches performed. However, in the process we lost the asymptotic guarantee of success. While Single Linkage is guaranteed to find a local optimum in every connected component of the level set

$$\{x \in S : f(x) \geq y_\gamma\}$$

where y_γ is chosen in such a way that this level set has an $\phi-$measure γ.

Typical Distance Clustering

A clustering procedure may form clusters of points by measuring the distanceof a candidate cluster point from the estimated center of the cluster. Thisdistance is checked against a threshold and a decision is made accordingly.This threshold should be chosen properly and depends on the problem under consideration. Hence to avoid the introduction of ad hoc threshold values, an adaptive threshold called "*typical distance*" (r_t) is defined as:

$$r_t \equiv \frac{1}{M} \sum_{i=1}^{M} |x_i - L(x_i)| \tag{7.10}$$

Here x_i are starting - points for the local search procedure L, and M is the number of the performed local searches. The main idea behind equation (7.10) is that after a number of iterations and a number of local searches the quantity r_t will be a reasonable approximation for the mean radius of the regions of attraction. To see this note that if we denote by M_l the number of times that the local search procedure discovered the minimizer x_l^*, then a mean radius for the region of attraction related to x_l^* may be defined as:

$$R_l = \frac{1}{M_l} \sum_{j=1}^{M_l} \left| x_l^{(j)} - x_l^* \right| \tag{7.11}$$

where $\left\{ x_l^{(j)},\ j = 1, \ldots, M_l \right\} = \{x_i,\ i = 1, \ldots, M\} \cap A(x_l^*)$, i.e. $L\left(x_l^{(j)}\right) = x_l^*$. Since by definition $M = \sum_{l=1}^{w} M_l$, where w is the number of local minima discovered so far, a mean radius may be defined as:

$$< R > \equiv \sum_{l=1}^{w} \frac{M_l}{M} R_l = \frac{1}{M} \sum_{l=1}^{w} \sum_{j=1}^{M_l} \left| x_l^{(j)} - x_l^* \right| \tag{7.12}$$

Comparing eqs. (7.10), (7.11) and (7.12), it follows that $r_t =< R >$.
A point x is considered to be a "start point" if none of the following conditions is satisfied:

- There is an already located minimum z that satisfies the conditions
  1. $(x - z)^T (\nabla f(x) - \nabla f(z)) > 0$.
  2. $|x - z| < \min_{i,j\ i \neq j} |z_i - z_j|,\ z_i \in X^*,\ z_j \in X^*$.
- x is near to another point $y\ \in V$ that satisfies the conditions
  1. $|x - y| < r_t$.
  2. $(x - y)^T (\nabla f(x) - \nabla f(y)) > 0$.

Hessian-based ISO-OCT Clustering

This method was proposed from [Tu and Mayne] and uses Hessian information to identify clusters around minima. The authors intuition is based that in the neighbourhood of

Algorithm 7.18 Typical Distance Clustering

Step 0. Set $k = 1,\ X^* = \emptyset$.

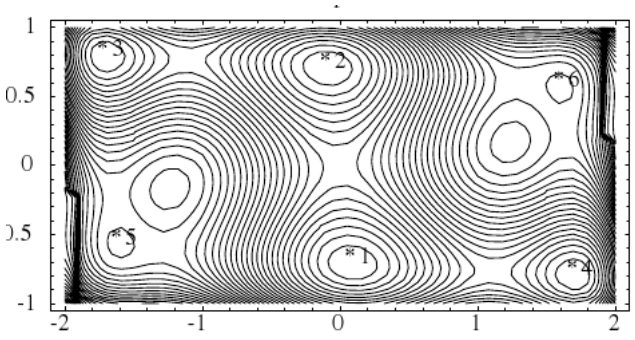
Step 1. Set $V = T = \emptyset$.
Sample N points via the Double Box procedure and add them to T.
For all $x \in T$ do
Check if x is a valid starting point and if so add it to V.

Step 2. If $\frac{|V|}{N} < \frac{1}{2}$ Then
$N = \min\left(N + \frac{N}{10}, \text{NMAX}\right)$

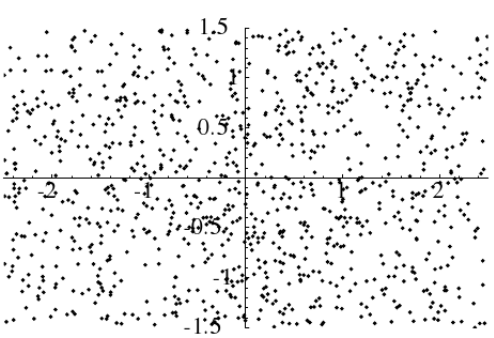
Step 3. For all $x \in V$ do:
If x is a start point Then
Start a local search $y = L(x)$
Compute the typical distance r_t using equation (7.10).
Add y to X^*

Step 4. Increase k and repeat from Step 1.

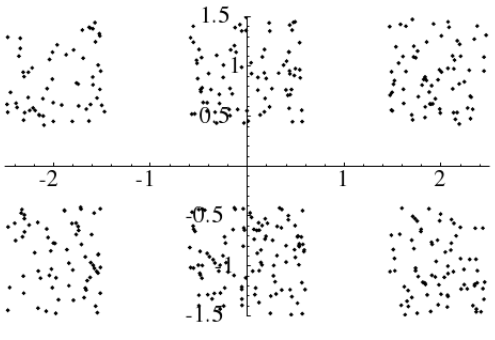
local minima, the objective function is convex and any two isolated local minima must be separated by a region where the function is non-convex, that is, the Hessian matrix is either negative definite or indefinite.



(a) Six-sump camel back



(b) Initial sample



(c) Sample with positive definite Hessian

Figure 7.1: An illustration of Hessian information

A description of this method is given in Algorithm7.19:

Algorithm 7.19 Hessian-based ISO-OCT Clustering

Step 0. Initialize M, n, λ.

Step 1. Generate Ξ random points x_i

Step 2. Compute the clustering parameters δ and s using Eqs 7.137.14.

Step 3. Compute the gradient $g(x_i) = \frac{\partial f(x_i)}{\partial x_x}$

Step 4. Find n closest points of x_i and compute the Hessian matrix $H(x_i)$ using Eq 7.15.

Step 5. Compute the eigenvalues of $H(x_i)$. Then compute the scaled eigenvalues $\lambda_{ij},\ i, j = 1, \ldots, N$ Discard x_i if $\lambda_{ij} < 0 \forall j$. Retain the 80% of the points with the largest scaled eigenvalues. .

Step 6. Identify clusters by applying the clustering analysis procedure ISO-OCT to the Sreduced sample points.

Step 7. Perform local search L starting from the point with the smallest function value in each cluster.

Step 8. Repear from Step 1.

$$\delta = 0.2 \left\{ \frac{1}{2} \left(\frac{1}{n} \sum_{i=1}^{n} |u_i - l_i| + \frac{1}{n} \sum_{i=1}^{n} |x_i^{\max} - x_i^{\min}| \right) \right\} \tag{7.13}$$

$$\sigma_s = 0.5 \cdot \delta \tag{7.14}$$

$$H^{k+1} = H^k + \frac{(y_k - H^k s^k)(y_k - H^k s^k)^T}{(y_k - H^k s^k)^T s^k} \tag{7.15}$$

where $y^k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$

7.3.3 Multi Level Single Linkage

Multi Level Single Linkage is a method with combines the computational efficiency of clustering methods with the theoretical virtues of Multistart. The local search procedure L is applied to every sample point, except if there is another sample point within some critical distance which has a larger function value. A stepwise description of the algorithm is the following: The critical distance is again chosen to be

$$r_k = \frac{1}{\sqrt{\pi}} \left(\Gamma(1 + \tfrac{1}{d}) \cdot m(S) \cdot \frac{\zeta ln(kN)}{kN} \right)^{1/d} \tag{7.16}$$

In spite of its simplicity the theoretical properties of this algorithm are quite strong [Rinnooy Kan and Timmer]:

1. If x is an arbitrary sample point, then the probability that L is applied to x in iteration k tends to zero with increasing k.

2. If $\zeta > 2$, then the probability that a local search is applied in iteration k tends to zero with increasing it.

Algorithm 7.20 Multi Level Single Linkage

Step 0. Set $k = 1,\ X^* = \emptyset$.

Step 1. Generate Ξ points, $x_{(k-i)N+i}, \ldots, X_{kN}$, from the uniform distribution over S, and determine the reduced sample consisting of the γkN best points from the sample $\Xi_1, \ldots, x_{kN}$. Set $i = 1$.

Step 2. If there exists some j such that $f(x_j) > f(x_i)$ and $||x_j \beta x_i|| < r_k$ then go to Step 3. Otherwise, apply L to x_i, and add the local minimum found to X^*.

Step 3. Increment i. If $i < kN$, go to Step 2. Otherwise, increment k and go to Step 1.

3. If $\zeta > 4$, then the total number of local searches started by MLSL is finite with probability one.

4. Any local maximum will be found by MLSL within a finite number of iterations with probability one.

The methods discussed in this section all aim at finding all local optima of the optimization problem (P). However, expecially in cases where the number of local optima is very large, this will not be the best strategy. Therefore in the next two sections we will discuss random search methods, which directly aim at finding the global optimum.

7.3.4 Healed Topographical Multilevel Single Linkage

We describe now a stochastic method based on the MLSL algorithm of Rinnoy Kan and Timmer[134], integrated with ideas from Viitanen [148] and Ali [3]. A healing technique along with a threshold on the number of iterations is used, to prevent premature termination at the early stages of the algorithm. As it can be readily realized, healing, protects the algorithm from premature termination, by delaying the growth of the t-values for a number of initial iterations. As an additional control parameter, a threshold I_t on the minimum number of iterations is used. This forces the algorithm to iterate for at least I_t times.

7.3.5 Random Linkage

Random Linkage alogirithm was introduced in [Locatelli and Schoen] and as stepwise description follows: It is understood that $\delta_k = \infty$ if $\nexists j : f(x_j) > f(x)$. The functions ϕ_k play the role of probabilistic thresholds. The whole algorithm may be seen as a generalized acceptance/rejection method. If we set

$$\phi_k(\delta) = \begin{cases} 1 \text{ if } \delta \geq a_k \\ \\ 0 \text{ otherwise} \end{cases}$$

Algorithm 7.21 Healed Topographical Multi Level Single Linkage

Step 0. Set $k = 1,\ X^* = \emptyset$.

Step 1. Generate Ξ points, $x_{(k-i)N+i}, \ldots, X_{kN}$, from the uniform distribution over S, and determine the reduced sample consisting of the γkN best points from the sample $\Xi_1, \ldots, x_{kN}$. Set $i = 1$.

Step 2. This is the step that characterizes the method as "Topographical MLSL". In this step we first add to the sample the already found (initially none) local minima. So the sample contains $N + w$ points, w being the number of the local minima found so far. For every point $r_i \in S$ we find its c closest neighbors $b_{ij},\ \ j = 1, \cdots, c$. If $f(r_i) \leq f(b_{ij}), \forall j = 1, \cdots, c$, then the point r_i is called a graph minimum. The start points for the local searches are chosen from within the set of the graph minima. A point from that set is a start point as long as:

1. It is not a local minimum found earlier, and
2. There is no other point within a critical distance r_k, with a lower function value.

Step 3. Increment k and go to Step 1.

Algorithm 7.22 Random Linkage

Step 0. Set $k = 0$.

Step 1. Sample a single point x_{k+1} from the uniform distribution over S

Step 2. Start a local search L from x_{k+1} with *probability*: $\phi_k\left(\delta_k(x_{k+1})\right)$, where $\delta_k = \min\{||x - x_j|| :\ j = 1, \ldots, k,\ f(x_j) > f(x)\}$

Step 3. Increase k and repeat from Step 1

where a_k is a sequence of non-negative real numbers then we define the special sub-class of Random Linkage Algorithms, denominated *Threshold Random Linkage*. For this special class if

$$\lim_{k\to\infty} k^{1/d} a_k = \infty$$

then the probability of starting a local search tends to 0 as $k \to \infty$

7.4 Simulated Annealing

Simulated Annealing is a random search technique that avoids getting trapped in local maxima by accepting, in addition to transitions corresponding to an increase in function value, also transitions corresponding to a decrease in function value. The latter is done in a limited way by means of a probabilistic acceptance criterion. In the course of the maximization process, the probability of accepting deteriorations descends slowly towards

zero. These "deteriorations" make it possible to move away from local optima and explore the feasible region S in its entirety.

Simulated Annealing originated from an analogy with the physical annealing proB˜cess of finding low energy states of a solid in a heat bath (see Metropolis et al. [59]). Pincus [66] developed an algorithm based on this analogy for solving discretizations of continuous global optimization problems.

7.4.1 The Algorithm

Recall that the Adaptive Search algorithm is based on the following property of the family of Boltzmann distributions:

$$\lim_{T \to 0} \pi_T(S_\epsilon) = 1 \tag{7.17}$$

for all $\epsilon > 0$. This same property is the basis of the Simulated Annealing algorithm. In fact, in this section we will show how Simulated Annealing can be derived as an approximation of Adaptive Search.

As noted in the previous section, the Adaptive Search is basically a conceptual algorithm, since, in general, it will be extremely difficult to generate points directly from the distribution π_T. Therefore, consider taking the following, approximating approach. Suppose we have a random walk on S, which converges to the uniform distribution on S. Let the transition probability distribution given that the Markov chain is in state $x \in S$ be denoted by $R(x, \cdot)$. We can then filter this random walk as follows. In every iteration, given iteration point x_n, we generate a point z_{n+1} from $R(x_n, \cdot)$. Then, we accept this point with probability:

$$\min\{1, e^{-(f(z_{n+1}-f(x_n)))/T}\}$$

(the *Metropolis criterion*), i.e. with this probability we set $x_{n+1} \leftarrow z_{n+1}$. If we filter the Markov chain given by R in this way, the sequence of points generated will converge to the Boltzmann distribution π_T. So, we can generate a sequence of points $\{X_n(T)\}_{n=0}^{\infty}$ with the property that for every $\epsilon > 0$

$$\lim_{n \to \infty} Pr(X_n(T) \in S_\epsilon) = \pi_T(S_\epsilon) \tag{7.18}$$

Combining Eq. 7.17 and Eq. 7.18 we get

$$\lim_{T \to 0} \lim_{n \to \infty} Pr(X_n(T) \in S_\epsilon) = 1 \tag{7.19}$$

for all $\epsilon > 0$. The Simulated Annealing algorithm is motivated by Eq. 7.19. It consists of generating a sequence $X_0, X_1, \ldots$ using the filtered random walk described above, except that now the temperature parameter Ξ will decrease to zero as we proceed, according to an adaptive cooling schedule denoted by $\tau_n(x_0, \ldots, x_n)$. We can formulate the general Simulated Annealing algorithm as follows.

For the simulated algorithm described in Alg. 7.23, $f_n^* \to f^*$ almost surely, as $n \to \infty$.

Algorithm 7.23 Simulated Annealing

Step 0. Set $n = 0$ choose $x_0 \in S$ and $T_0 \in [0, \infty]$.

Step 1. Select y_{n+1} according to the probability $R(x_n, \cdot)$.

Step 2. If $f(y_{n+1}) \leq f(x_n)$, set $x_{n+1} = y_{n+1}$. If $f(y_{n+1}) > f(x_n)$, set $x_{n+1=y_{n+1}}$ with probability $\exp((f(x_n) - f(y_{n+1}))/T)$. Otherwise, set $x_{n+1} = x_n$

Step 3. Set $T_{n+1} = \tau_{n+1}(x_0, \ldots, x_{n+1})$, increment n amd return to Step 1.

7.4.2 Practical Implementations

In this section we will discuss several specific Simulated Annealing algorithms from the literature.

Table 7.1: Simulated Annealing algorithms

Authors	**Candidate point generation**	**Cooling scheme**
Vanderbilt and Louie []	$y = x + Qu, \quad u \in U(-\sqrt{3}, \sqrt{3}),\ QQ^T = H$	$T_{n+1} = \chi T_n$
Bohachevsky, Johnson and Stein []	$y = x + \Delta r \frac{(u-x)}{\|\|u-x\|\|}, \quad u \in R^d$	$T_n = \frac{(f^* - f(x_n))^g}{\beta}, \quad g, \beta > 0$
Corana et al. []	$y = x + Ue_h$, e_h unit vector $\in R^d, U \in U(-u_h, u_h)$	$T_{n+1} = \chi T_n$
Romeinj and Smith	$y = x + \lambda\theta, \quad \|\|\theta\|\| = 1$	$T_n = \frac{f^* - f(x_n)}{\gamma_{1-\alpha}(\nu,1)}1,$
Dekkers and Aarts []	Uniformly in S, with probabilty p apply local search	

7.5 Genetic Algorithms

Genetic Algorithms (GA), introduced by Holland [71], fall in the class of evolutionary algorithms, and they certainly have become quite popular recently. There is a host of articles, conference proceedings and books dedicated to their introduction, illustration and description. We refer for example to Bäck [4], Davis [32], Fogel [47], Goldberg [55], Michalewicz [103], Rawlins [131], Whitley [159], [160] and Schwefel [139].

These methods search the space by letting a population of candidate solutions evolve in an environment governed by rules inspired from the field of genetics; namely, crossover and mutation. The individual solutions tend to improve over the generations, mimicking the evolution of living species. Genetic algorithms require only function values and not gradient or Hessian information, hence they are applicable as well to problems where the objective function is non-smooth or contains noise. On the other hand, GAs may converge, and usually do, at a very slow rate towards an accurate solution. To cure this

inefficiency, several methods blend GAs with local search techniques and this combination has proved to be quite satisfactory. GAs have been successfully applied to a host of different optimization problems, like wire routing, scheduling, neural network training, portfolio management, differential equation solving, etc. In the literature there are quite a few, slightly different, methods that are based on genetic algorithms, and this may be causing some confusion. However each variant is simply an adaptation of the underlying basic algorithm, for a specific problem. This "spinal" algorithm on which all these variants are based, may be described as follows:

1. **Initialization** : Create an initial population of individuals (points) randomly.

2. **Elitism** : Use a fitness function to rate each individual. This function is related to the objective function and hence it depends on the problem at hand. The "fittest" individuals survive, i.e. they will be present in the population of the next generation.

3. **Selection** : Individuals are selected for further genetic processing. The selection, that is of stochastic nature, is biased by the fitness value.

4. **Production** : The selected individuals are transformed by genetic-like operations to reproduce "children" for the next generation. These operations are:

   a. **Crossover** : Probabilistic recombination of two parents to result in the production of two children. There are various ways to implement this. A simple one for continuous problems is to use linear combinations of the parents.

   b. **Mutation** : Probabilistic random modification of a number of individuals. This operation enhances the diversity of the population . There are many possible ways as far as implementation is concerned.

   The steps 2 through 4 are repeated until a termination condition is satisfied. There are several termination criteria. A commonly used one, is to stop when a preset upper bound for the number of generations is reached. Another is based on measuring the diversity of the produced population and decide to stop when this is below a preset threshold.

Practical implementations use various schemes for the fitness function, as well as for the elitism, crossover and mutation operators. If the objective function is denoted by $f(.)$, the fitness function for the current generation may be casted as a probability measure for survival. Let $x_i, \forall i = 1, 2, \cdots, M$ be the members (points) of the generation of population size M. Then the probability for survival (fitness) may be written as:

$$P(x_i) = \frac{f_{max} + \epsilon - f(x_i)}{M(f_{max} + \epsilon) - \sum_{j=1,M} f(x_j)} \tag{7.20}$$

ϵ being a small number (e.g. 10^{-7}), and $f_{max} = \max_{i=1,M}\{f(x_i)\}$.

Note that $P(x_i) > 0 \ \forall i = 1, 2, \cdots, M$ and that $\sum_{i=1,M} P(x_i) = 1$.

Elitism may be a deterministic operation, for example:

"Pick K members with the highest fitness values"

or it may be complemented with a stochastic process of the sort:

"Pick additional L members at random, taking in account their fitness".

Hence a member x_i will be selected with a probability $P(x_i)$, as given by relation (7.20). This procedure is also known as the *"Roulette Wheel Selection"*.

Maintaining the fittest member through the generations corresponds to imposing $K \geq 1$. Note that multiple copies of a highly fit member may occur in the generation to come. This "cloning" enhances the local character of the search, however, since the diversity of the population is decreased, it may lead to a premature termination.

Crossover is a stochastic operation. A couple of parents (i.e. two distinct points) are selected via the *Roulette Wheel* mechanism. Children then may be produced in a number of ways.

Component Exchange : Let $P_f, P_m \in R^N$ be the points corresponding to the two parents and let Ch_1 and Ch_2 be "children" points to be created. With probability $\frac{1}{2}$ choose either $Ch_1(i) = P_f(i)$ and $Ch_2(i) = P_m(i)$, or $Ch_1(i) = P_m(i)$ and $Ch_2(i) = P_f(i)$, $\forall i = 1, 2, \cdots, N$.

Linear Combination : In this case the children points are linear combinations of the parent points, namely: $Ch_1(i) = P_f(i) + q_1(P_m(i) - P_f(i))$ and $Ch_2(i) = P_m(i) + q_2(P_f(i) - P_m(i))$. q_1 and q_2 are random mixing parameters and they are chosen in $(-d, 1+d)$, d being a small positive number typically around 0.25. Note that if we set $d = 0$, the "children" automatically satisfy box constraints. However since d is usually different than zero, children may not respect the box constraints and a corrective action should be performed, for instance a projection on the bounds.

Mutation is a stochastic operation that adds random noise to selected members. A choice made quite often is that as the generations proceed, the random noise added is decreasing in amplitude. The rationale for this choice is that at the begining, mutation is important for the exploration of the search domain. However as the generations advance and the space is rather well covered, there is no need to spend function calls at random points that are very likely to be useless.

Let N_g and I_g denote the maximum number of generations allowed and the current generation count. Let also $\beta > 0$ be the mutation parameter. Then from a parent P a child Ch is created component by component as:

$\forall i = 1, 2, \cdots, N$ choose with probability $\frac{1}{2}$ either

$$Ch_i = P_i + (b_i - P_i)(1 - r_i^{(1-\frac{I_g}{N_g})\beta})$$

or

$$Ch_i = P_i - (P_i - a_i)(1 - r_i^{(1-\frac{I_g}{N_g})\beta})$$

a_i, b_i are the lower and upper bounds for the i^{th} component and r_i is a random number in $(0, 1)$. Since $P_i \in (a_i, b_i)$, note that $Ch_i \in (a_i, b_i)$ as well, i.e. the box constraints are respected.

7.6 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) is a method inspired from the social behavior of a group of living beings, like "*bird flocking*" and "*fish schooling*". The social sharing of information helps birds to profit from the discoveries and the gained experience of other birds, in their search for food. This fact sparked the idea that such a model may be used as an optimization method. Note that other methods as well, have been inspired from natural processes, for example *Genetic Algorithms* and *Simulated Annealing*. PSO provides a population based stochastic search, in which the individual point–particles are moving according to rules that simulate a bird flock social behavior. The particles in the swarm explore a multidimensional space in search of promising regions, i.e. regions where the objective function assumes lower values. Each particle's position is influenced by its own observations and by information acquired from nearby particles. The best particle positions are taken under consideration. Local information is conveyed by the particle's best position while global information is conveyed by the best positions of the neighboring particles. PSO is suitable for minimizing all types of functions, continuous or not, multimodal, or functions containing noise. No gradient information is required and is not computationally expensive. Additional information may be found in the articles [71, 118, 117]. PSO caters for all types of objective functions, continuous or not, multimodal, containing noise, etc. It does not require gradient information, only function values and is very simple to implement. PSO methods have been successfully applied to many problems [138] such as structural optimization, scheduling, neural network training, etc.

7.6.1 Description and rationale

At first one creates a swarm of $\boldsymbol{M}$ particles, with positions x_i^0 and corresponding velocities v_i^0 $(i = 1, 2, \ldots, \boldsymbol{M})$, where the superscript denotes the number of time steps taken so far. The position of each particle represents a potential solution. The particles will start to move according to a relation of the form:

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{7.21}$$

Before discussing the way the velocities are updated a few comments are proper. Each particle keeps a record of its current position x_i^k, its best historical position b_i^k, in the sense that $f(b_i^k) = \min_{l=1,\ldots,k} f(x_i^l)$ and the best position y^k attained by any particle so far. Influence from a particle's own best position results to local exploration, while the influence from the overall best position a particle ever had, points to global optimality. Having observed the above we may proceed to derive a velocity update. A term proportional to $b_i^k - x_i^k$ directs

the particle towards its best attained position. A term proportional to $y^k - x_i^k$ directs the particle towards the globally best position attained so far. A linear combination of the above terms is considered, with random coefficients, i.e.:

$$v_i^{k+1} = v_i^k + c_1\xi_1(b_i^k - x_i^k) + c_2\xi_2(y^k - x_i^k) \tag{7.22}$$

ξ_1, ξ_2 are random numbers uniformly distributed in $(0, 1)$, while c_1, c_2 are parameters adjusting the velocity gain. Many authors have suggested the values $c_1 = c_2 = 2$. The drawback of the update as it stands in eq. (7.22), is that it allows too high velocities to occur, an undesirable fact that forces the swarm to drift past the region of the global minimum. To address this issue, an inertia weight w is introduced [38] transforming the update as:

$$v_i^{k+1} = wv_i^k + c_1\xi_1(b_i^k - x_i^k) + c_2\xi_2(y^k - x_i^k) \tag{7.23}$$

Large w values (i.e. values close to 1) encourage global exploration, while small values (i.e. values close to 0) facilitate local searches. This observation led to another modification by introducing a time-dependent inertia weight that initially assumes large values, so that global exploration is favored, and subsequently is gradually reduced to promote local tuning [141].

Another velocity update is based on the so called *constriction factor* denoted by χ. The scheme is quite similar to that of eq. (7.23). The advantage is that the value of χ has been derived analytically [25]. The update is given by:

$$v_i^{k+1} = \chi\left(v_i^k + c_1\xi_1(b_i^k - x_i^k) + c_2\xi_2(y^k - x_i^k)\right) \tag{7.24}$$

and χ is obtained by:

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{7.25}$$

where $\phi = c_1 + c_2$, and $\phi > 4$.

PSO has been successfully applied to a host of different problem classes, and has also been combined with other techniques such as *deflection, stretching and repulsion* [119], in order to retrieve not one but all the global minimizers, that an objective function may possess.

PSO is still unexplored and further research is required to comprehent the capabilities of this rather new technique. Many realistic applications await and it remains to be seen if PSO can cope with them. The simplicity, the robustness and its wide applicability render PSO an attractive method for global optimization.

Chapter 8

Towards "Ideal Multistart" A stochastic approach for locating the minima of a continuous function inside a bounded domain

8.1 Summary

A stochastic global optimization method based on *Multistart* is presented. In this, the local search is conditionally applied with a probability that takes in account the topology of the objective function at the detail offered by the current status of exploration. As a result, the number of unnecessary local searches is drastically limited, yielding an efficient method. Results of its application on a set of common test functions are reported, along with a performance comparison against other established methods of similar nature.

8.2 Introduction

Global optimization (GO) has received a lot of attention in recent years [116], due to the ever emerging scientific and industrial demand. For instance the description of the stable conformations of a molecule [99, 158, **?**], the management of mutual funds [167, 135, 9, 74], location and allocation issues [45, 70], engineering design and the design of drugs [46, 154], to mention a few topics, are in need of efficient global optimization techniques.

There exist several categories of GO methods. We distinguish two main classes: the deterministic [44, 73] and the stochastic one. For a detailed account on classification of stochastic algorithms we refer to [14]. Deterministic methods provide a theoretical guarantee of locating the global optimum. Stochastic methods offer only a probabilistic (asymptotic) guarantee: their convergence proofs usually declare that the global optimum will be identified in infinite time with probability one. Moreover, stochastic methods adapt

better to black-box formulations and extremely ill-behaved functions, whereas deterministic methods are usually based on at least some theoretical assumptions such as Lipschitz continuity and heavily depends on the problem at hand. A direct comparison between these two approaches may be found in [**?**], where the authors conclude that the stochastic approach is to be preferred. In addition deterministic methods suffer from the problem of dimensionality. For example, the complexity of interval global optimization [**?**] rises exponentially with the problem's dimension.

The problem we are interested in, may be expressed as:

$$\text{Find all } x_i^* \in S \subset R^n \text{ that satisfy:}$$
$$x_i^* = \arg\min_{x \in S_i} f(x), \quad S_i = S \cap \{x, ||x - x_i^*|| < \epsilon\} \tag{8.1}$$

S is considered to be a bounded domain of finite measure and ϵ a positive infinitesimally small number. We are adopting the stochastic class of methods. One of the most widely used stochastic algorithms is the so called *Multistart* [13]. It's popularity stems from it's simplicity and inherent parallelization [153, 98, 37, 152]. Many stochastic methods have been developed around it starting from the classic papers of [13, 133, 134, 150] were the popular *Single Linkage Clustering*, *Density Clustering* and *Multi–Level Single Linkage* procedures were introduced. Torn and Viitanen in [148] presented a *Topographical Clustering* algorithm which was extended by Ali and Storey in [3] to the well known *Topographical Multi–Level Single Linkage* algorithm. More recently Hart in his PhD dissertation [67] proposes an adaptive method based on clustering and local searches, Locatelli [93] introduces the family of *Random Linkage* algorithms and Schoen [137] and Locatelli [92] give an analysis *Two–phase methods*. More recently, Liang et. al. [91] introduce a function's landscape approximation, Bolton et. al. [16] provide a parallel framework based on clustering procedures while Tsoulos and Lagaris [151] proposed the so called *typical distance* clustering. Also related software may be found in [**?**].

In *Multistart* a point is sampled uniformly from the feasible region, and subsequently a local search is started from it. The weakness of this algorithm is that the same local minima may be found over and over again, wasting so computational resources. For this reason clustering methods have been developed that attempt to avoid repetitive discovery of the same minima [133, 134, 150, 151, 152].

The *Multistart* algorithm is presented bellow:

Multistart Algorithm

Initialize: Set k=1

Sample $x \in S$

$y_k = \mathcal{L}(x)$

Termination Control: If a stopping rule applies, STOP.

Sample: Sample $x \in S$

Main step: $y = \mathcal{L}(x)$

If ($y \notin \{y_i, i = 1, 2, \ldots, k\}$) Then

$k = k + 1$

$y_k = y$

Endif

Iterate: Go back to the Termination Control step.

The **"region of attraction"** of a local minimum associated with a local search procedure $\mathcal{L}$ is defined as:

$$A_i \equiv \{x \in S, \mathcal{L}(x) = x_i^*\} \tag{8.2}$$

where $\mathcal{L}(x)$ is the minimizer returned when the local search procedure $\mathcal{L}$ is started at point x. If S contains a total of w local minima, from the definition above follows:

$$\cup_{i=1}^{w} A_i = S \tag{8.3}$$

Let $m(A)$ indicate the *Lebesgue measure* of $A \subseteq R^n$. If we assume a deterministic search $\mathcal{L}$, then the regions of attraction do not overlap, i.e. $A_i \cap A_j = \emptyset$ for $i \neq j$, and from eq. (8.3) one obtains:

$$m(S) = \sum_{i=1}^{w} m(A_i) \tag{8.4}$$

If a point in S is sampled from a uniform distribution, the apriori probability p_i that it is contained in A_i is given by $p_i = \frac{m(A_i)}{m(S)}$. If K points are sampled from S, the apriori probability that at least one point is contained in A_i is given by:

$$1 - (1 - \frac{m(A_i)}{m(S)})^K = 1 - (1 - p_i)^K \tag{8.5}$$

From the above we infer that for large enough K, this probability tends to one, i.e. it becomes "asymptotically certain" that at least one sampled point will be found to belong to A_i. This holds $\forall A_i$, with $m(A_i) \neq 0$.

In this article we first define the *"Ideal Multistart"*, a variation of Multistart in which every local minimum is found only once. This ideal version assumes that the region of

attraction of a minimizer is determined as soon as the minimizer is located. Since this is a false hypothesis this version is of no practical value. It offers however a framework and a goal to work towards.

In section (8.3), we lay–out the new ideas involved and we present the corresponding algorithm, while in section (8.4), we give a description of the numerical experiments that were performed along with the results. Finally in section (8.6), our conclusions are summarized and we give a recommendation for future research

8.3 Description of the Method

"Ideal Multistart" starts by sampling a point from S and applying a local search leading to the first minimum y_1, with region of attraction A_1. Sampling points from S is continued until a point is found that does not belong to A_1. Once such a point is encountered, a local search is performed that leads to the second minimum y_2, having a region of attraction A_2. The next sample point from which a local search will start, is a point that belongs neither to A_1 nor to A_2, i.e. it does not belong to their union $(A_1 \bigcup A_2)$. This procedure goes on, until a stopping rule instructs termination. The detailed algorithm is laid out in the following paragraph.

8.3.1 Ideal Multistart

Ideal Multistart Algorithm

Initialize: Set k=1

Sample $x \in S$

$y_k = \mathcal{L}(x)$

Termination Control: If a stopping rule applies, STOP.

Sample: Sample $x \in S$

Main step: If $(x \notin \cup_{i=1}^{k} A_i)$ Then

$y = \mathcal{L}(x)$

$k = k + 1$

$y_k = y$

Endif

Iterate: Go back to the Termination Control step.

This algorithm invokes the local search procedure only w times, w being the number of existing minima of $f(.)$ in S. The main step is deterministic and requires the regions of attraction A_i of the already located minima to be known, which is not the case in practice. Hence we apply a stochastic modification to the main step, by allowing the local search to be performed with a probability, namely:

Main step (**Stochastic**):

Calculate the probability p, that $x \notin \cup_{i=1}^{k} A_i$

Draw a random number $\xi \in (0, 1)$ from a uniform distribution

If ($\xi < p$) Then

$y = \mathcal{L}(x)$

If ($y \notin \{y_i, i = 1, 2, \ldots, k\}$) Then

$k = k + 1$

$y_k = y$

Endif

Endif

This step requires the probability that a point does not belong to the region of attraction of any of the minima collected so far. This requirement is easier to fulfill, since even with a low accuracy estimate for the probability, the algorithm will succeed. Notice that an overestimated probability ($p \rightarrow 1$) will transform the algorithm into the usual *Multistart*. On the other hand underestimation ($p \rightarrow 0$) is not of considerable cost, since no local search is performed. Performance however will be optimized if reasonably accurate estimates for the probability can be calculated. Several ways may be designed to accomplish this goal. We suggest one in the following paragraph.

8.3.2 Estimating the local search probability

The required probability may depend on several factors, such as the distance from existing minimizers, the direction of the gradient, the number of times each minimizer has been discovered, etc. We consider how each factor influences the probability and combine them together to get the required estimate.

Let us define the *maximum attractive radius (MAR)* as:

$$R_i = \max_j \{||x_j^{(i)} - y_i||\} \tag{8.6}$$

where $x_j^{(i)}$ are the sampled points which led the subsequent local search to the i^{th} minimizer y_i.

Given a sampled point x, let y be anyone of the recovered minimizers, with *MAR* denoted by R. If $||y - x|| < R$, then x is likely to be inside the region of attraction of y. If however $\nabla f(x)^T(y - x) \geq 0$, i.e. the direction from x to y is ascent, then x is likely to be outside y's region of attraction. Letting $z \equiv ||y - x||/R$, then an estimate of the probability that $x \notin A(y)$ may be given by:

$$p(x \notin A(y)) = \begin{cases} 1, \text{ if } z > 1 \text{ or } \nabla f(x)^T(y - x) \geq 0 \\ \phi(z, l) * \left[1 + \frac{(y-x)^T \nabla f(x)}{||y-x||\nabla f(x)|}\right], \text{otherwise} \end{cases} \tag{8.7}$$

l is the number of times y has been recovered so far, while $\phi(z, l)$ is a model with the following properties.

$$\begin{aligned} &\lim_{z \to 0} \phi(z, l) \to 0 \\ &\lim_{z \to 1} \phi(z, l) \to 1 \\ &\lim_{l \to \infty} \phi(z, l) \to 0 \\ &0 < \phi(z, l) < 1 \end{aligned} \tag{8.8}$$

Notice that the factor inside the square brackets in eq. (8.7), varies from zero to one, as the gradient from anti-parallel becomes perpendicular to $y - x$.

The probability that $x \notin \cup_{i=1}^k A_i$ is given by the product $\prod_{i=1}^k p(x \notin A_i)$ and may now be approximated by the probability that $x \notin A_n$, A_n being the region of attraction of the nearest to x discovered minimizer y_n. The rationale for this approximation is that if $x \notin B(y_i, R_i)\ \forall i \neq n$, where $B(y, R)$ is a sphere of radius R centered at y, then the above approximation is exact since all other probabilities as following from eq. (7) equal 1. If on the other hand x is inside the intersection of two or more overlapping spheres, the product of small terms may result to too small a probability for a point that could lead to a new minimum (see in fig. 8.1, an example). The spheres are expected to overlap, due to the manner their radii are chosen by eq. (8.6). Hence the approximation is prudent, and essentially in most cases does not overestimate the local search probability. One may employ alternative approximations, by considering for example the first two (or more) nearest minimizers. This is an issue that needs further consideration and is outside the scope of the present article.

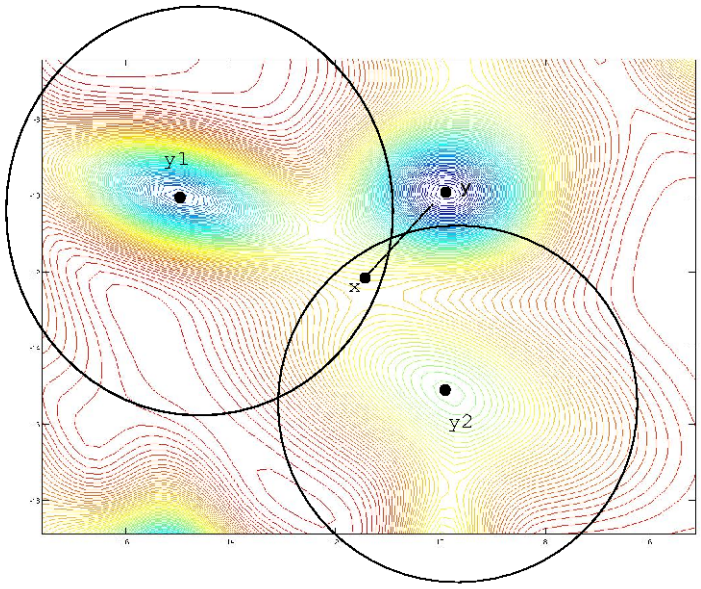


Figure 8.1: A point x that would lead to a new minimum y, is inside the overlap region of the spheres around two recovered minima y_1 and y_2

8.3.3 Local search properties

The probability model is based on distances from the discovered minima. It is implicitly assumed that the closer to a minimum a point is, the greater the probability that falls inside its region of attraction. This implies that the regions of attraction are contiguous and surround the minima. This is not true for all local search procedures and hence this assumption influences the local search choice. For example widely used methods such as Newton or quasi Newton, employing either a line search or a trust region strategy, create disjoint regions of attraction. Hence these methods have to be modified so that their regions of attraction are contiguous, resembling those of a descent method with an infinitesimal step. In Fig. 8.3 we connect start-points (marked by +) to the minimum they arrive via a local search. This is a desirable local search since its regions of attraction are contiguous. Start points are attracted towards the closeby minima.

In this work we apply the BFGS method with a modified line search. This modification creates contiguous regions of attraction ensuring a strictly descent path [133]

We present the associated algorithm bellow:

Modified Line Search Algorithm

Input:

$k = 0, B_k = I, \epsilon > 0$

Step 1 (**Calculate descent direction**):

$$p_k = -B_k^{-1} \nabla f(x_k)$$

If $||\nabla f(x)|| > \epsilon$ Then

$p_k = \frac{p_k}{||\nabla f(x_k)||}$

End if

Step 2 (**Line search**):

$min_a(f(x_k + \alpha p_k))$, yielding a_k

Step 3 (**Next iterate**):

$x_{k+1} = x_k + \alpha_k p_k$

Step 4 (**Update approximation**):

$\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

$\delta_k = x_{k+1} - x_k$

$B_{k+1} = bfgs\_update(B_k, \gamma_k, \delta_k)$

Step 5 (**Termination Control**):

If termination conditions are met stop, Else set $k \leftarrow k + 1$ and repeat from Step 1.

To illustrate the behaviour of this normalization at Step 1 of the line search we provide figs. 8.2(a)-8.2(d). The unique minimum appearing int fig. 8.2(d) is the first minimum in fig. 8.2(b). Note that in fig. 8.2(c) the line search ends up to the nearest minimum while that of fig. 8.2(a) in a different minimum further apart.

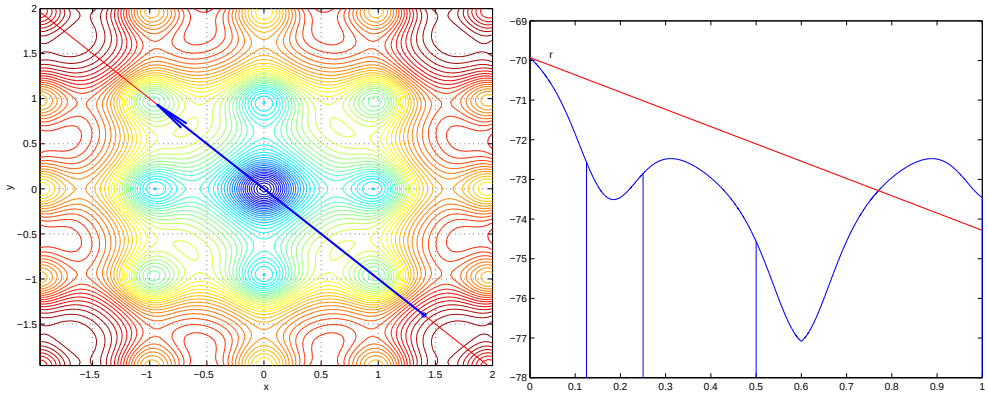
In fig. 8.4 we connect start-points (marked by +) to the minimum the arrive via a different local search. This illustrates an undesirable local search since its regions of attraction are disjoint. Start points are attracted towards distant minima.

8.3.4 Asymptotic guaranty

The probability that minimizer y is found with one trial is given by:

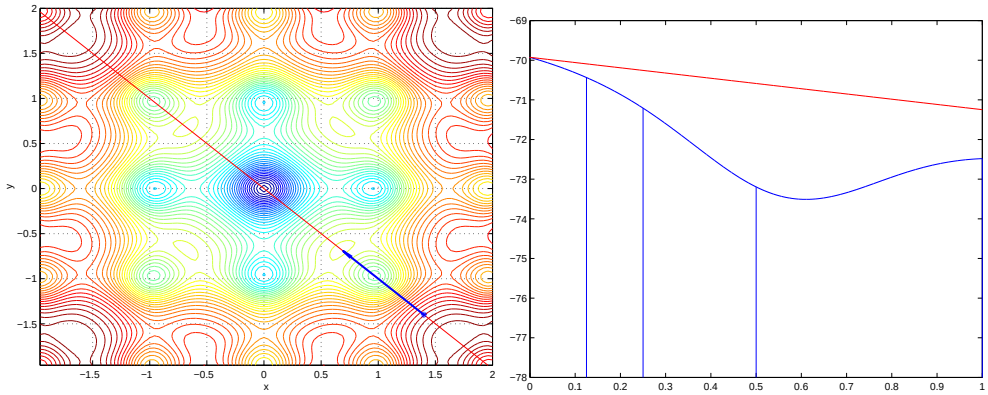
$$p_y^{(i)} = \int_{x \in A(y)} p_{LS}^{(i)}(x) \frac{dx}{|S|} \tag{8.9}$$

where $1/|S|$ is the pdf of the uniform distribution and $p_{LS}^{(i)}(x)$ is the local search probability at x. The superscript i denotes the state of the process, i.e. the number of minima



(a) Ackley's function contour plot and search step with $a_0 = 1$.

(b) Function profile along with the backtracking points



(c) Same as (a) but using $a_0 = \frac{1}{||\nabla f||}$

(d) Same as (b) for the normalized direction

Figure 8.2: Illustration of the modified line search

discovered so far, the number of times each minimizer is found, the *MAR's* etc. The probability that after k trials y is not found is then given by:

$$\pi_y^{(k)} = \prod_{i=1}^{k}(1 - p_y^{(i)}) \leq \left(1 - \min_i\{p_y^{(i)}\}\right)^k \tag{8.10}$$

From the definition of $p_y^{(i)}$ in eq. (8.9), we have:

$$p_y^{(i)} = \int_{x \in A_1(y)} p_{LS}^{(i)}(x)\frac{dx}{|S|} + \int_{x \in A_2(y)} p_{LS}^{(i)}(x)\frac{dx}{|S|} \tag{8.11}$$

where

$$\begin{aligned} A_1(y) &= \{x \in A(y); (y_c - x)^T \nabla f(x) \leq 0\} \\ A_2(y) &= \{x \in A(y); (y_c - x)^T \nabla f(x) > 0\} \end{aligned}$$

and $y_c = y_c(x)$, is the closest to x discovered minimizer.

If y is not found yet (and hence $y_c \neq y$), then $A_2(y) \neq \emptyset$ and hence $|A_2(y)| \neq 0$. Note that

$$\forall x \in A_2(y), \quad p_{LS}^{(i)}(x) = 1$$

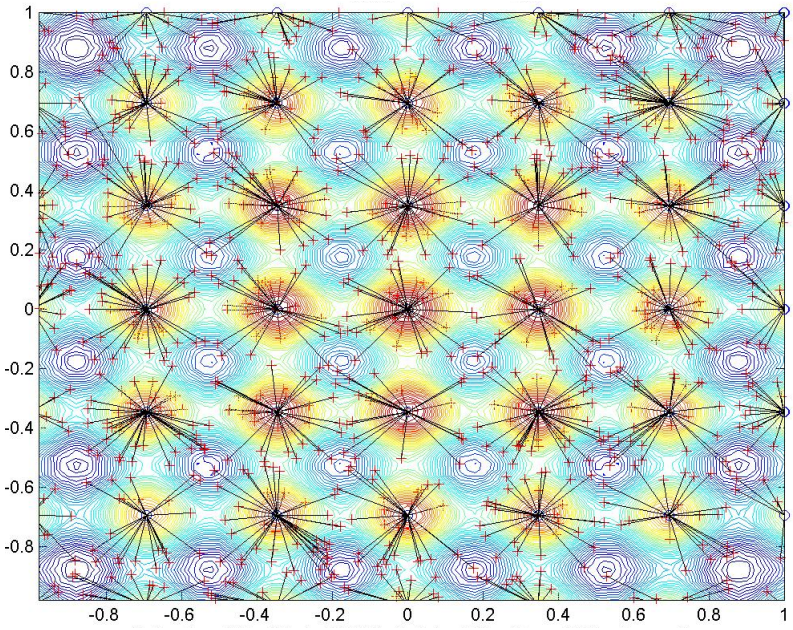


Figure 8.3: A suitable local search, with contiguous regions of attraction

and hence from eq. (8.11)

$$p_y^{(i)} \geq \frac{|A_2(y)|}{|S|} > 0, \forall i = 1, 2, \cdots, k$$

At the limit as $k \to \infty$ we deduce from above and eq. (8.10) that $\pi_y^{(k)} \to 0$, i.e. asymptotically all minimizers will be found.

8.3.5 A model for $\phi(z, l)$

Many models may be constructed with the desired properties described in (8.8). We propose one that is simple to visualize and easy to implement.

$$\phi(z, l) = ze^{-l^2(z-1)^2}, \ \forall z \in (0, 1) \tag{8.12}$$

A graphical representation is depicted in Fig.(8.5).

8.3.6 The ADAPT Algorithm

The proposed algorithm, in summary, is presented below:

ADAPT Algorithm

Input:

The input function $f : R^n \longrightarrow R$ The search domain $S \subseteq R^n$ A local search procedure $\mathcal{L}(x)$ having the properties described in Section 8.3.3.

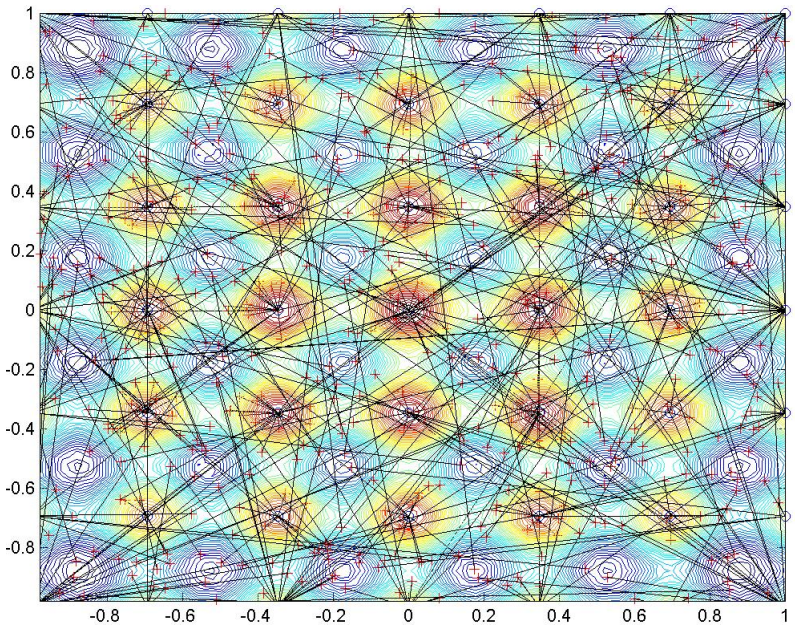


Figure 8.4: An improper local search, with disjoint regions of attraction

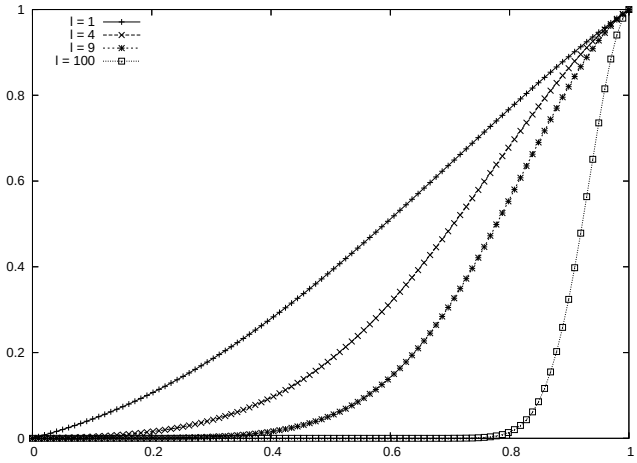


Figure 8.5: Model plots for several l values

Initialize: Set k=1

Sample $x \in S$

$y_k = \mathcal{L}(x)$

$r_k = ||x - y_k||$, $n_k = 1$

Termination Control: If a stopping rule applies, STOP.

Sample: Sample $x \in S$

Main step: i $= \underset{j=1,\ldots,k}{argmin} ||x - y_j||$

$d = ||x - y_i||$

If $(d < r_i)$ Then

If $(\nabla f(x)^T (y_i - x) < 0)$ Then

$z = \frac{||y_i - x||}{r_i}$

$p = \phi(z, n_i) \left[1 + \frac{(y_i - x)^T \nabla f(x)}{||(y_i - x)^T \nabla f(x)||} \right]$

Else

$p = 1.0$

Endif

Else

$p = 1.0$

Endif

Let ξ be a uniform random in $[0, 1]$

If $(\xi < p)$ Then

$y = \mathcal{L}(x)$

If (y is new minimum) Then

$k = k + 1$, $r_k = ||x - y_k||$, $n_k = 1$

Else { *We discovered the l-th local minimum* }

$r_l = \max(r_l, ||x - y_l||)$, $n_l = n_l + 1$

Endif

Else { *Assuming that x belongs in the region of attraction of the i-th minimum* }

$r_i = \max(r_i, ||x - y_i||)$, $n_i = n_i + 1$

Endif

Iterate: Go back to the Termination Control step.

8.4 Experiments and Comparison

The method has been tested on a number of test problems that are listed in Appendix A. These test functions have been used in the past by many authors and hence they constitute a convenient platform for comparison. We count for every problem the number of local searches, the number of function and gradient evaluations and we report averages on thirty experiments performed with different random number sequences. We also count the number of minima found. All experiments used the "*Double-Box*" stopping rule [85], with the suggested compromise factor (0.5). The local search used by ADAPT is a modification of BFGS so that the resulting regions of attraction have the properties described in Section 8.3.3. A comparison is made with the standard *"Multistart"* with the *"Topological Multilevel Single Linkage"* (TML) method [**?**] and with *MinFinder* [151]. All of the above methods use as a local minimizer subroutine `TOLMIN` due to M.J.D. Powell [127]. We coded Multistart, while the codes for *TML* and *MinFinder* used, were obtained from the corresponding authors with the default parameters. Observing the results listed in Table 8.1 we note that the performance of the new method (*ADAPT*) is overall superior. *MinFinder* has similar performance on functions M0, Borne, Shubert(N=5, 10) while it has an edge with functions having a periodicity in their contour plots like Holder, Levy No3, Rastrigin(N=2), and Shubert(N=2).

8.5 A parallel scheme

A sample Master-Slave parallel implementation is displayed below. The Master CPU creates candidate start points. The Slave CPUs perform local searches. Note, that since our method uses one point per iteration, each search is independent, enabling so maximum utilization of the Slave CPUs. On the other hand, most clustering methods use a collection of points, as for example in [133, 134, 151], that in turn create dependencies in the application of the local searches, a fact that makes the parallelization less profitable.

Definitions:

- M-list: A list that holds the minimizers (managed by the Master CPU)
- S-list: A list of possible starting points (managed by the Master CPU)

Table 8.1: Adapt results using uniform random distribution

Function	TML				Multistart				MinFinder				Adapt			
	Min.	FC	GC	LS	Min.	FC	GC	L.S	Min.	FC	GC	L.S	Min.	FC	GC	L.S
Ackley	121	10259	14457	1207	121	23281	36543	2054	121	7510	11926	208	121	7340	4600	539
Bird	158.5	84798	103889	2507	141.5	212196	150529	3737	172.8	122639	145460	1832	171.7	56008	55296	1468
Bohachefsky	25	139190	125684	2369	25	241501	187175	2547	18.7	25907	32243	538	24.3	18332	23112	215
Giunta	196	104812	16606	719	196	45688	67311	1212	196	18753	20972	791	196	10211	17821	771
Grienwank	527.2	1883423	1461617	39090	526.4	1912452	1892111	38727	529	1133908	1284982	30577	528.5	231123	278112	15733
Guillin Hills	25	81153	69847	2451	24.9	87411	76563	2617	24.8	22901	23570	820	24.7	17751	31811	691
Holder	85	28749	23346	622	85	69038	34468	988	85	8289	8977	261	85	16788	16461	352
Langermann	257	129521	124360	3566	260	185669	111478	4169	270	503470	500675	19123	270	80578	80386	2479
Levy No3	527	170541	171643	6999	527	494578	277868	8909	527	59830	91479	2320	527	146574	179502	5481
Levy No5	508	173026	183092	5011	508	365258	175718	6783	508	81037	160683	2733	508	84152	83462	2644
Liang	224.6	90506	51538	2464	233.1	180419	79899	3161	236	637784	676941	22607	235.8	73215	50569	2340
Piccionni	43	58042	42536	1475	42.9	74125	72918	2090	43	33333	36238	1222	43	48123	45647	987
Rastrigin	49	11340	14812	741	49	22233	17063	1705	49	1730	2833	85	49	17810	7481	136
Voglis	60.8	16938	1888	944	60.5	35408	2505	2304	61	21684	23126	694	61	16932	1267	437
Schaffer	93.7	48401	23295	865	94.8	56722	73922	1811	94.5	22370	24682	876	94.7	18922	16779	702
Shubert	399.6	890899	2594	2297	398.2	1062260	10732	10475	400	16551	36065	665	400	193211	10780	1439
M0	65.1	53817	35311	1654	64.5	85266	87221	2741	64	14667	16005	799	65.7	16659	17033	1023
M3	25.8	30601	20295	1507	25.8	47188	33872	2184	23.7	8914	11285	790	25.6	8752	17168	711
Borne	595.6	1090974	322968	11324	593.4	3821280	3343620	15922	598.2	1916295	2138766	68865	598.2	1880314	2626185	70161
Rast(N=5)[a]	243	131646	36084	662	243	399909	111467	2011	243	59298	64349	1022	243	30350	37634	1214
Griew(N=5)[b]	160.1	1859878	1619266	27717	159.6	2154055	2023821	32101	170	2074573	2193769	32710	169.8	1833628	2052681	34911
Griew(N=10) [c]	11.1	86815	78843	1277	11.4	145552	125187	2141	10.4	85454	85209	1270	12.7	76221	121176	1782
Shub(N=5)[d]	32	12221	15622	508	32	17881	19822	811	32	6136	6520	158	32	7022	8112	205
Shub(N=10)[e]	1021.2	1779357	220435	3977	1002.1	1866619	1973621	33230	1024	563927	565624	10302	1024	406503	526229	12853

[a] 243 minima in $[-0.5, 0.5]^5$
[b] 171 minima in $[-5, 5]^5$
[c] 13 minima in $[-3, 3]^1 0$
[d] 32 minima in $[-1, 1]^5$
[e] 1024 minima in $[-1, 1]^1 0$

- L-list: There is one such list for every Slave CPU. Each contains the minimizers discovered by the corresponding CPU.

Master CPU:

1. Check if a stopping rule applies. If so terminate.
2. Take in account the updated minimizers list (M-list).
3. Create candidate start points and add them to the starting list (S-list) and assign to each one a zero flag.

Slave CPUs:

1. If no zero flag start-points exist in the S-list, wait.
2. Pick from the S-list a start–point with zero flag, change its flag to one, and apply a local search.
3. Add the minimizer to a temporary local minimizer list (L-list).

Updater CPU:

1. Pick a minimizer from the L-list and check if it is a new minimizer.
2. If so, add it to the S-list.

8.6 Conclusions and further Work

The adaptive character of the method enables a reasonably accurate estimate of the probability that a point belongs to a region of attraction. This in turn, on one hand saves a large fraction of local search applications, and on the other hand prevents the systematic overlook of regions of attraction, reducing therefore the risk of loosing minima. The method is robust and efficient as has been deduced from the results of the computational experiments. Most of the stochastic global optimization approaches use a population of points to proceed and thus the population size is an additional parameter that affects the performance of the method. The present work in contrast, uses a single point per iteration without any adjustable parameters. This feature adds another (obvious) advantage in the case where the parallel implementation is of interest.

A parallel algorithm that would benefit from a cluster of tightly coupled processors or from a parallel shared memory system would be significant development. Such systems are nowadays widely available and offer the possibility of solving harder problems. Work in this direction is underway.

Other models for the probability, such as adaptively grown Gaussian mixtures, may be considered and some early, preliminary results are promising.

Chapter 9

Sampling from a Sum of Normal Distributions. An application to Global Optimization

9.1 Introduction

In this chapter we will propose a novel method for selecting candidate starting points for stochastic two-phase algorithms, that will take into account previous local searches. The information revealed from the local search forms a normal distribution around the most recently found local minimum. This is a direct way to implement General Algorithm 2, presented in introduction.

9.2 Global Optimization using Normal Distributions

Before we present our sampling methodology we must make clear that the proposed algorithm can be used in addition to all sampled based global optimization algorithm that employ local searches.

In the original methods a sample point x is selected using a uniform random distribution, and the global optimization strategy should decide whether or not to start a local search from it. We propose the usage of multivariate normal distribution, centered at local minima. More specifically after a local minimum y^* is retrieved, we assign to it a probability distribution function F that is defined as:

$$F = \frac{1}{\sqrt{2\pi}} \frac{1}{|\Sigma|} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{9.1}$$

where μ = mean value and Σ = covariance matrix of the distribution. We used the normal distribution to model the probability distribution, because it is parameterizable and has strong local properties.

Given N points $x_1, x_2, \ldots x_N$ we can calculate the mean value μ and the covariance matrix Σ using:

$$\mu = E(X) = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{9.2}$$

$$\Sigma = E\left((X-\mu)(X-\mu)^T\right) = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^T \tag{9.3}$$

The algorithm for creating the proposed sampling distribution is given bellow:

Algorithm 9.24 Global optimization using sum of Normal Distributions

- Initialization:
  From a uniform random starting point, apply the local optimization algorithm retrieve minimum y^* and calculate μ_i and Σ_i from Eq 9.2 and Eq 9.3 from the specific minimum $i = 1$.

- In every iteration:

  1 *Sampling*: Get a uniformly random point x and select it for starting point if for a random $\xi \in (0,1)$ the inequality

  $$\xi \times max_x(F(x)) > F(x)$$

  holds.

  $$F(t) = \sum_{i=1}^{Nlocal} N(t, \mu_i, \Sigma_i)$$

  2 *Update:* For every local minimum that is retrieved update:

  i The approximations μ_i Σ_i from every minimum y_i

  ii The quantity ρ_i, which counts how many times the i-th minimum is found.

  iii The position of the minimum and its function value

  3 *Termination*: Apply a termination criterion

9.3 Sampling

The sampling method chosen for a global optimization task is considered very important for the overall performance. Their application ensures the coverage of all solution space and hence the recovery of the global minimum. In the global optimization bibliography several sampling methods have been proposed such as:

1. Sampling from uniform distribution
2. Sampling from normal distribution (eg. Simulated Annealing)
3. Sampling from Quasi-Random uniform distributions
4. Importance Sampling
5. Rejection Sampling

In uniform distribution we can derive an analytic expression and hence for $X, U, V \in U(0,1)$ we can have::

- Normal distribution $o(0,1)$: $o = \sqrt{-2ln(U)}\cos(2\pi V)$
- Exponential distribution with parameter λ: $Y = -\frac{ln(X)}{\lambda}$
- β-distribution with parameters 1 and ν: $Y = 1 - X^{1/\nu}$

Quasi-random distribution are created using sequential algorithms that take into account previously selected points and after an infinite number of iterations they approximate the uniform distribution. Some well known quasi-random sequences are the Halton sequence, the Sobol sequence and the Nierderreiter sequence. These sequences posses an important property (specially for low dimensioned problems) that they cover uniformly the search space . This is depicted in Figure 9.1.

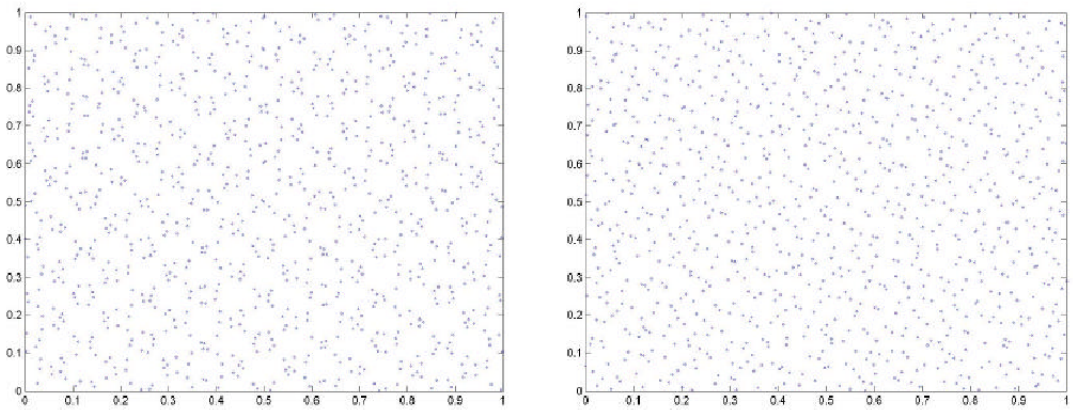
We can deduce from Figure 9.1 that uniform distribution leaves randomly "uncovered" areas in the search domain, whereas the quasi-random distributions mange to cover the search space uniformly.

Finally, rejection sampling is used to sample a point from complex distributions $F(x)$. This technique uses an auxiliary function $G(x)$ for which $F(x) < MG(x)$ holds, where $M > 1$. The rejection sampling algorithm is presented bellow:

Algorithm 9.25 Rejection Sampling

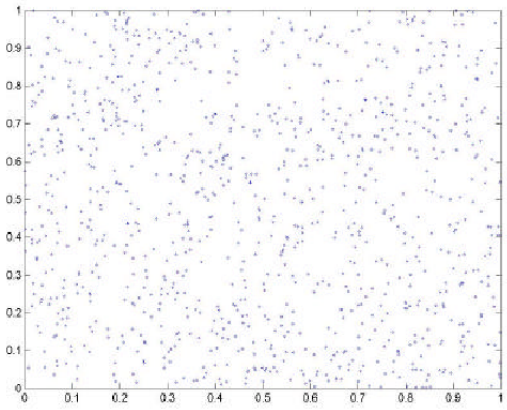
* Sample x from $g(x)$ and u from $U(0,1)$
* Check whether or not $u < \frac{f(x)}{Mg(x)}$.
    - o If this holds, accept x as a realization of $f(x)$;
    - o if not, reject the value of x and repeat the sampling step.

In our methodology we set $F(x) = \sum_{i=1}^{Nlocal} \frac{\rho_i}{\sum_j \rho_j} N(x)$, $g(x) = U(0,1)$ and $M = max_x N(x)$. Alternatively one can perform the an inverse rejection sampling (if we do not want to sample from the distribution $F(x)$) by reversing the inequality from the second step, that is $u > \frac{f(x)}{Mg(x)}$



(a) Sobol sequence

(b) Niederreiter sequence



(c) Uniform sequence

Figure 9.1: Sampled points in two dimensional search space quasi-random and uniform sequences

Let us illustrate the main idea behind our sampling technique. Assume that in the process of the global optimization algorithm we have retrieved 3 local minima and we have just sampled x from a uniform distribution in $[a, b]$. For every local minimum we have created a normal distribution $N(x, \mu_i, \Sigma_i)$ according to Algorithm 9.24. Algorithm 9.24 selects a starting point x when $\xi \times max(F(x)) > F(x)$ for $\xi \in U(0, 1)$. The sampling algorithm is described in Algorihm 9.26.

In Figure 9.2we present three different normal distributions centered at three aforementioned local minima and their weighted sum. At the sampled point x we compute the value of the weighted sum of normals F and we compare it to the global maximum value of the weighted sum. The greatest the difference, the largest the probability to actually allow x to become a starting point for local search. It can be easily deduced that if x is sampled close to an already reached minimum, the probability to accept it would be very small. In Figure 9.2 the probability to accept a starting point at $x = -4$ is almost one, whereas to accept a starting point at $x = 0$ is almost zero.

Algorithm 9.26 Inverse Rejection Sampling

- repeat the following until a point is accepted
  - Get x from $U([a,b]^n)$ where n is the problem's dimension.
  - Sample x_i from $U(0,1)$
  - $\tilde{F} \leftarrow 0$, $maxF \leftarrow 0$
  - for every local minimum retrieved
    - $\tilde{F} \leftarrow \tilde{F} + \frac{rho_i}{\sum_j \rho_j} N(x, \mu_i, \Sigma_i)$
    - $maxF \leftarrow maxF + \frac{rho_i}{\sum_j \rho_j} N(\mu_i, \mu_i, \Sigma_i)$
  - If $\xi \times maxF > \tilde{F}$ then accept x as starting point

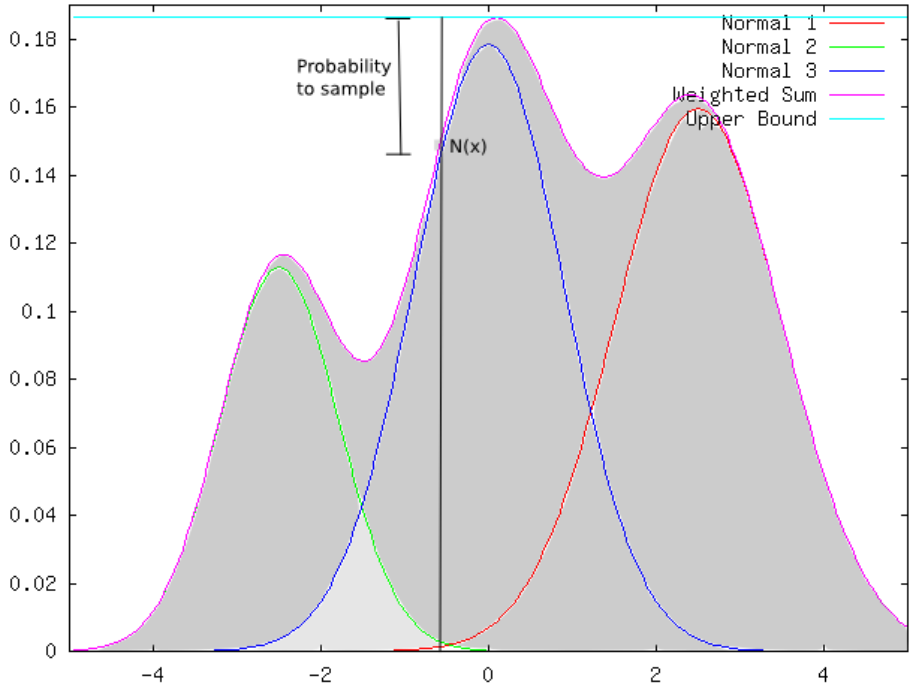
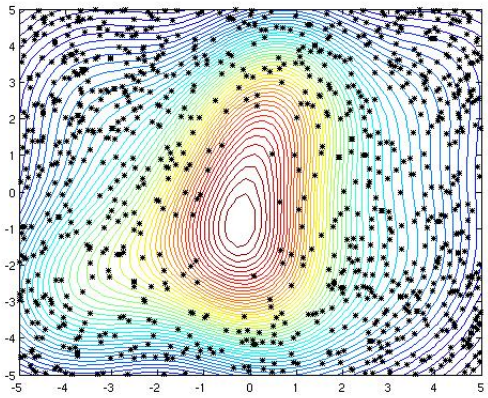


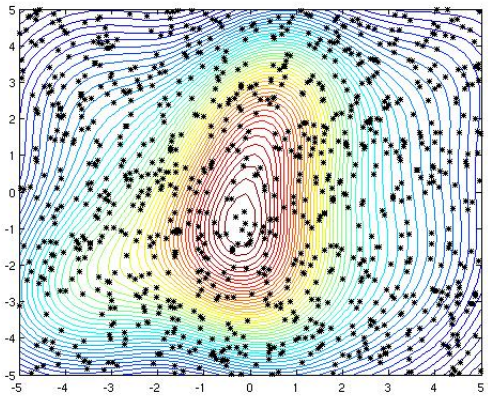
Figure 9.2: Selecting a sampled point

To continue the illustration of our sampling method, we now consider two-dimensional examples, specifically the six-hump-back and the Rastrigin function.

In Figures 9.3 and 9.4 we used 200 uniformly random starting point to create the normal distributions around local minima. We then sampled 1000 points using the proposed sampling methodology and uniform distribution. Figures 9.3(a) and 9.4(a) present the selected points from our methodology whereas Figures 9.3(b) and 9.4(b) display points from uniform distribution. We can notice that the proposed sum-of-normals distribution, avoids starting points near the basin of attraction of an already recovered minimum, since



(a) Six Hump sum-of-normals distribution



(b) Six Hump uniform distribution

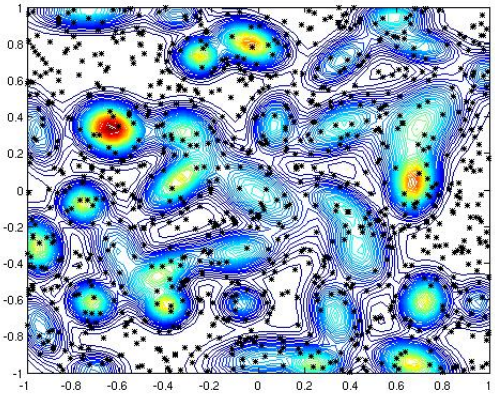
Figure 9.3: Sampling around a minimum in Six-Hump-Camel function using the proposed and uniform distribution

the probability of *not sampling* a point near a local minimum is hight. Notice that eventually the sampled points from our distribution will fill up the space around local minima and even reveal the shape of the regions of attraction.

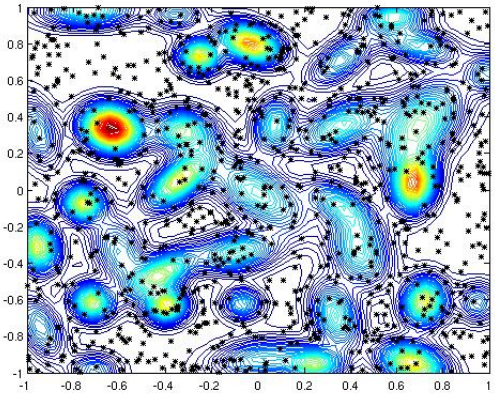
9.4 Online Estimation of Normal Distribution parameters

The maximum likehood update for the parameters μ_i and Σ_i is performed during execution of our sampling and minimization proposal. For every minimum we calculate and update the following quantities:

- $\mu_i = \mu_{i-1} + \alpha_i(x_i - \mu_{i-1})$
- $\Sigma_i = \Sigma_{i-1} + \alpha_i(x_i - \mu_i)(x_i - \mu_i)^T$, where $\alpha_i \in (0,1), \mu_0 = (0,0,\ldots,0)^T, \Sigma_0 = \alpha I_n$



(a) *Rastrigin sum-of-normals distribution*



(b) *Rastrigin uniform distribution*

Figure 9.4: Sampling around a minimum in Rastrigin function using the proposed and uniform distribution

The covariance matrix Σ_i is updated through Rank-1 updates (quantity $\alpha_i(x_i-\mu_i)(x_i-\mu_i)^T$). We can therefore use Cholesky decomposition $\Sigma_i = L_i L_i^T$ and store factor L_i (lower triangular) for every minimum. In this way we achieve update in $O(n^2)$ time and not in $O(n^3)$ if we used the original matrix Σ_i. The determinant of Σ_i can be calculated straightforwardly by using the Cholesky factor as:

- $det(\Sigma) = \prod_{i=1}^{N} L_{ii}^2$

Also Cholesky factors can be used for the calculation of the exponential part of the distribution as follows:

- $(x-\mu)^T\Sigma^{-1}(x-\mu) = \left(L^{-1}(x-\mu)\right)^T \left(L^{-1}(x-\mu)\right)$

where

$$L^{-1}(x-\mu) = y \Rightarrow Ly = x-\mu$$

. Hence all the expensive exponential calculation is reduced in solving a lower triangular linear system.

The quantity α_i can be regarded as learning factor for the normal distribution around the minimum. It can be a constant small number or a variable quantity as same minimum is found consecutively. In the second case one possible formulation could be:

$$\alpha_i = \frac{1}{\rho_i}$$

where ρ_i counts the times the $i-$th minimum is recovered. Following this strategy suggests that, the position of the first sampled points, that lead to the $i-$th minimum, will play the most important role in defining the shape of the normal distribution. In the case of constant learning factor we chose a small value of order 10^{-4}. This constant however is not suitable for every objective function.

Initialization of μ_i can be performed in two ways:

1. Initialize using the mean among the starting point and the minimum $\mu = (x_{start} - x^*)/2$
2. Initialize using the minimum $\mu = x^*$

Initialization of Σ_i can be performed in two ways:

1. Initialize using identity matrix I_n multiplied by a small quantity a_i.
2. Initialize using the Hessian matrix (or an approximation) at the minimum. The Hessian matrix at the minimum is returned by almost all gradient based minimization algorithms.

In the first case we define the initial normal distribution to be concentrated near the minimum, and subsequent starting points that lead to the same minimum will "stretch it" accordingly. In the second case can achieve the best possible representation of the minimum 's region of attraction.

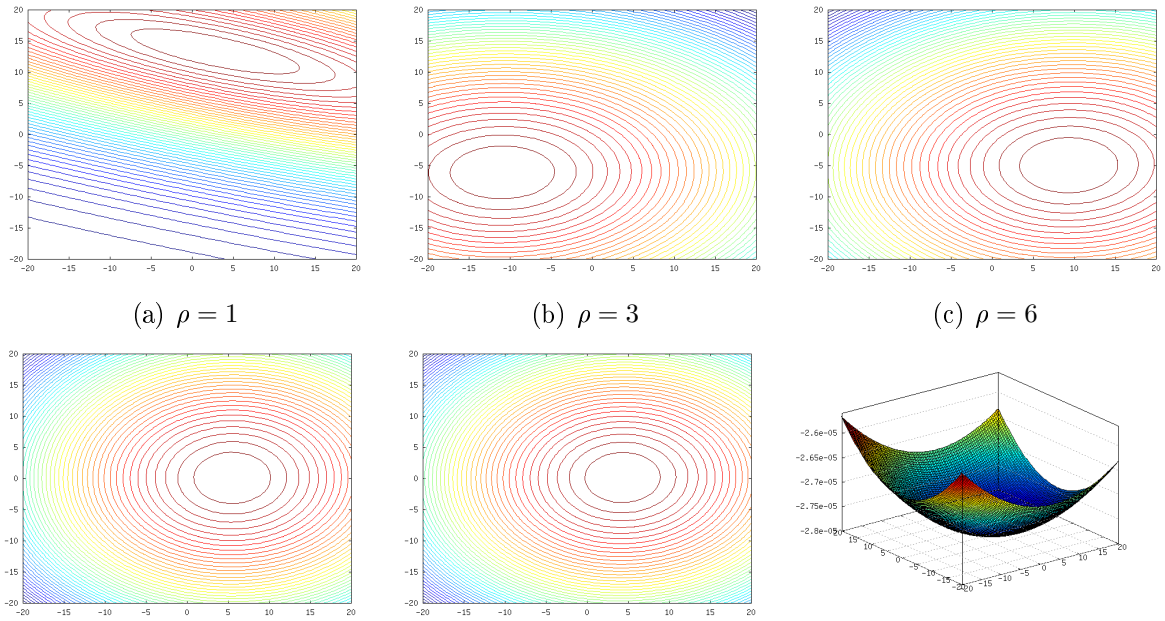
In conclusion the final distribution model we propose would be the weighted sum of the normal distributions calculated on-line for every local minimum so far. The formula will be:

$$F(x) = \sum_{i=1}^{M} \pi_i N(x; \mu_i, \Sigma_i) \tag{9.4}$$

with $\sum_{i=1}^{M} \pi_i = 1$ and $\pi_i = \frac{\rho_i}{\sum_{i=1}^{M} \rho_i}$. The quantities p_i should sum up to unity, so that $\int_{-\infty}^{\infty} N(x)dx = 1$

In Figure 9.5 we present the progress of adjusting a normal distribution around a local minimum, as it is recovered repeatedly by different starting points. After 12 updates of the covariance $Sigma_i$, and center μ_i

In Figure 9.6 we present contour plots for the final sum-of-normals distribution obtained for several test function. We can see how the proposed distributions succeeds in learning the region of attraction of local minima.



(a) $\rho = 1$ (b) $\rho = 3$ (c) $\rho = 6$

(d) $\rho = 9$ (e) $\rho = 12$ (f) Region of attraction

Figure 9.5: On-line computation of μ and Σ for a minimum at $x^* = [4, 0]^T$

9.5 Sampling as termination criterion

The proposed sampling algorithm from the sum-of-normal distribution can be used as a termination criterion. When all the minima will be recovered and all the search space will be covered by normal distributions, then accepting a point using the rejection sampling would be difficult. Recall that for $\xi \in U(0, 1)$ in order to accept a point x the inequality $\xi \times \max_x F(x) > F(x)$ must hold. The natural assumption is that when all the minima are recovered repeatedly $F(x) \to \max_x F(x)$.

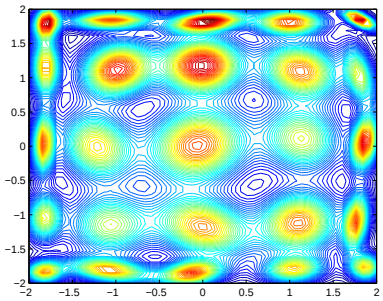
We propose two criteria for termination:

Count the rejections If the inequality $\xi \times \max_x F(x) > F(x)$ does not hold for k consecutive random $\xi \in U(0, 1)$ values then STOP.

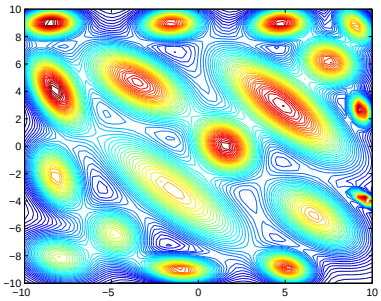
and additionally:

Measure the ratio If the the ratio $\dfrac{F(x)}{\max_x F(x)}$ for any random x is close to 1, then STOP

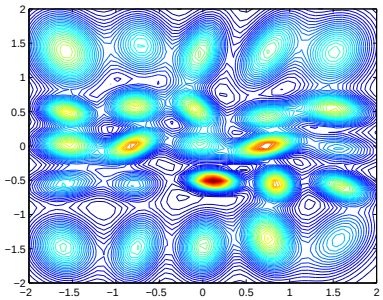
For the first criterion the number of failures must be counted each time the rejection sampling is applied, while in the second we only have to measure the ratio as it is unavoidable reaches 1.



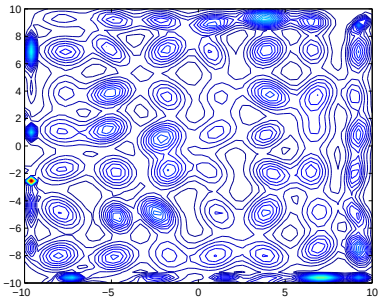
(a) Ackley's test function



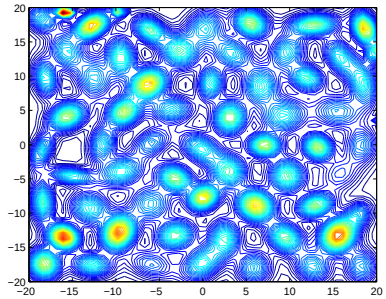
(b) Bird test function



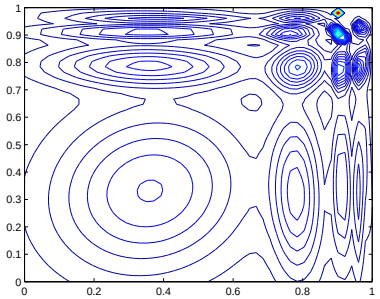
(c) Bohachefksy's test function



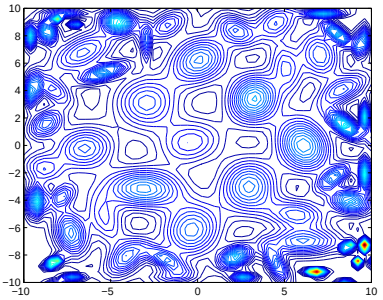
(d) Giunta's test function



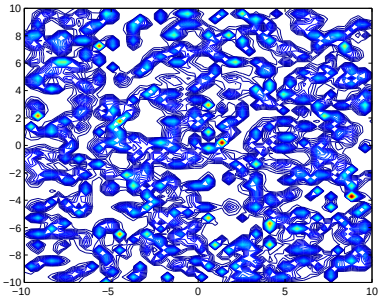
(e) Griewank's test function



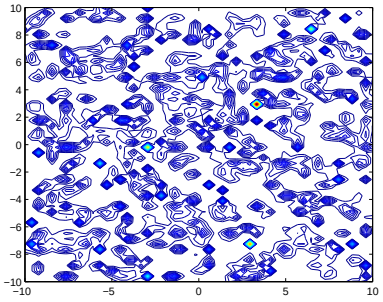
(f) Guillin Hills test function



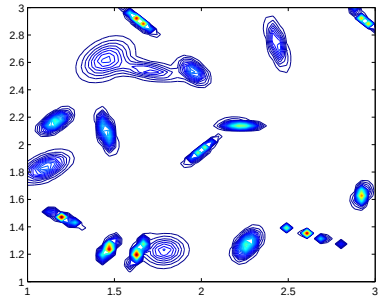
(g) Holder test function



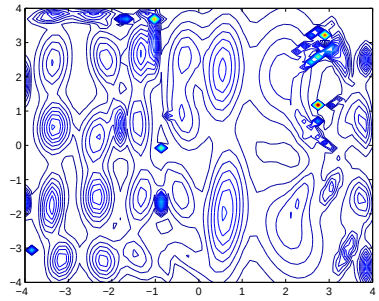
(h) Levy No 3 test function



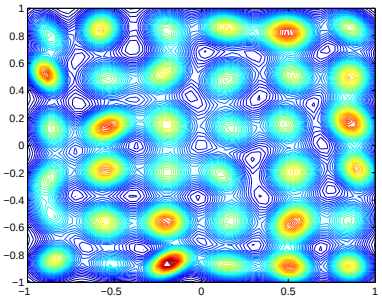
(i) Levy No 5 test function



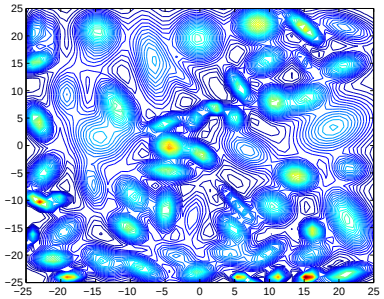
(j) Liang test function



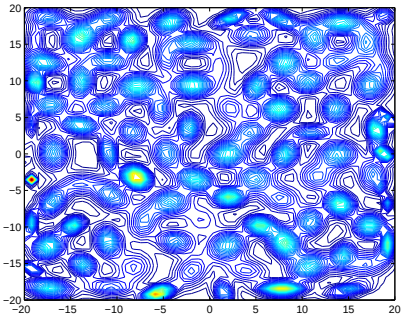
(k) Piccioni's test function



(l) Rastrigin's test function



(m) Rotating Quadratics test function



(n) Tube test function

Figure 9.6: Distribution of standard test functions

9.6 Experimental results

In this Section we will provide experimental verification of the effectiveness of the proposed sampling method. All algorithms were implemented in ANSI C and tested on a Intel Pentium 4 (2.8 GHZ) running Linux (Ubuntu 9.10) operating system. For the local optimization part of the algorithm, Merlin Optimization environment was employed.

The measure in every experiment is the overall number of local searches needed to retrieve a specific number of local minima. Our sampling technique is opposed to uniform random sampling. The testing methodology is simple. Sample a starting point, perform local search and count the minima retrieved so far. For the proposed sampling methodology, a number of M uniform samples where first drawn in order to create some initial normal distributions. Every experiment was conducted 50 times using different random seed and the mean value of local searches is presented.

We have conducted experiments using the following options:

1. Initialize Σ_i to unity.
2. Initialize Σ_i to Hessian at the minimum.

and

1. Use constant learning rate $\alpha_i = 10^{-4}$.
2. Use decreasing learning rate.

All experiments performed using the Rinnoy-Kan termination criterion presented in []. In Tables 9.1-9.4 we present comparison of our sampling method to the uniform distribution for 17 global optimization test functions. We count the number of local optimizations performed, since the local optimization method was the same in all experiments. Columns with header *Normal(N)*, mean that the first N points were sampled from the uniform distribution and normal distributions are computed around minima. After the first N samples, our sampling algorithm is employed to produce the subsequent starting points.

It can be easily deduced from the results that the proposed methodology for sampling exhibits better performance than the commonly used uniform sampling. In all results the lowest number of local searches was achieved when the first $N = 50$ starting points were sampled from the uniform distribution, and then the sum-of-normals distribution was used. During this first "*uniform*" phase normal distributions were created and updated around local minima.

Judging from Table 9.3 our methodology achieves best performance when variable learning rate is used and the Hessian matrix is used to initialize Σ_i. A comparison of all possible parameter configuration using $N = 50$ initial uniform samples is displayed in Table 9.5.

Table 9.1: Results using $\Sigma_i = 10^{-4}I$ and constant learning rate

Function (nom)	Sampling method				
	Uniform	Normal(5)	Normal(10)	Normal(50)	Normal(100)
ackley (49)	2502	2387	2353	2337	2571
giunta (36)	1371	1275	1198	1174	1243
guilin (25)	9612	3868	3300	3273	3685
levy3 (130)	17163	15799	14922	14748	11026
rast (121)	14886	16227	15751	15579	15688
griew (123)	15378	14925	14585	14392	14584
levy5 (130)	12944	11998	11666	11648	12499
rotquad (59)	12091	11696	11588	11571	12738
holder (180)	2000	2035	1970	1965	2044
bird (25)	667	563	554	553	555
piccioni (37)	1446	1536	1433	1420	1479
shekel (10)	123	112	107	106	107
m0 (152)	4805	3746	3664	3657	3978
dejong (64)	4098	3308	3189	3093	3174
lager (64)	4227	5737	5291	5271	5555
tube (45)	2118	1736	1706	1701	1756
liang (99)	9811	10591	10172	10098	10955

Table 9.2: Results using $\Sigma_i = 10^{-4}I$ and variable learning rate

Function (nom)	Sampling method				
	Uniform	Normal(5)	Normal(10)	Normal(50)	Normal(100)
ackley (49)	2502	2559	2443	2414	2599
giunta (36)	1371	1229	1220	1189	1192
guilin (25)	9612	9255	8457	8429	8779
levy3 (130)	17163	18704	17843	17709	18150
rast (121)	14886	11300	10914	14551	11161
griew (123)	15378	14199	13068	13003	13811
levy5 (130)	12944	11814	11743	11677	12191
rotquad (59)	12091	13122	12725	12622	13063
holder (180)	2000	915	746	734	895
bird (25)	667	525	412	398	407
piccioni (37)	1446	1453	1373	1363	1398
shekel (10)	123	119	131	128	134
m0 (152)	4805	5277	5130	5108	5190
dejong (64)	4098	2157	2079	2022	2183
lager (64)	4227	4230	4150	4139	4451
tube (45)	2118	2057	2056	2047	2164
liang (99)	9811	8337	8015	7887	9062

Table 9.3: Results using Σ equal to the Hessian and constant learning rate

Function (nom)	Sampling method				
	Uniform	Normal(5)	Normal(10)	Normal(50)	Normal(100)
ackley (49)	2502	2488	2447	2390	2512
giunta (36)	1371	1478	1396	1385	1455
guilin (25)	9612	8182	8015	7929	8341
levy3 (130)	17163	17578	16192	16099	16851
rast (121)	14886	15794	14038	14883	14002
griew (123)	15378	14184	13322	14139	13690
levy5 (130)	12944	11737	11574	11525	12515
rotquad (59)	12091	11172	11622	11582	10073
holder (180)	2000	1997	1962	1950	2139
bird (25)	667	596	592	587	663
piccioni (37)	1446	1486	1268	1262	1285
shekel (10)	123	111	121	104	131
m0 (152)	4805	3847	3483	3461	3608
dejong (64)	4098	2637	2607	2597	2826
lager (64)	4227	4117	4022	3991	4265
tube (45)	2118	2159	2107	2104	2151
liang (99)	9811	9407	8766	8708	9509

Table 9.4: Results using Σ equal to the Hessian and variable learning rate

Function (nom)	Sampling method				
	Uniform	Normal(5)	Normal(10)	Normal(50)	Normal(100)
ackley (49)	2502	1978	1811	1782	1790
giunta (36)	1371	1215	1073	1056	1104
guilin (25)	9612	5975	5281	5170	6212
levy3 (130)	17163	14618	14531	14434	16043
rast (121)	14886	15275	14782	14455	15081
griew (123)	15378	13925	12962	12793	13278
levy5 (130)	12944	11874	11501	10895	11073
rotquad (59)	12091	12604	12525	12412	13167
holder (180)	2000	1842	1803	1801	1856
bird (25)	667	712	671	670	716
piccioni (37)	1446	1341	1176	1136	1147
shekel (10)	123	132	126	109	115
m0 (152)	4805	4639	4593	4560	5009
dejong (64)	4098	4231	4014	3995	4332
lager (64)	4227	3581	3331	3324	3592
tube (45)	2118	1668	1534	1506	1586
liang (99)	9811	9557	9483	9162	9527

Table 9.5: Comparison of Normal(50) for all possible configurations

Function (nom)	Parameter Configuration (Normal(50))			
	Const. + Unity	Var. + Unity	Const. + Hess.	Var. + Hess.
ackley (49)	2337	2414	2390	**1782**
giunta (36)	1174	1189	1385	**1056**
guilin (25)	**3273**	8429	7929	5170
levy3 (130)	14748	17709	16099	**14434**
rast (121)	15579	14551	14883	**14455**
griew (123)	14392	**13003**	14139	12793
levy5 (130)	11648	11677	11525	**10895**
rotquad (59)	**11571**	12622	11582	12412
holder (180)	1965	**734**	1950	1801
bird (25)	553	**398**	587	670
piccioni (37)	1420	1363	1262	**1136**
shekel (10)	106	128	**104**	109
m0 (152)	3657	5108	**3461**	4560
dejong (64)	3093	**2022**	2597	3995
lager (64)	5271	4139	3991	**3324**
tube (45)	1701	2047	2104	**1506**
liang (99)	10098	**7887**	8708	9162
Sum	**102586**	**100264**	**104696**	**99260**

9.7 Conclusive remarks

We have presented a new methodology for sampling random points for the our stochastic two-phase optimization algorithm. The new sampling method is based on normal distribution and it is created "on-line" during the global optimization process.

Chapter 10

A Spectral Clustering Approach for Recovering Multiple Minima

10.1 Introduction

Clustering methods is a class of global optimization methods, which as an important part include a cluster analysis technique. The motivation for exploring clustering methods is based on the following:

(a) It is possible to obtain a sample of points in A consisting of concentration of points in the neighborhood of local minimizers of .

(b) The points in the sample can be clustered giving clusters identifying the neighborhoods of local minimizers and thus permitting local optimization methods to be applied.

(c) The procedure (a)-(b) can be implemented efficiently enough to compete with other methods proposed for global optimization.

If the procedure employing the steps (a) and (b) is successful, then starting a single local optimization from each cluster would determine the local minima and thus also the global minimum. Step (a) consists of a sampling step and a grouping step. The sampling can either be deterministic, using a grid, or random. The main idea is to cover the whole A in some uniform manner.

For grouping points around minima two strategies have been used. The first is based on the idea that retaining only points with relatively low function values these points would form groups around some of the local minima [6, 150]. The second strategy is to push each point towards a local minimum by performing a few steps of a local minimizer[150]. This latter technique with its double effect of removing high value points and creating low value points is then expected to produce groups around all local minima detected during the sampling phase. The clustering methods in the first published compared

Most of the algorithms presented can be put into the form of the following general algorithm. In this Chapter we describe our approach on recognizing the group of sampled

Algorithm 10.27 General Clustering Algorithm

GA1 *Sample points in the region of interest*:The goal for this step is to explore the whole region of interest in order to find a point leading to the global minimum. In accordance with this, all the methods use uniform sampling, and in one method (Torn) also stratified sampling may be used.

GA2 *Concentrate the sample to obtain groups around the local minima*:The goal for this step is to concentrate the points around the local minima so that they can be recognized by a clustering algorithm. Two pure strategies and some combinations of these are used.

(a) Retain a predefined portion of the lowest points.

(b) Displace the points by some steps of a local minimizer.

GA3 *Recognize these groups by the aid of a clustering method*: The goal for this step is to identify points bound for a certain minimum. *Our work proposes an alternative method for this step.*

GA4 If a stopping condition is met, stop.

GA5 Transform, sample for the next iteration, go to step 2.

points into clusters (S3). We also address the problem of concentrating the initial sampled points to obtain the groups around local minima. We propose the Spectral Clustering technique for grouping start points into clusters.

10.2 Clustering techniques

Clustering is normally applied to a given distribution of N objects, in our case points in R^d, with normally $d = n$. We assume that these points represent regions of attraction of local minima. The task of the clustering procedure is to recognize the regions of attraction by forming clusters corresponding to these regions. A central problem when applying cluster analysis techniques is the problem to choose the proper distance function D(-, o′Ωĩ), i.e., a function that defines the distance between any two given points. One possible distance function to use is the usual Euclidean distance. The results obtained with this distance function is sensitive to the scaling of the coordinates and if used the optimization problem variables should therefore be scaled so that the contribution from each variable to the distance is proportional to its importance in separating the points. We will not further discuss this problem here and if nothing else is said it will be assumed

that the Euclidean distance will be used as the distance function. A second important problem in cluster analysis is to fix some threshold distances to determine if a point is near enough to be included into a particular cluster. The choice of such thresholds will highly affect the number of resulting clusters. If chosen too small each point would form its own cluster and if chosen too large a single cluster containing all points could be obtained. We will comment on the choice of thresholds in connection with the detailed description of proposed global optimization methods.

The context in which clustering is used here makes it possible to perform auxiliary computations if needed to make the right decision in the clustering process. The function values corresponding to the points are available and contains important information. It is possible check if the direction of the gradient in a point points towards a cluster center. By using a local optimizer a point may be moved towards its cluster center. It is also possible to compute the function value in points between some given points, a possibility that can be important for correct classification.

Methods for cluster analysis may be divided into hierarchical and non-hierarchical (*partitional* methods [Anderberg 1973; Dubes and Jain 1980]. A hierarchical clustering is a nested sequence of groupings, whereas a partitional clustering is a particular partitioning of the objects.

10.2.1 Hierarchical Clustering

Hierarchical clustering can either be agglomerative or divisive. In agglomerative clustering clusters are hierarchically merged two and two starting with one point in each cluster. In divisive clustering the clusters are hierarchically divided, each into two new clusters starting with a single cluster containing all points.

Single Linkage

The simplest of all agglomerative hierarchical clustering techniques is single linkage clustering. In this technique the next two clusters to be merged are those for which the distance between the nearest points (one point belonging to one cluster and the other to the other) is the smallest. When this distance becomes larger than the threshold distance the procedure is stopped. By considering other measures of similarity between clusters other hierarchical methods are obtained. Starting with each point in a separate cluster the points at distances less than the threshold distance are linked. A cluster is recognized as a set of points linked together.

Density Linkage

Density linkage refers to a class of clustering methods using nonparametric probability density estimates [SAS 1985]. It consists of two steps:

1. A new dissimilarity measure $D*$ based on density estimates and adjacencies is computed.

2. Single linkage is performed using D*.

The $k-$th nearest neighbor method [Wong and Lane 1983] uses $k-$th nearest neighbor density estimates. Let $d_k(x_i)$ be the $k-$th nearest neighbor distance of x_i and D be the Euclidean distance function. Then

$$D^*(x_i, x_j) = \begin{cases} \frac{d_k(x_i)+d_k(x_j)}{2}, & \text{If } D(x_i, x_j) < d_k(x_i) \text{ or } d_k(x_i) \\ \infty & \text{otherwise} \end{cases} \tag{10.1}$$

Based on empirical data a possible choice for ? is 2 log2 N for N ranging from 50 to 500 points.

10.2.2 Partitional Clustering

Most of the partitional clustering techniques grow clusters starting from so called seed points.

Growing Clusters from Seed Points

One way of creating clusters, given a set of points, is to use a seed point to initiate a cluster and add nearby points as long as they are near enough. This is sometimes applied recursively so that each added point also becomes a seed point. The cluster is closed when no new point can be added.

It is obvious that a seed point ideally should be a point in the center of the underlying cluster. In cluster analysis such a seed point is normally not directly available.

In global optimization the transformed sample can be expected to contain points located around the local minima, with the minimizer somewhere in the middle of each cluster. This means that the minimizer should be a good seed point to use when forming a cluster. For each cluster one can therefore single out a point, the one with the lowest function value (i.e. an estimate of the corresponding local minimizer) which can be used as a seed point for the cluster. This means that in the global optimization setting natural seed points are easily recognizable by utilizing the function values corresponding to the given points.

Mode-seeking Algorithms

Another clustering method used for global optimization is the mode-seeking technique. In this the points to be clustered $x_1, x_2, \ldots, x_N$ are used to estimate a point density function ψ. The idea is to identify local maxima or modes in the density function and use them as cluster centers. The number of nodes in the estimate of the density function indicates the number of clusters present in the data. The clustering procedure is finished by assigning each point $x_i, i = 1, \ldots, N$ to its closest center.

The mode-seeking algorithms need a large ratio of sample size to dimensionality for accurately estimating the density function. For large N these algorithms are not feasible,

owing to the amount of computation needed to estimate the density function [Dubes and Jain 1980].

10.3 Clustering in Global Optimization

In this section, the clustering methods proposed for global optimization are described essentially as they are presented by the authors in the papers referenced.

10.3.1 Existing Algorithms

The Algorithm of Becker and Lago

This was the first work, where a clustering technique comes to aid a global optimization algorithm. The steps of the algorithm are presented in Algorithm 10.28:

Algorithm 10.28 Clustering 1: Becker and Lago Algorithm

S1 *Sample points*: Perform simple random search (random sampling) over the entire region A, giving $N*$ trial points.

S2 *Reduce sample*: Retain a predetermined number M of points with the lowest values of f (These are expected to form clusters about the minima of f in A).

S3 *Cluster*: Group the retained points into clusters by a mode-seeking algorithm.

S4 *Sort clusters*: Rate the clusters according to the lowest function value in each cluster, the best cluster being the one with the lowest f.

S5 *Construct subregions*: Construct subregions $A_1, A_2, \ldots$ containing all of the retained points of each cluster.

S6 *Treat subregions*: Treat each subregion as in steps $1-5$ starting with the best cluster, then taking the second best cluster and so on.

Clustering algorithm A mode-seeking technique with an nonparametric density function is used. The cluster centers are determined in the following way. Let r_1 be the average distance for nearest neighbor and r_2 be the average distance between the points $x_1, x_2.., x_M$ to be clustered. Take x_1 and locate the hyperspheres of radii r_1 and r_2 centered at x_1. If $x_k, k = 2, \ldots, M$ lies within the r_1 hypersphere it is averaged with x_1. This average becomes the centre of the cluster. If x_k lies between the two hyperspheres it is returned. If x_k lies outside the r_2 hypersphere, then two more hyperspheres of radii r_1 and r_2 are located with x_k as the center. This gives a number of cluster centra. The

remaining points are then assigned to the clusters according to some nearest neighbo r rule.

Törn's Algorithm The main steps of Törn' s algorithm are given in Algorithm 10.29.

Algorithm 10.29 Clustering 2: Torn's Algorithm

S1 *Sample points*: Sample N points in A.

S2 *Concentrate*: Concentrate the points about the minima.

S3 *Cluster* Identify the resulting clusters by a clustering analysis technique.

S4 *Test for stop*: If a stopping condition (e.g. only one point left in each cluster, or the same number of clusters in two consecutive clusterings) is met, go to step 6.

S5 *Reduce sample* Rate the clusters and the points in a cluster according to the lowest f value. Retain every m'th point in each cluster. Go to step 2.

S6 *Find local minima*: Determine the minima starting from the best point in each cluster.

Concentrating Points Two approaches were tried:

(a) retain a predefined number of points with the lowest f

(b) for each point perform a number of steps of a local optimizer.

Based on the outcomes of some experiments, method (b) was deemed more successful in concentrating the points and method (a) was not further considered.

Clustering Algorithm A cluster is grown about a seed point by adding all unclustered points lying in successively larger hyperspheres centred at the current seed point, as long as the point density in the volume between the successive hyperspheres remains greater than the average density in the region of reference. The radii of the successively larger hyperspheres are determined so that a single point between successive hyperspheres is enough to guarantee this limiting density. Natural seed points are used. The region of reference $S \in R^n$ is the region spanned by the points, and its volume V is given by:

$$V = \prod_{i=1}^{n'} 4\sqrt{\lambda_i}$$

where $\lambda_1 \leq \lambda_2 \leq \ldots \lambda_d$ are the roots of equation $(C - \lambda I) = 0$ where C is the covariance and I is the identity matrix. The quantity d is either n or $n+1$, with $f(x)$ as the $n+1$-th

coordinate. The value of $n' \leq d$, determined by the clustering algorithm, may disclose that the minima are contained in a lower space $(R^{n'}, n' \leq d)$ which in some cases can be valuable information about the problem.

Spircu's Algorithm

The main steps of this algorithm are presented in Algorithm 10.30:

Algorithm 10.30 Clustering 3: Spircu's Algorithm

S1 *Sample points*: Generate $N*$ points, uniformly distributed in A.

S2 *Concentrate, reduce sample*: Transform the sample by performing some steps of a local optimizer from these points and by retaining a fraction γ of the best points ($\gamma = .25$) S3.

S3 *Find modes*: Estimate the density function ψ determined by $z_1, \ldots, z_M$ and calculate the modes $m_1, \ldots, m_K$ If $K = 1$, go to step 5, otherwise go to step 4.

S4 *Reduce points, generate new points*: Generate $N' = (NoK)/2$ points, uniformly distributed in A. Modify $N* = K + N'$ and take as the current sample the modes plus the new points. Continue with step 2.

S5 *Find local minimum*: Perform a final local optimization starting from m_1.

Clustering The clustering method used is a mode-seeking technique. The explicit density function is statistically estimated as sum of M so called δ-generating sequences. It is pointed out that this gives a consistent, asymptotically normal, unbiased estimate of ψ. The modes are determined by starting a local algorithm for the estimated density function from each point $z_i, i = 1, \ldots M$. In Spircu's method only the modes are used and no assignment of points to the cluster centers or modes is performed.

The Algorithm of Boender et al

In their paper [Boender et al 1982], the authors describe their method as a stochastic method involving a combination of sampling, clustering and local search, terminating with a range of confidence intervals on the value of the global minimum.

Clustering algorithm Two methods were developed. Both methods use local information about the objective function (the Hessian in the corresponding local minimum) and rely on properties of the sampling distribution. The first method is a refinement of the clustering idea of Torn. Instead of using successively larger hyper spheres the successively larger sets are approximations of the level sets around a seed point (local minimum) giving ellipsoids.

Algorithm 10.31 Clustering 4: Boender et al Algorithm

B1 *Initialize*: Choose values for $N*$ and γ. $x*$ is the set of local minima found so far and $X^{(1)}$ is the set of sample points leading to a minimum $x* \in X*$.

B2 *Sample points*: Draw $N*$ points and add them to the sample.

B3 *Reduce, concentrate sample*: Construct the transformed sample by taking the fraction γ lowest points of the current sample and by performing a steepest descent step from each of these points.

B4 *Cluster*: Apply a clustering procedure to the transformed sample. The elements of $X*$ are first chosen as seed points followed by the elements of $X^{(1)}$. If all points of the transformed sample can be assigned to a cluster, go to step 6.

B5 *Find local minimum, cluster*: Among the points not yet clustered, let $x^{(1)}$ be the point with the lowest function value. Apply the local search procedure to $x^{(1)}$ to find a local minimum $x*$. If $x* \notin X*$, add $x*$ to $X*$ and choose it as the next seed point. If $x* \in X*$, add $x^{(1)}$ to $X^{(1)}$ and choose $x^{(1)}$ as the next seed point. Repeat step 5 until all points have been assigned to a cluster. If a new point has been added to $X*$, go to step 2.

B6 *Stop*: Determine y*, the smallest local minimum value found, and stop.

The second method is a partitional application of the single linkage method where unclustered points x are added to a cluster, initiated by a seed point either in $X*$ or $X^{(l)}$ as long as the distance

$$D(x, x') = \left[(x - x')^T H(x_i^*)(x - z')\right]^{1/2}$$

to the nearest neighbor x' in the cluster is less than a threshold distance r. The threshold distance is computed using an approximation of the probability distribution of the nearest neighbour statistic within a set of uniformly distributed points, giving

$$r = \left[\frac{\Gamma(1 + n/2)|H(x_i^*)|^{1/2}\mu(A)}{\pi^{n/2}}(1 - \alpha^{1/(N-1)})\right]^{1/n}$$

,where ? corresponds to the probability of type 1 error. In preliminary experiments this method was found to be more accurate than density clustering and was therefore chosen. A further check is made before a point is added to a cluster, it is checked that the negative gradient points in the direction of $x*$, or, if the seed point is a member of $X^{(1)}$ that the gradient points in the same direction as in the seed point. This modification proved very useful for detecting close minima.

The Algorithm of Betro and Rotondi

A Bayesian nonparametric approach to global optimization is presented in [Betro and Rotondi 1984]. The aim of the algorithm is to quickly and inexpensively find an improvement of the best sampled value hopefully sufficient to decide that, on the bases of estimating the probability $P(F(\tilde{f}) < \epsilon | \tilde{F} = \tilde{f})$, that the required accuracy $\mu\{L_j\} < e$ has been achieved, rather than to identify all relevant regions of attraction of local minima.

Algorithm 10.32 Clustering 5: Betro and Rotondi Algorithm

BR1 *Initialization*: Choose values for $N*$, M, f_0 a guess for $f*$,α and ϵ. $X*$ is the set of local minima found so far and X_s the set of starting points for these minima. Initially these sets are empty.

BR2 *Initial sampling estimate $\tilde{f}*$, test for stop*: Sample M points. Start a local search from best point in sample obtaining $\tilde{f}$. Set $N = 0$. If $P\left(F(\tilde{f}) < \epsilon | F = \tilde{f}\right) > \alpha$ then accept $\tilde{f}*$ as the final estimate and stop.

BR3 *Sample points, reduce sample*: Sample $N*$ points in A. Retain the $\gamma N*$ best. Set $N = N + N*$.

BR4 *Cluster*: A cluster is formed taking as seed point the best point x_b in the sample. Possibly existing subclusters within this cluster are identified. The best point within each cluster is retained as a possible starting point for a local search, together with the radius of the cluster.

BR5 *Find local minimum?*: Compare the information retained from the present clustering with information gathered during previous stages. If it seems possible that the region of attraction of a new local minimum has been identified start a local search from x_b and adjust $X*$ and X_s accordingly.

BR6 *Test for stop* If $P(F_f(\tilde{f}) < \epsilon | y_1, \ldots, y_N, F = \tilde{f}) > \alpha$ then accept $\tilde{f}*$ as the final estimate and stop, otherwise go to step 3.

The Algorithm of Timmer

The algorithm of Boender et al has been modified by Timmer [Timmer 1984]. Timmer considered several clustering methods. Based on experiments, a method, miilfi level single linkage, was deemed most accurate.

The Algorithm of Rotondi

Rotondi's clustering technique is based on the concept of the k-th nearest neighbour [Rotondi 1978]. In order for the problem to be solvable in a finite number of steps, it is

Algorithm 10.33 Clustering 6: Timmer's Algorithm

T1 *Initialize*: Choose values for $N*$, γ and σ. $X*$ is the set of local minima found so far.

T2 *Sample, reduce sample* Draw $N*$ points at random and transform the sample by taking the fraction γ lowest points of these $N*$ points. Set $k = \gamma N*$.

T3 *Find local minimum*: Start a local optimizer from each new point x_i except if there is a sample point x_j with $f(x_j) < f(x_i)$ and $||x_j o x_i|| < r_k$. Add new stationary points encountered during the local search to $X*$.

T4 *Test for stop*: If the expected number of minima $\tilde{w}$ exceeds the number of different minima found w by less than 0.5 stop, else sample a new point, set $k = k + 1$ and go to step 3.

assumed that there exists a positive constant ϵ, such that the distance between any two local minima exceeds ϵ. The algorithm is presented in Algorithm 10.34.

Clustering Algorithm Natural seed points are used to initiate clusters. The critical distances $r_j, j = l, \ldots, k$ are determined based on k-th nearest neighbour statistics for uniformly sampled points giving $rj = (\frac{\beta_{j,1-\alpha}}{\lambda})^{1/n}$, where $\beta_{j,1-\alpha}$ is the $(1-\alpha)$-quantile of the beta distribution $Be(j, Noj)$ and N is the sample size. The type I error under the null-hypotesis H_0 is approximately equal to α for the choice of r_j.

10.4 A new Clustering Approach for Global Optimization

We mentioned already that clustering methods have proved very successful in tackling the global optimization problem. One reason is that they make it possible to very efficiently combine global and local search.

Motivated by the strong theoretical and practical properties of the general clustering method for global optimization we present a new technique for creating clusters around minima. Our approach address the clustering problem of sampled points around minima using components form:

1. *Spectral clustering* a promising alternative that has recently emerged in a number of fields, that works in the eigenspace of a matrix derived from the points to be clustered,

2. *Global k-Means* a hierarchical clustering technique that address the problem directly at the input space.

Algorithm 10.34 Clustering 7: Rotondi's Algorithm

R1 *Sample, reduce sample*: Sample $N*$ points uniformly in A and add them to the N sampled points (initially $N = 0$), take the γN ones with the smallest function values.

R2 *Choose seed*: If all reduced sample points have been clustered, go to step 5. Else choose as seed point and possible "father" the best remaining point.

R3 *Grow cluster about seed*: Build successively larger hyperspheres $S_i i$ with radii $r_i, i = l, \ldots, k$ around the "father" as long as the hypersphere contains at least one new reduced sample point having larger function value than the "father" (Function value test). Let the last such hypersphere be S_L Assign all unclustered points in S_L passing the function value test to the cluster (and call them "sons"). Complete the cluster by letting each "son" in turn become the "father" and repeat the procedure until each "son" has become "father", then go to step 2.

R4 *Merge clusters* Let x_1 be a point rejected by the Function Value Test in cluster C_a, which becomes a seed point for a new cluster C_b. If an element x_2 of C_a is within distance $2r_1$ of x_1 and the middle point z of the segment (x_1, x_2) is such that $f(x_2) < f(z) < f(x_1)$ then the clusters C_a and C_b merge.

R5 *Find local minimum*: A local search from the seed point of a new cluster is started only if (a) the number of points in the cluster is at least two, and (b) no local search has been started from points in the cluster.

R6 *Test for stop*: If in two cycles no new minimum is found stop, otherwise go to step 1.

These two powerful clustering tools are employed for the first time in the global optimization framework. We prefer *Spectral Clustering* analysis for two reasons. Firstly, it is based on pairwise affinities between points, which in standard clustering applications are calculated using Euclidean distances. However in the optimization framework both function and gradient values at points are available. We will show that this extra information can be essential in associating or disassociating concentrated points. The second reason is that using a simple eigenvalue analysis we can calculate a surprisingly accurate estimate for the number of clusters.

On the other hand the *Global k-means* algorithm, although an expensive procedure, is classified among the best clustering techniques in the bibliography. Moreover, it can be straightforwardly modified to take into account affinity information from spectral clustering analysis. More specifically global k-means algorithm is based on successively applying the simple k-means algorithms which, in turn, is based on Euclidean distance between points and the introduction of new points that represent the cluster (means). Instead of

the Euclidian distance one can use the pairwise affinities between the concentrated sampled points, and instead of new points one can define an existing point as representative for a cluster (medoids). We attempt to get the most out of these two methodologies combining them using:

- Function and/or gradient information in addition to Euclidean distance between points
- Global k-means variation that operates on these affinities and uses existing representatives, namely global k-medoids.

Recall the general clustering algorithm presented in Algorithm 10.27. Following the same general scheme we present a first sketch of our clustering algorithm

Algorithm 10.35 The proposed method – Outline

S1 *Sample points in the region of interest*: For this step we use two alternatives:

(a) Uniform random sampling

(b) Quasi–uniform sampling (Halton sequence)

S2 *Concentrate the sample to obtain groups around the local minima*: We prefer Törn's alternative to concentrate the sample around minima, by displacing it using a fraction of the negative gradient or few steps of a local optimizer.

S3 *Recognize these groups by the aid of a clustering method*: In general out clustering method consists on two main steps (which will be thoroughly analyzed later):

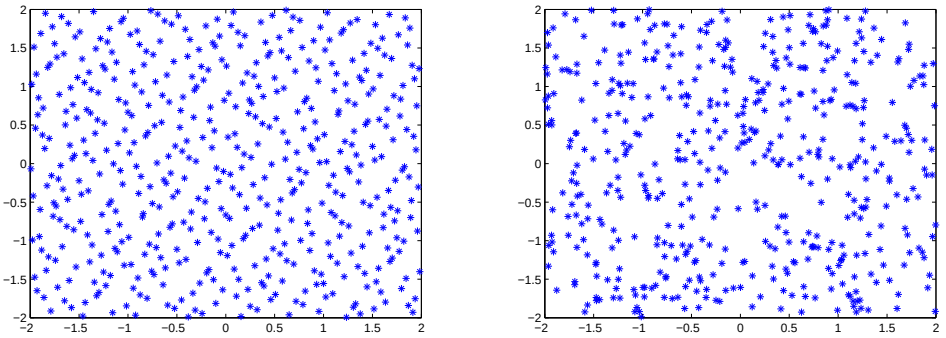
(a) Estimate the number of clusters k formed by the concentrated sampled points

(b) Apply global k-means (or a proposed variation) seeking k clusters.

S4 *Stopping condition*: Any stopping criterion from the bibliography can be used.

10.4.1 Step 1: Sampling methodology

In order to explore the whole region of interest, we chose to apply uniform sampling. However by using quasi–random sequence of number (like Halton sequence) we witnessed an improvement especially in low dimensional problems. Quasi–random sequences are sequences of n-tuples that fills n-dimensional space more uniformly than uncorrelated random points. The nature of quasi-random sequences, offers a better coverage for the search space because they are constructed so that their discrepancy would be low. The application of quasi–random sequence enables our methodology to retrieve the same number of local minima by sampling less points than uniform random sequence. This becomes



(a) Halton sequence (b) Uniform random

Figure 10.1: Sampling 500 points

more obvious for low dimensional problems. In Figure 10.1 we illustrate the difference between sampling 500 points using a Halton sequence and a uniform number generator. Notice, the "clusters" formed by the 500 uniform random numbers.

10.4.2 Step 2: Concentrate samples around minima

For the concentration step we use two alternatives. The first, and more straightforward, is to allow each sampled point to move for a fraction of the negative gradient direction.

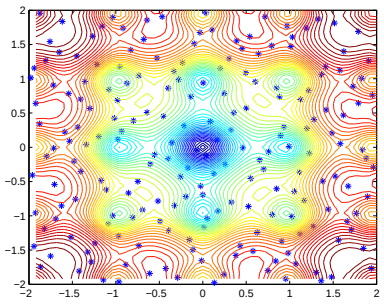
$$x_s \leftarrow x_s - \alpha \nabla f(x_s)$$

This give rise to a very important parameter α which controls the step that will be taken in this direction. One cannot find a unique value for α that would produce reasonable concentration for every objective function, and this parameter affects greatly the final outcome. On the other hand one can utilize a local optimization method and apply it for a few iterations for every sample point. The main defect for this approach is that local optimizers also produce "jumps" at the search space, meaning that a starting point may converge to a local minimizer far away, that does not belong to the same region of attraction. This problem is handled by a *local–local* optimization algorithm that is presented in the next chapter of this thesis. In Figure 10.2 we illustrate an example of poor choice of α parameter opposed to an example of good choice of α. We also present the application of a local search from each sample point[1].

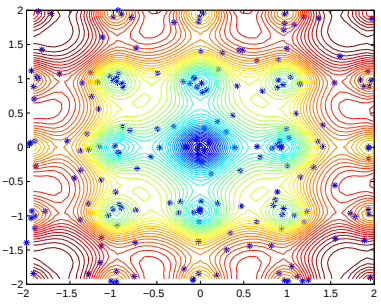
10.4.3 Step 3: Clustering

We are now ready to present the basic contribution of our work in the clustering global optimization framework. In order to solve the clustering around minima phase, we utilize spectral information of the concentrated sample with the Global k-means algorithm (and

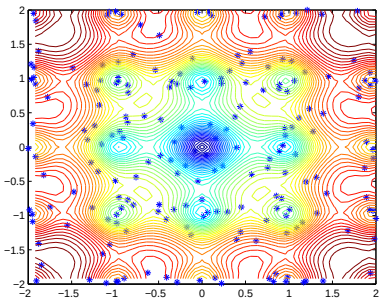
[1]The same overall cost



(a) Three iterations using $\alpha = 0.01$



(b) Three iterations using $\alpha = 0.1$



(c) Three iterations of a local search

Figure 10.2: Sampling 200 points, concentrating using a step on negative gradient

its *median* modification). The clustering procedure is rather straightforward and we can distinguish two phases:

C1 Estimate the number of clusters k formed by the concentrated sampled points.

C2 Apply global k-means (or a proposed variation) seeking k clusters.

In the first phase the main computational task is to calculate the eigenvalues off a symmetric association matrix based (a) on the positions of the reduced sample and (b) on the information we provide to associate/disassociate two samples. By analyzing the eigenvalues we can derive an extremely accurate approximation of the number of clusters (hopefully the number of minima). We can also use this association matrix as an input for the second phase. The second phase is essentially the Global k-means algorithm in his original form, utilizing Euclidean distances, and with one modification, operating on association matrix.

In the next paragraphs we will analyze the basic components of out clustering approach.

Spectral estimation of the number of clusters – The eigengap

Spectral clustering is a recently emerged clustering technique that has its origins in spectral graph partitioning. We will present the basic algorithm as it was presented in [Ng,

Jordan] although we only use the first three steps in our method. In these steps we present the calculation of the association (from now on similarity) matrix. A short listing of the estimation of the number of cluster algorithm using spectral information, is shown in Algorithm 10.36. The estimated number of cluster using the *eigengap* heuristic has

---

Algorithm 10.36 Spectral Clustering

Input: A set of points $X = x_1, x_2, \ldots, x_N, x_i \in R^d$

A small number σ

Output: The estimated number of clusters k.

1. Form the affinity matrix $A \in R^{N \times N}$ defined by $A_{ij} = exp(-||x_i - x_j||^2/2\sigma^2)$ if $i \neq j$, and $A_{ii} = 0$
2. Define D to be the diagonal matrix whose (i,i)-element is the sum of A's i-th row, and construct the matrix $L = D^{-1/2}AD^{-1/2}$
3. Calculate and sort decreasingly the eigenvalues of L. Let $e_1, e_2, \ldots, e_N$ be the sorted eigenvalues.
4. Calculate the differences $\delta_i = e_{i+1} - e_i, i = 1, \ldots, N-1$.
5. Find the maximum eigengap: $k = argmax_{k=1,\ldots,N-1}\delta_i$

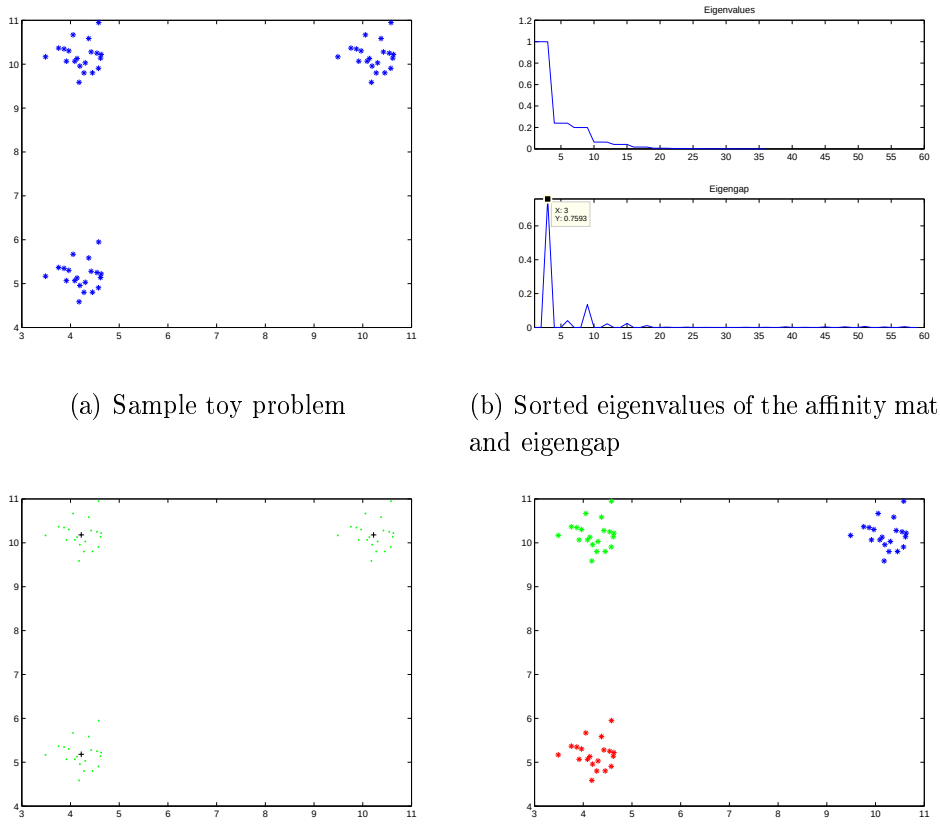
---

its origins in graph theory and especially in graph partitioning theory. There are several justifications for this procedure. The first one is based on perturbation theory, where we observe that in the ideal case of k completely disconnected clusters, the eigenvalue 0 has multiplicity k, and then there is a gap to the $(k+1)$-th eigenvalue $\lambda_{k+1} > 0$. Other explanations can be given by spectral graph theory. Here, many geometric invariants of the graph can be expressed or bounded with the help of the first eigenvalues of the graph Laplacian. In particular, the sizes of cuts are closely related to the size of the first eigenvalues.

In Figure 10.3 we present a simple application of the eigengap heuristic. The original clustering problem is presented in Figure 10.3(a), where we can clearly distinguish three clusters. After calculating the matrix L and its eigenvalues we obtain the plot of Figure 10.3(b) that shows the eigenvalues in decreasing order as well as their differences.

Calculation of σ

One of the most important parameters for the calculation of the affinity matrix is the denominator σ in expression $exp(-||x_i - x_j||^2/2\sigma^2)$. An inappropriate choice for this parameter may affect the ability to distinguish clusters and hence determine their number k. In essence this quantity identifies a distance threshold that separates clusters, hence controlling the width of the neighborhoods. Small values of σ dictate that cluster points



(a) Sample toy problem

(b) Sorted eigenvalues of the affinity matrix and eigengap

(c) Calculated cluster centers (Simple k-means algorithm)

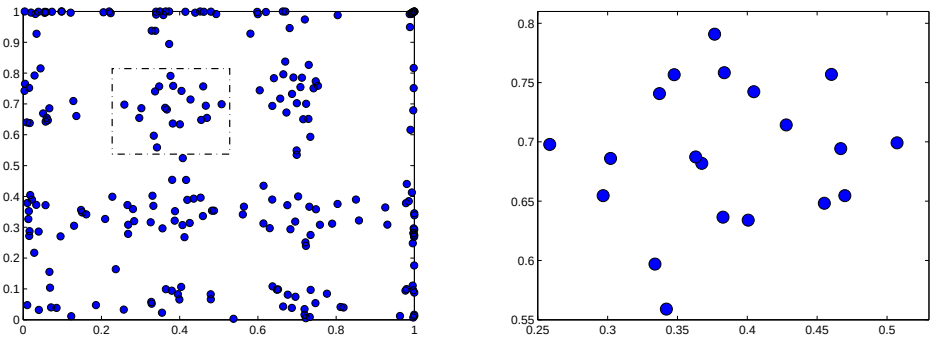
(d) Final clustering

Figure 10.3: Number of cluster estimation using spectral information, on a simple example

must be closer together than large values. Essentially, the value of σ defines the magnitude of scaling the input data. Consider the example in Figure 10.4 where the same data set is presented with two different scales (zoom levels). In Figure 10.4(a) the cluster inside the dashed rectangle is obvious. On the other hand, in Figure 10.4(b) we draw the same cluster using different scaling. In this way we can *visualize* the way the parameter σ affects the clustering algorithm.

In order to automatically select an appropriate value for σ we calculate the distances for every data point to it's k-nearest neighbors. We then average these k distance for every point to obtain the mean distance from it's k-snearest neighbors. Finally, we average the mean distance over all data points and obtain the σ. The algorithm for σ calculation is shown in Algorithm 10.37.

From the Algorithm 10.37 we introduce the value of $Inei$ as the basic parameter of our clustering approach. The final value of σ depends solely on the quantity $Inei$. In Figure 10.5 we present 200 sampled points well concentrated around the local minima of Ackley's function. We have applied our clustering method (Algorithm 10.35) on this example for $Inei = 2, 3, 4, 5$, and study the impact of this parameter on the final clustering



(a) Normal scale (b) Zoom-in one cluster

Figure 10.4: A sample dataset

Algorithm 10.37 The calculation of σ

Input: D: NxN matrix containing the distance between sampled points, N: Sample size, $Inei$: Number of neighbors for the calculation of σ

Output: σ

1. **For** i=1 to N **do**
   For j=1=1 to N **do**
   $dis(j) = D_{i,j}$
   End For
   $dis_{sort} \leftarrow sort(dis)$
   $m(i) \leftarrow \frac{1}{Inei} \sum_{k=1}^{Inei} dis_{sort}(k)$
   End For
   $\sigma = \frac{1}{N} \sum_{k=1}^{N} m(i)$

outcome. Notice that, out main interest here is the ability to obtain the number of clusters by just examining the eigenvalues of the affinity matrix. Since the clusters are well separated and many starting-sampled points have reached each minimum (approximately 8 points per local minimum), all choices of $Inei$ lead to reasonable estimates to the number of clusters (and hence minima). Choosing $Inei = 2$ results in 44 clusters (Figure 10.6), where with $Inei = 3$ the number of predicted clusters is reduced to 30(Figure 10.7). Averaging among the 4-th and 5-th nearest neighbors though, produces the correct number of clusters which is 25 (Figures 10.8 and 10.9). It is obvious that when the clusters are properly distinguished, and a modest number of sampled points are concentrated around each minimum, choices $Iner = 4$ and $Inei = 5$ produce the best results.

On the other hand, we have studied the case that the sampled points are not well concentrated around local minima. This case is shown in Figure 10.10. In this setting we

have slightly transformed the initial sample, so the clusters are not yet formed. In this case choosing $Inei = 2$ or $Inei = 3$ lead to an exact estimation of the number of clusters (Figures 10.11, 10.12), where averaging for 4 and above nearest neighbors concludes to the poor choice of one cluster (Figures 10.13, 10.14). We can expect that kind of behavior, due to the almost uniform distribution of the sampled points.

As a rule of thumb we state that when we have an a-priory knowledge for the ratio of the sampled points to number of local minima, we can use it to define a proper value for $Inei$. In most cases though a value of 3 seems adequate.

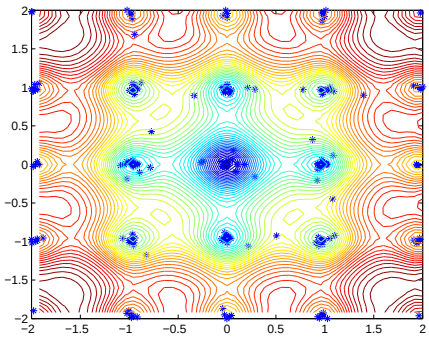
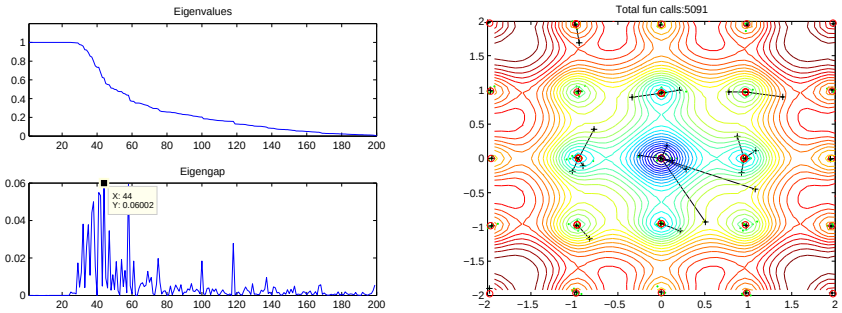


Figure 10.5: Ackley's function 200 starting points well concentrated around minima

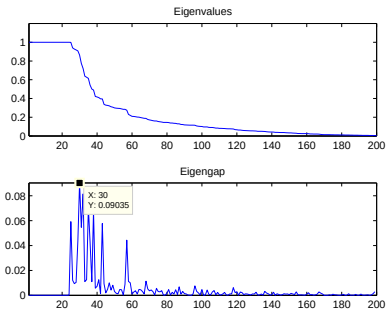


(a) Sorted eigenvalues, eigengap (b) Final clusters

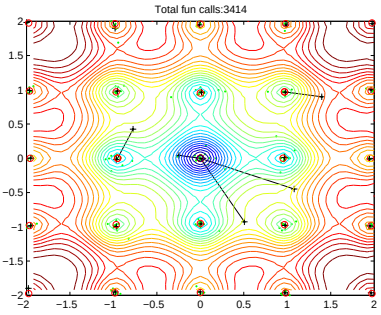
Figure 10.6: Using 2 neighbors for affinity matrix ($k = 44$)

Including gradient information

Clustering points around possible minima of a multimodal function, provides a significant source for information: the objective function itself. By this we mean that we do not just identify groups of points in n-dimensional Euclidean space, but we seek relationships between sample points, that eventually will lead to the same local minimum. Both the function values and the gradient vectors can assist in an attempt to associate or disassociate sample points in addition to their Euclidean distance.

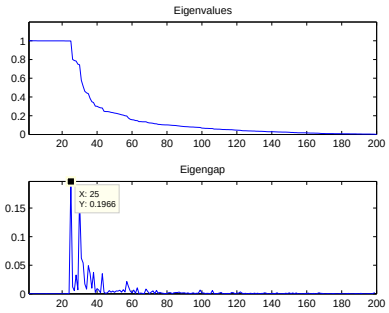


(a) Sorted eigenvalues, eigengap

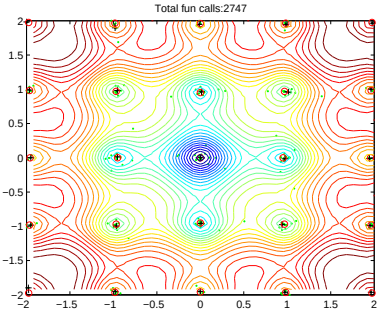


(b) Final clusters

Figure 10.7: Using 3 neighbors for affinity matrix ($k = 30$)

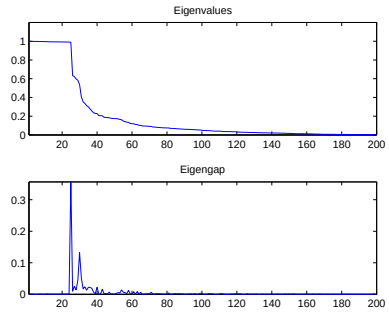


(a) Sorted eigenvalues, eigengap

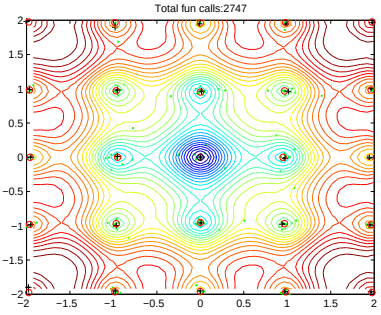


(b) Final clusters

Figure 10.8: Using 4 neighbors for affinity matrix ($k = 25$)



(a) Sorted eigenvalues, eigengap



(b) Final clusters

Figure 10.9: Using 5 neighbors for affinity matrix ($k = 25$)

Consider a local minimum and the corresponding region of attraction. Also consider two points inside the region of attraction and near the specific minimum. Their negative gradient directions, will point to that minimum and by following it they will both approach the local minimum. On the other hand consider two that points belong to different regions of attraction but their Euclidean distance is small. In this case their negative gradient directions will draw that points apart, each one near the corresponding minimum. This is depicted in Figure 10.15, where the points x_1 and x_2 belong to the same region of

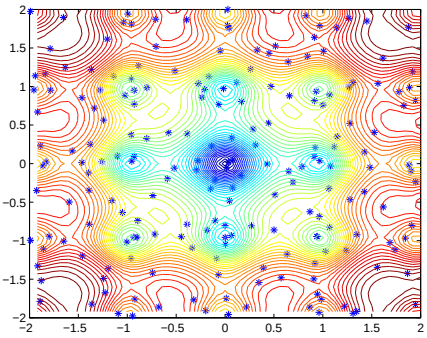
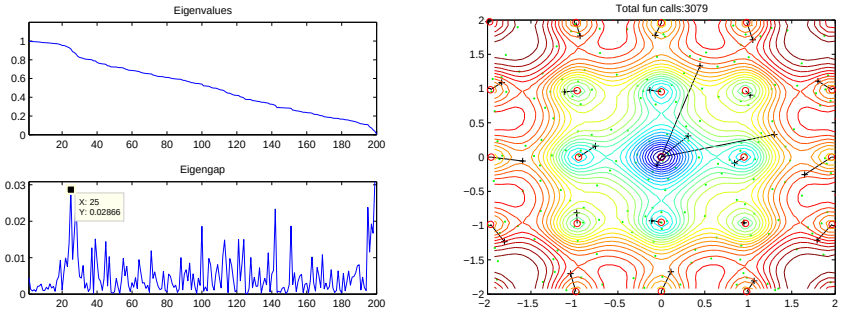
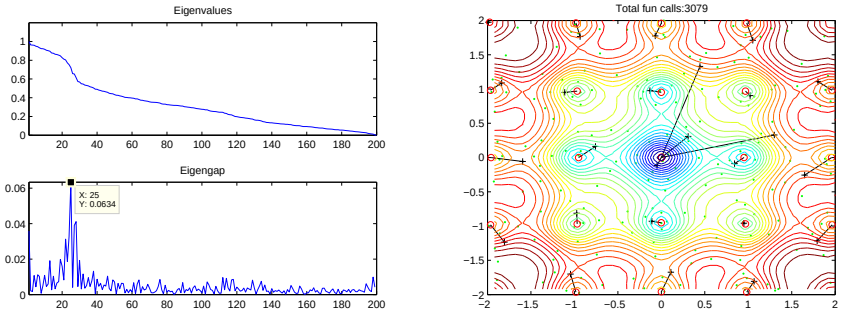


Figure 10.10: Ackley's function 200 starting points. Slightly transformed sample



(a) Sorted eigenvalues, eigengap (b) Final clusters

Figure 10.11: Using 2 neighbors for affinity matrix ($k = 25$)



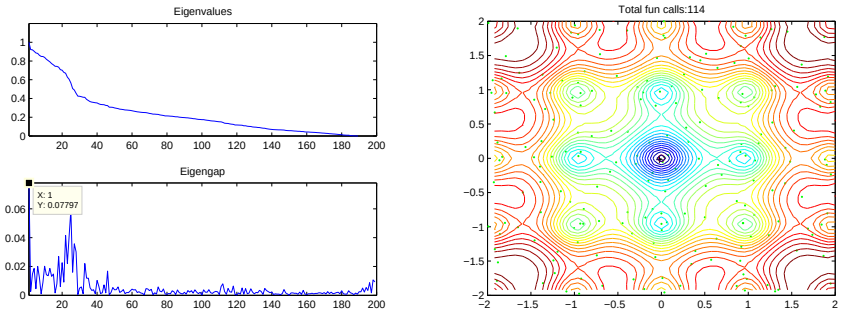
(a) Sorted eigenvalues, eigengap (b) Final clusters

Figure 10.12: Using 3 neighbors for affinity matrix ($k = 25$)

attraction, and their negative gradient directions (green arrows), drive them near the same minimum. This is not the case considering the pairs x_1 with y_1 and x_2 with y_2. It is clear from the figure that following the negative gradient, will lead them to different minima.

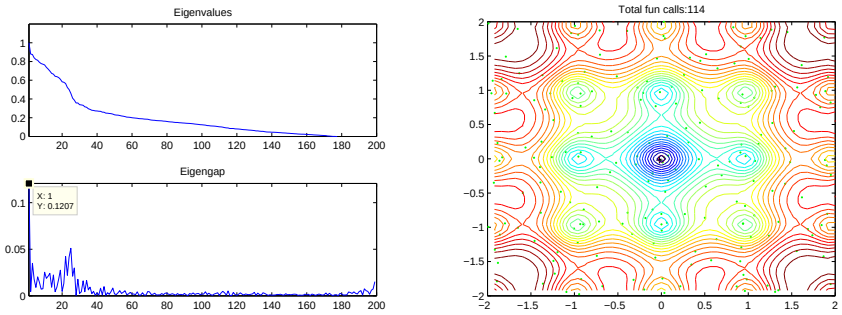
The idea is simple: *Two points are associated if following the negative gradient, reduces their distance.*

Consider two transformed points x_1 and x_2. For each point, perform an infinitesimal



(a) Sorted eigenvalues, eigengap (b) Final clusters

Figure 10.13: Using 4 neighbors for affinity matrix ($k = 1$)



(a) Sorted eigenvalues, eigengap (b) Final clusters

Figure 10.14: Using 5 neighbors for affinity matrix ($k = 1$)

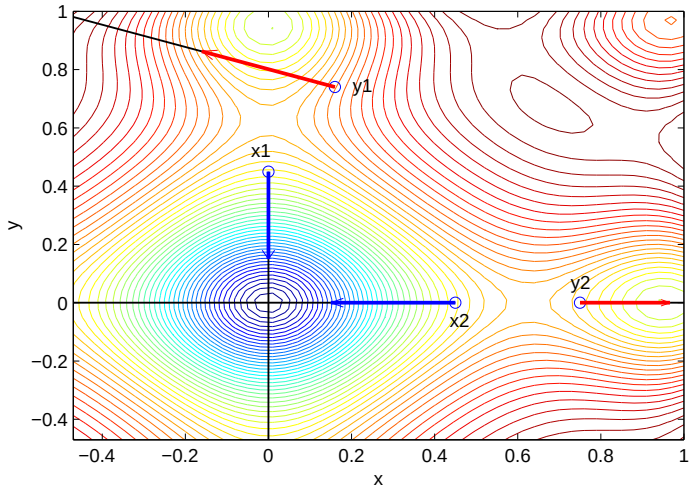
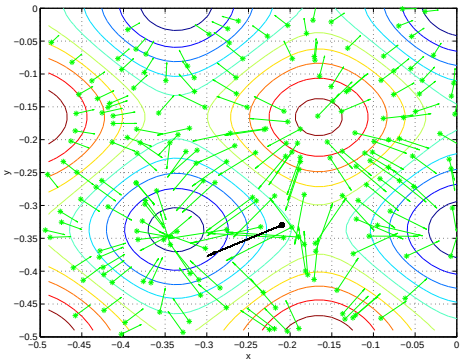


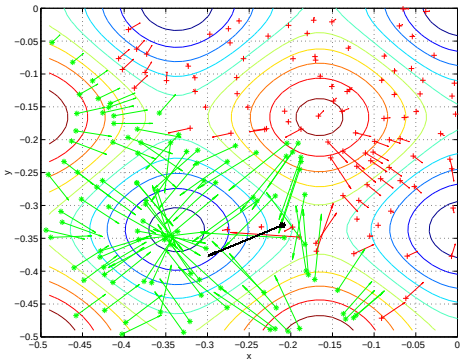
Figure 10.15: Example of gradient association criterion

step along the negative gradient direction and obtain $y_1 \leftarrow x_1 - \epsilon \nabla f(x_1)$ and $y_2 \leftarrow x_2 - \epsilon \nabla f(x_2)$. If the distance $||y_2 - y_1|| < ||x_2 - x_1||$ then points x_1 and x_2 tend to approximate each other by following the negative gradient direction. On the other hand, if $||y_2 - y_1|| > ||x_2 - x_1||$ then a small step along the negative gradient dissociates the two

points.

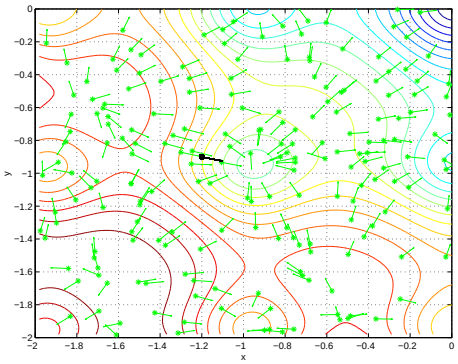


(a) Without gradient information

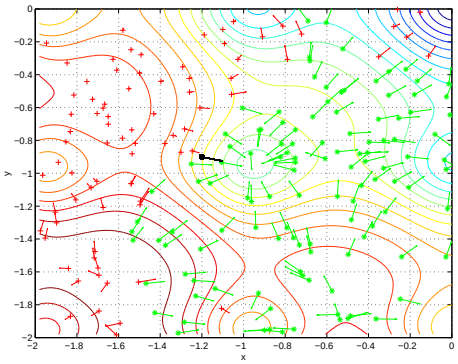


(b) With gradient information: Green color denotes amplification, red color denotes reduce

Figure 10.16: A plot of pairwise affinities between samples using Rastrigin's function



(a) Without gradient information



(b) With gradient information: Green color denotes amplification, red color denotes reduce

Figure 10.17: A plot of pairwise affinities between samples using Ackley's function

Global k-means Algorithm

Global k-means algorithm [**?**] is an incremental approach to clustering that dynamically adds one cluster center at a time through a deterministic global search procedure consisting of N (with N being the size of the data set) executions of the k-means algorithm from suitable initial positions. Global k-means attempts to find the global minimum of the clustering error metric.

$$E(m_1, m_2, \ldots, m_M) = \sum_{i=1}^{N} \sum_{k=1}^{M} I(x_i \in C_k) ||x_i - m_k||^2, \tag{10.2}$$

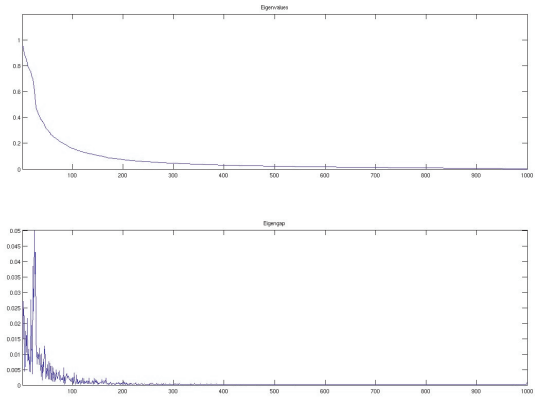


Figure 10.18: Sorted eigenvalues of the affinity matrix and the corresponding eigengap, without gradient information

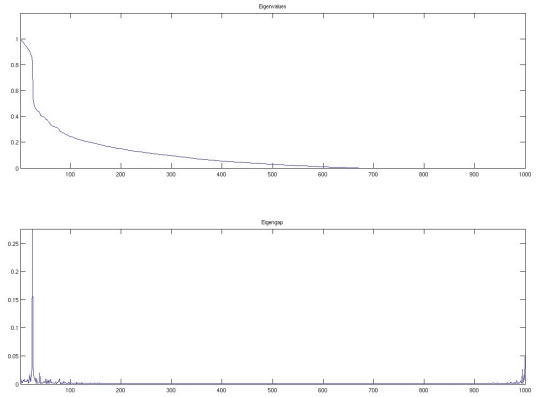


Figure 10.19: Sorted eigenvalues of the affinity matrix and the corresponding eigengap using the gradient information

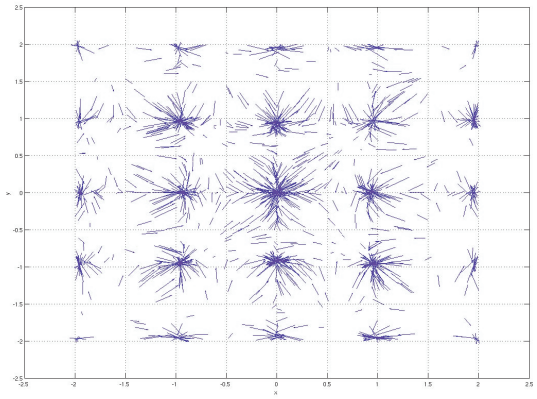


Figure 10.20: Gradient vector plot of the concentrated sampled points

where $I(Cond) = 1$ if $Cond$ is true and $X = x_1, x_2, \ldots, x_N, x_i \in R^d$ is the data set that is going to be partitioned into M disjoint clusters $C_1, C_2, \ldots, C_M$ with centers $m_1, m_2, \ldots, m_M$ respectively.

More specifically, to solve a clustering problem with M clusters the method proceeds as follows. The method starts with one cluster ($k = 1$) and find its optimal position

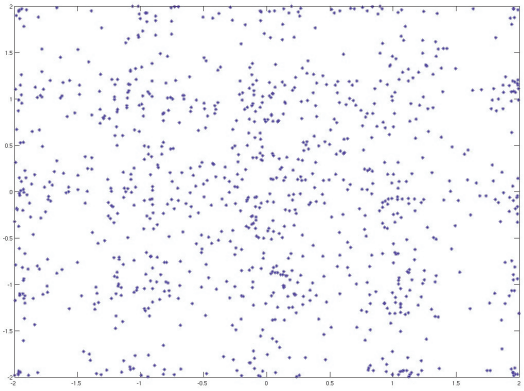


Figure 10.21: Positional plot of the concentrated sampled points

which corresponds to the centroid of the data set X. In order to solve the problem with two clusters ($k = 2$) N executions of the k-means algorithm are performed from the following initial positions of the cluster centers: the first cluster center is always placed at the optimal position for the problem with $k = 1$, while the second center at execution n is placed at the position of the data point $x_n (n = 1, \ldots, N)$. The best solution obtained after the N executions of the k-means algorithm is considered as the solution for the clustering problem with $k = 2$.

In general, once the solution for the $(k-1)$- clustering problem is found, the algorithm attempts to find the solution of the k-clustering problem by performing N runs of the k-means algorithm with k clusters where each run n starts from the initial state $(m_1^*(k-1), \ldots, m_{(k-1)}^*(k-1), x_n)$[2]. The best solution obtained from the N runs is considered as the solution $(m_1^*(k), \ldots, m_k^*(k))$ of the k-clustering problem. By proceeding in the above fashion a solution with M clusters is finally obtained having also found solutions for all k-clustering problems with $k < M$.

Global k-means algorithm main advantage is that is independent of any parameter initialization. Also it is stated by its authors that is *experimentally optimal*, in a sense that is equivalent to numerous random restarts of k-means.

10.5 The proposed algorithm

The proposed algorithm is presented in a complete form in Algorithm 10.38. The user must set the switches that define the sample strategy, the concentrating method, whether or not to use gradient information.

[2] $m_j^*(k)$ is the j-the center computed when the $k-$clustering problem is solved

Algorithm 10.38 Proposed clustering algorithm

Input: f: Minimizing function, *xl, xu*: Problem's bounds, N: Sample size,
Isampl: Defines sample strategy,
Ired: Switches between the concentrating method,
Iuseg: Use gradient information,
Iclust: Switches between global kmeans/kmedoids,
Inei: Number of neighbors to calculate σ

S1: **If** `Isampl = 1` **Then** { *Sample N starting points* }
    $X \leftarrow \text{Uniform}(N, xl, xu)$
Else
    $X \leftarrow \text{Halton}(N, xl, xu)$
End If

S2: **If** `Ired = 1` **Then** { *Concentrate sample points* }
    For i=1 **to** N
        $X(i) \leftarrow \text{Local}(X(i), iter)$
    End For
Else
    For i=1 **to** N
        For k=1 **to** *iter*
            $X(i) \leftarrow X(i) - \alpha \nabla f(X(i))$
        End For
    End For
End If

S3: $A \leftarrow \text{Affinity}(X, Iuseg, Inei)$
$[e_1, \ldots, e_n] \leftarrow \text{Eigenvalues}(A)$
Sort $[e_1, \ldots, e_n]$ in decreasing order and calculate maximum eigengap at k
If `Iclust = 1` **Then** { *Apply global k-means/medoids* }
    $[M, Dis] \leftarrow \text{Gkmeans}(X, k)$
Else
    $M \leftarrow \text{Gkmedoids}(A, k)$
End If

Algorithm 10.39 Algorithm *Affinity*

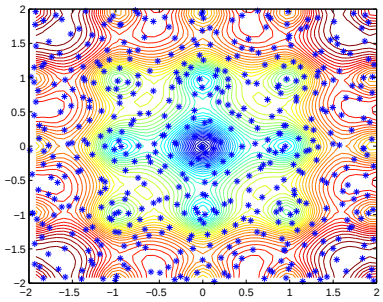
Input: f: Minimizing function, X: Sampled points, N: Sample size,
Iuseg: Use gradient information
Inei: Number of neighbors for the calculation of σ

Output: A: Affinity matrix

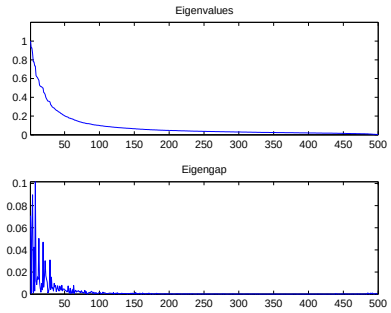
S1: **For** i=1 to N **do**
  For j=1=i to N **do**
    $D_{i,j} = ||X_i - X_j||_2$
  End For
End For

S2: **For** i=1 to N **do**
  For j=1=1 to N **do**
    $dis(j) = D_{i,j}$
  End For
  $dis_{sort} \leftarrow sort(dis)$
  $m(i) \leftarrow \frac{1}{Inei} \sum_{k=1}^{Inei} dis_{sort}(k)$
End For
$\sigma = \frac{1}{IN} \sum_{k=1}^{N} m(i)$

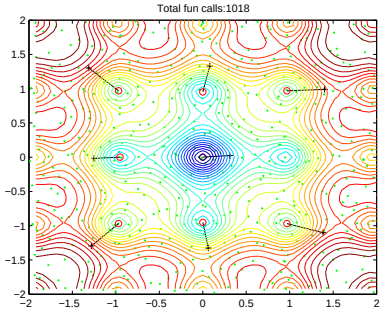
Figure 10.22: An illustration of our approach for Ackley's test function



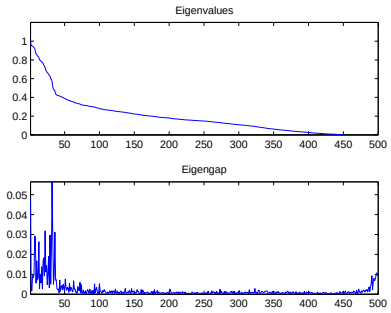
(a) 500 random starting points



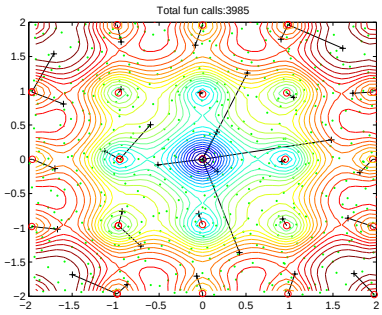
(b) Estimate k without gradient information ($k = 8$)



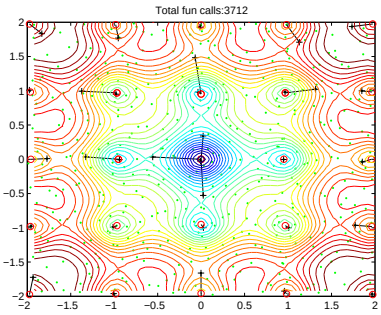
(c) Apply global k-means using $k = 8$



(d) Estimated k using gradient information ($k = 32$)



(e) Apply global k-means using $k = 32$

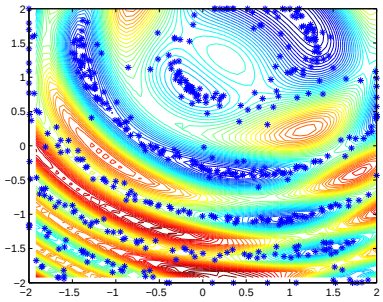


(f) Apply global k-medoids using $k = 32$

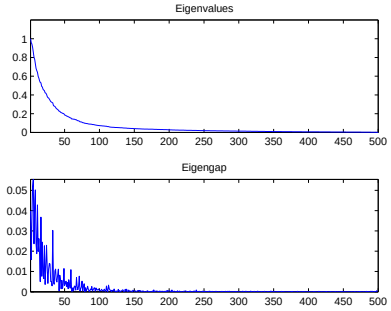
10.6 Implementation and numerical experiments

We have implemented our clustering approach along with the overall global optimization framework in Matlab. We used the Global K-means algorithm implementation from `http://lear.inrialpes.fr/~verbeek/code/` and modified the source code to implement the medoid modification.

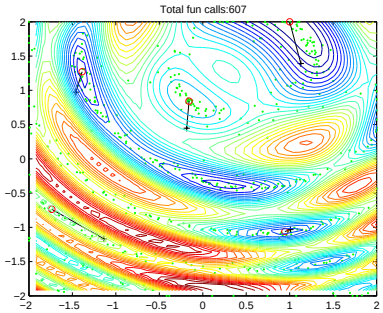
Figure 10.23: An illustration of our approach for random quadratics test function



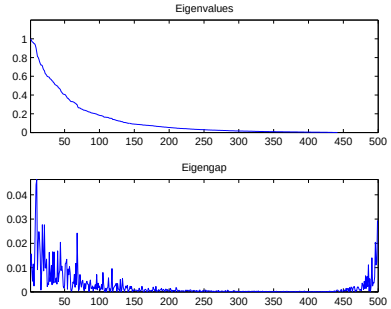
(a) with 500 random starting points



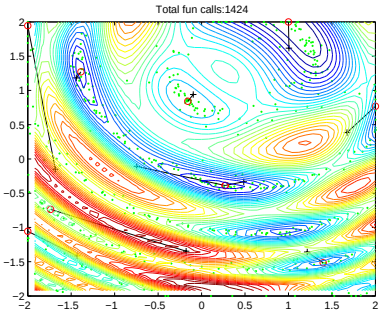
(b) Estimate k without gradient information ($k = 8$)



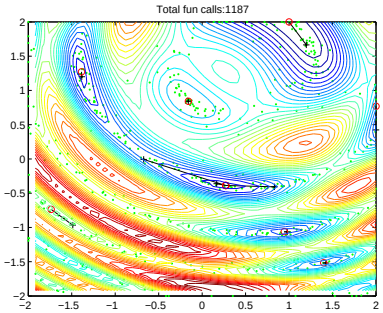
(c) Apply global k-means using $k = 5$



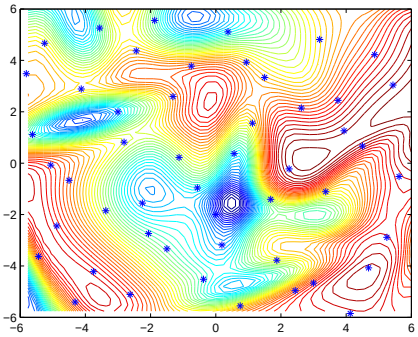
(d) Estimated k using gradient information ($k = 32$)



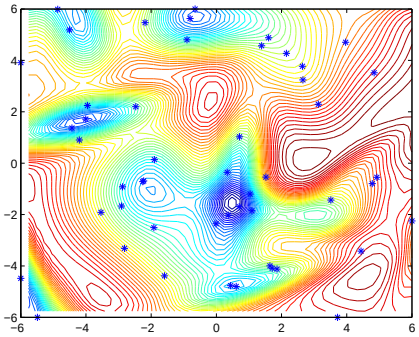
(e) Apply global k-means using $k = 10$



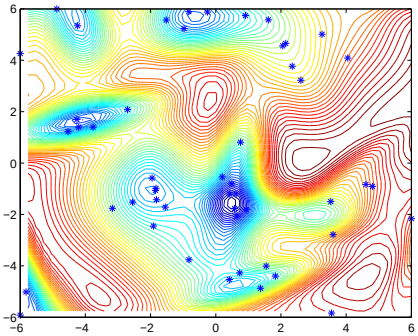
(f) Apply global k-medoids using $k = 10$



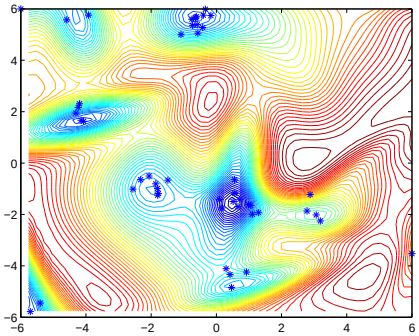
(g) iter 1



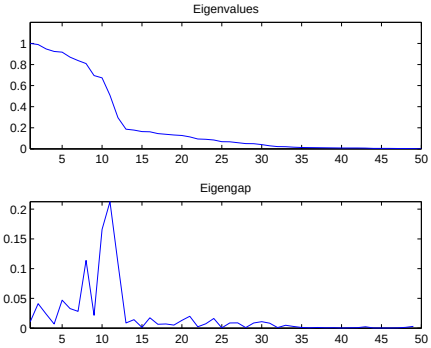
(h) iter 5



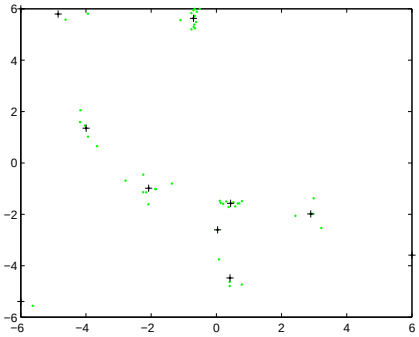
(i) iter 10



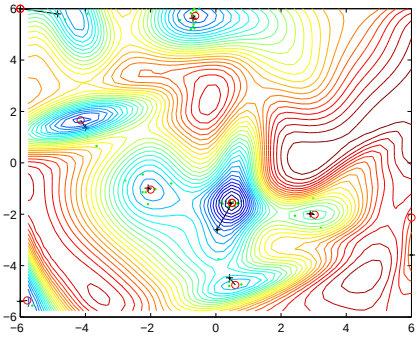
(j) iter 15



(k) Eigenvalues



(l) Clustering



(m) Minima retrieved

Chapter 11

A Local Search with "Strictly" Monotonic Descent and its Application in Global Optimization

11.1 Introduction

Stochastic methods based on multistart, that employ a clustering scheme to separate different regions of attractions have proven to be quite successful. The research in this direction was pioneered by Rinnoy Kan and its group in a series of articles eg.[]. Various authors followed up this line, see for example Torn and Viitanen, Schoen and Locatelli, Aliand Storey and a host of methods and software implementations have appeared in the literature. A common feature of these methods is the use of a local search (LS), i.e. a procedure for locating a local minimum. The characteristics of this procedure play an important role as far as the performance and the effectiveness of the global method is concerned. If by $x^* = \mathcal{L}(x)$ we denote that a local search started at point x, will end up finding the local minimizer x^*, then the region of attraction of a minimizer x^* may be defined as the set $A(\mathcal{L}, x^*) = \{x_i, x^* = \mathcal{L}(x_i)\}$ and depends in addition to the position of the minimum x^*, on the LS procedure.

If x^* and y^* are distinct local minima $A(\mathcal{L}, x) \cap A(\mathcal{L}, y) \neq \emptyset$ provided that the local search is deterministic. Stochastic LS procedures create overlapping regions of attraction a fact that in the framework is rather undesirable. Also regions of attraction may be contiguous or not. Evidently a non–contiguous region can not be described by a single cluster, and hence the existence of such regions may influence the performance of the method negatively. So a proper LF for clustering should be such that the regions of attraction that it creates are contiguous. Vrahatis et al have provided a tool for visualizing the regions of attraction. An interesting fact is that all of the most successful LS search create disjoint regions. Hence a LS with contiguous regions of attraction would be very useful for clustering methods.

$$\text{Find all } x_i^* \in S \subset R^n \text{ that satisfy:}$$
$$x_i^* = \arg\min_{x \in S_i} f(x), \quad S_i = S \cap \{x, ||x - x_i^*|| < \epsilon\} \tag{11.1}$$

S is considered to be a bounded domain of finite measure and ϵ a positive infinitesimally small number. This problem appears frequently as a subproblem in a variety of scientific applications. Local search procedures play important role in most of the robust global optimization algorithms.

11.2 Motivation towards a new local search

In their seminal paper Kan and Timmer [133] introduced two local searches mostly for theoretical reasons. The first one was called *strictly local search* that generates sequences of points x_k and descent directions p_k such that

$$x_{k+1} = x_k + a_k p_k \quad (||p_k|| = 1, a_k > 0) \tag{11.2}$$

which converges to a stationary point and moreover

$$f(x_k + \beta p_k) \leq f(x_k + \alpha p_k) \tag{11.3}$$

for all k and all α, β. In order to derive a more tractable local search, since *strictly local search* cannot be verified computationally, they defined an ϵ-*descent* procedure that satisfy (11.2) and moreover

$$f(x_k + i\epsilon p_k) \leq f(x_k + (i-1)\epsilon p_k) \quad \left(i = 1, 2, \ldots, \left[\frac{\alpha_k}{\epsilon}\right]\right) \tag{11.4}$$

The authors use this ϵ-*descent* procedure in order to prove that this local search when started from a point inside the proper level set that contains the minimum should always converge to that minimum (see Theorem 6 in [133]). When the authors present their computational results in [134] they apply a local search with practically far behavior from the ϵ-*descent* procedure. However to our knowledge this is the only reference in the bibliography that reveals the need (theoritical) to define a strictly descent local search algorithm.

In the spirit of [133], we present a special local search which we are going to call *Infinitesimal gradient descent* and is equivalent to the ϵ-*descent* procedure.

We indent to use Algorithm 11.40 as a model local search with specific properties, in conjuction with a common local search that combines quasi–Newton updates (specifically BFGS) with a simple backtracking local search. For demonstration we will use the well known Ackley's function [] inside the bounds $[-1.5, 1.5]^2$ where nine minima exist. We have chosen uniformly random some starting points inside $[-1.5, 1.5]^2$ and performed local searched using infinitesimal gradient descent and the common BFGS–backtracking

Algorithm 11.40 Infinitesimal gradient descent

Let x_k the current iterate and $\epsilon > 0$ a small number:

1. Set $p_k = -\nabla f(x_k)$
2. Set $x_{k+1} = x_k + \epsilon p_k$
3. If termination criteria are met stop else goto 1.

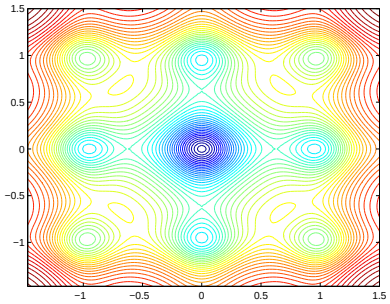
scheme. In Figures 1(c) and 1(d) we present contour drawing of Ackley function where the starting points are connected with a blue line to the ending points (minima). It is obvious from Figure 1(c), were the infinitesimal gradient descent was applied, that all local searches ended up to the minimum in the same basin, whereas in Figure 1(d) we can clearly see *jumps* from one basin to another. Also the fraction of starting points that resulted in the minimum of the same basin, to the total number of minima is in direct correspondence to the relative size of the basin itself when infinitesimal local search is applied. Another observation is that the mean distance from starting points to the minimum, which is shown in Figures 1(c), 1(d) using red cycles is also is also proportional to the relative size of the basin of the specific minimum. Hence, we can easily deduce that the application of a strictly local search, allows a better "topographical" mapping of the underlying function.

Another advantage of a strictly local search is displayed in Figures 1(e), 1(f). In these figures we attempt a cartography of the regions of attraction using the aforementioned local search procedures. It is clear that the strictly local search produces contiguous regions of attractions which approximate very closely the basins of attraction of the corresponding minima, whereas common local search result in discontiguous regions of attraction.

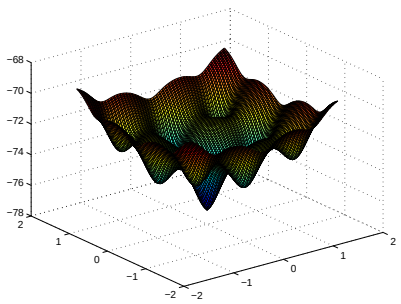
In order to illustrate the significance of a strictly local search in a global framework consider the case where for each local minimum we create a gaussian function that describes the region of attraction. Let $x_1, x_2, \ldots, x_m$ the starting points that a local search $\mathcal{L}$ maps to a minimum x^*. Then the gaussian can be defined as:

$$N(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}} \frac{1}{\sqrt{|\Sigma|}} \exp(-0.5\ (x-\mu)^T \Sigma (x-\mu));$$

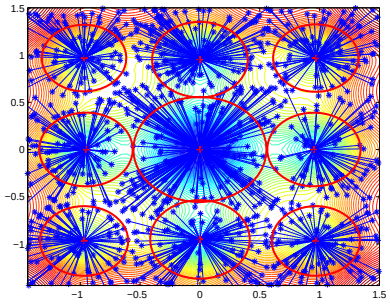
where $\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x_i - x^*)^T (x_i - x^*)$ and $\mu = \frac{1}{m} x_i$. We have performed different local searches from random initial points, recorder the minima obtained and constructed the gaussians per minimum. In Figure 11.2 we present plots of these gaussians for different local searches: (i) classical BFGS, (ii) steepest descent with very small step, (iii) the proposed methodology.



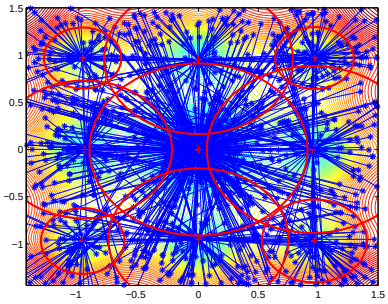
(n) Ackley's function contour



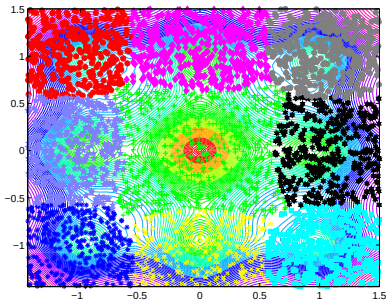
(o) Ackley's function surface



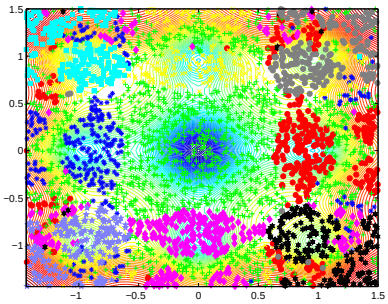
(p) Steepest descent + Infinitesimal step



(q) BFGS + Armijo backtracking



(r) Steepest descent + Infinitesimal step

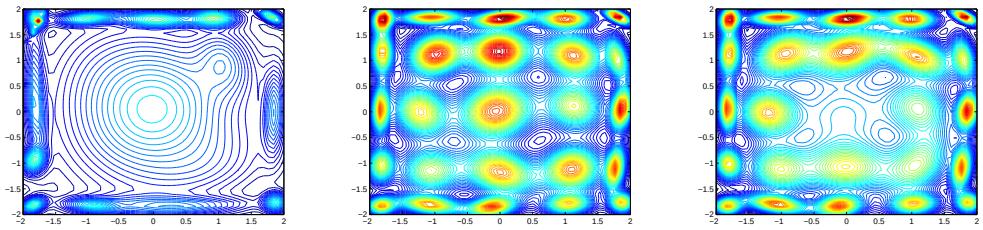


(s) BFGS + Armijo backtracking

Figure 11.1: Regions of attraction

11.3 Description of the new local search

The local search presented in this work belongs to the line search framework. The choice of the search direction will be determined be the algorithm used (gradient descent, Newton, Quasi-Newton) and it is not discussed here. It is the line search part that produces sufficient descent for the local search that has to be modified in order ti match the requirements imposed in previous section. In this section we will present the necessary modification of the line search step, so that the overall local search algorithm (regardless the search direction selection) will lead to the "nearest minimum".



(a) BFGS + Armijo backtracking (b) Infinitesimal gradient descent (c) Proposed local search

Figure 11.2: Contour plot of the gaussians around minima

11.3.1 Original idea

Suppose that s is a descent direction at x, i.e. $s^T \nabla f(x) < 0$. We want to minimize $f(x + \lambda s)$ with respect to λ, in such a way that the minimum is the same with the one that would be found via a steepest descent with infinitesimal step. In addition we do not want to take infinitesimal steps, since convergence will be deteriorated and of course we want to avoid large steps, since otherwise distant local minima may be recovered. The Armijo condition:

$$f(x + \lambda s) < f(x) + \rho \lambda s^T \nabla f(x)$$

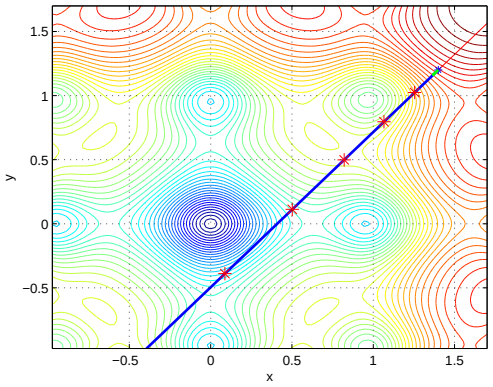
bounds the step from above, and guarantees sufficient descent. We will not use the Wolfe condition to bound the step from below since this bound may force a too high lower bound. Instead make a grid on the permissible values of λ as follows:

$$\lambda_i = \frac{\mu^i - 1}{\mu^\nu - 1} \min(1, \frac{\max(1, |x|)}{|s|})$$

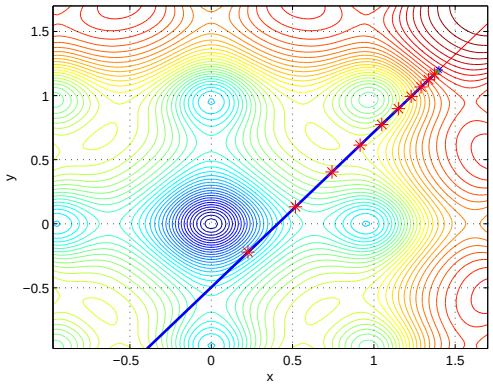
where ν is the number of allowed values of λ, and $\mu > 1$. Both ν, μ are set by the user. Typical values are $\nu = 10, \mu = 1.8$. The $\min(1, \frac{\max(1,|x|)}{|s|})$ part is to adjust to the problem's typical size. This choice guarantees that finite steps are taken, and preference is given to the area close to x. This is illustrated in Figure 11.3, where on the left side we see the points along the direction without the scaling factor and on the left side the scaling factor is applied. It is obvious from observing the figures that

The first λ_i that meets the Armijo condition may be taken. Alternatively, more steps satisfying Armijo may be taken, as long as no increase in the function's value is observed. It is implicitly assumed that this line-search will be used in conjunction with a Newton type of method, where $\lambda = 1$ is a useful choice. The grid suggested above for λ takes on as the n^{th} value $\min(1, \frac{\max(1,|x|)}{|s|})$ which guarantees that $\lambda \leq 1$. If Armijo condition is not satisfied for all $i = 1, \ldots, \nu$ then we scale the maximum step to $\frac{\mu^i - 1}{\mu^n - 1} \min(1, \frac{\max(1,|x|)}{|s|})$, and repeat the process. Keeping in mind the above analysis a first algorithm is presented in Algorithm 11.42.

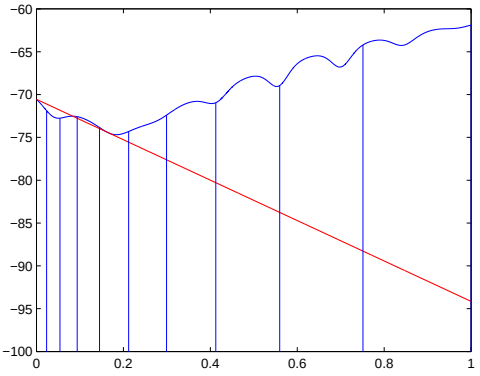
Figure 11.4 illustrates the behavior of Algorithm 11.42 in two cases. In the first case the desired minimum is missed. We also present the line search at the first iterate.



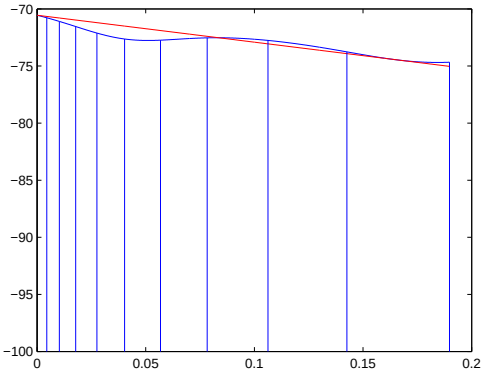
(a) No scaling factor: Contour



(b) With scaling factor: Contour



(c) No scaling factor: Along search direction



(d) With scaling factor: Along search direction

Figure 11.3: The significance of scaling factor $\min(1, \frac{\max(1,|x|)}{|s|})$

Algorithm 11.41 New local search: Version 1

Input:

x: Current iterate

f: Function to be minimized

d: Descent direction from the outer Newton–like local search

ρ: Armijo rule parameter

$\mu, \nu > 0$: Method's parameters

Output:

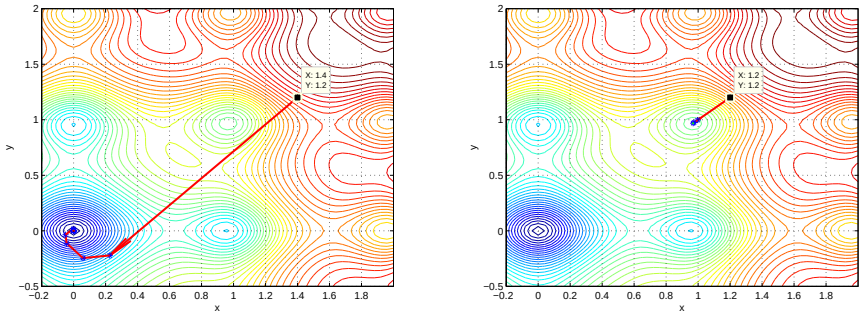
x': Next iterate, α: Line search step , fc: Function calls

1. **Initialize:**

   $scale \leftarrow 1,\ fc \leftarrow 0,\ \text{term} \leftarrow \textbf{false}$

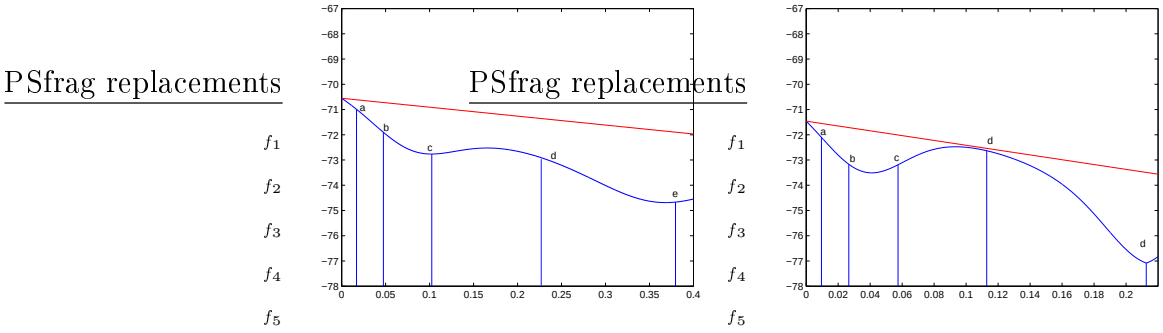
2. **Main Step:**

   while term=**true do**
   for i=1, ν **do**
   $\lambda_i \leftarrow scale \cdot \frac{\mu^i - 1}{\mu^\nu - 1} \cdot \min\left(1, \frac{\max(1, ||x||))}{||d||}\right)$
   if $f(x + \lambda_i d) < f(x) + \rho\lambda_i \cdot d^T \nabla f(x)$ **then** { *Bellow ρ line* }
   if $f(x + \lambda_i d) > f(x + \lambda_{i-1} d)$ **then** { *No improvement* }
   $\alpha \leftarrow \lambda_{i-1}$
   $x' \leftarrow x + \alpha d$
   term $\leftarrow$ **true**, **break**
   end if
   else { *Above ρ line* }
   $\alpha \leftarrow \lambda_{i-1}$
   $x' \leftarrow x + \alpha d$
   term $\leftarrow$ **true**, **break**
   end if
   $fc \leftarrow fc + 1$
   end
   $scale \leftarrow scale \frac{\mu^i - 1}{\mu^\nu - 1} \cdot \min\left(1, \frac{\max(1, ||x||))}{||d||}\right)$
   end



(a) Algorithm 11.42 unable to locate the desired minimum

(b) Algorithm 11.42 locates the desired minimum



(c) Decreasing sequence $f_1 > f_2 > f_3 > f_4 > f_5$

(d) The algorithm stops at x_3 because $f_3 < f_4$

Figure 11.4: Illustrative behavior of Version 1

11.3.2 Including gradient information

In the case off differentiable objective function, gradient information can be used to stop the forward looking line search even though the function values are decreasing. Gradient information provide the necessary conditions to stop even in the bad case presented in Figure 11.4.

The idea is simple and is illustrated in Figure 11.5. Compare the dot product of the search direction to the starting point with the dot product of the search direction to every trial point on the line. The intuition is that if a trial point overpass a local minimum the gradient vector will still point to its direction. In Figure 11.5 the 7-th trial point is the first trial point that the dot product of its gradient to the search direction is positive (first red line). All trial points before that have negative corresponding dot product (green lines). In this way even though a decrease in function value is achieved, the algorithm is stopped from considering further points.

The modification of the gradient criterion is shown in Algorithm 11.42.

Algorithm 11.42 New local search: Version 2 with gradient information (Main Step)

1. **Main Step:**

while term=**true do**
for i=1, ν **do**
$\lambda_i \leftarrow scale \cdot \frac{\mu^i - 1}{\mu^\nu - 1} \cdot \min\left(1, \frac{\max(1, ||x||))}{||d||}\right)$
if $f(x + \lambda_i d) < f(x) + \rho\lambda_i \cdot d^T \nabla f(x)$ **then** { *Bellow* ρ *line* }
if $f(x + \lambda_i d) > f(x + \lambda_{i-1} d)$ **then** { *No improvement* }
$\alpha \leftarrow \lambda_{i-1}$
$x' \leftarrow x + \alpha d$
term $\leftarrow$ **true**, **break**
else { *Bellow* ρ *line and improving* }
$g_i \leftarrow \nabla f(x_i)$
$gc \leftarrow gc + 1$
if $g_i^T d > 0$ **then**
$\alpha \leftarrow \lambda_{i-1}$
$x' \leftarrow x + \alpha d$
term $\leftarrow$ **true**, **break**
end if
end if
else { *Above* ρ *line* }
$\alpha \leftarrow \lambda_{i-1}$
$x' \leftarrow x + \alpha d$
term $\leftarrow$ **true**, **break**
end if
$fc \leftarrow fc + 1$
end
$scale \leftarrow scale \frac{\mu^i - 1}{\mu^\nu - 1} \cdot \min\left(1, \frac{\max(1, ||x||))}{||d||}\right)$
end

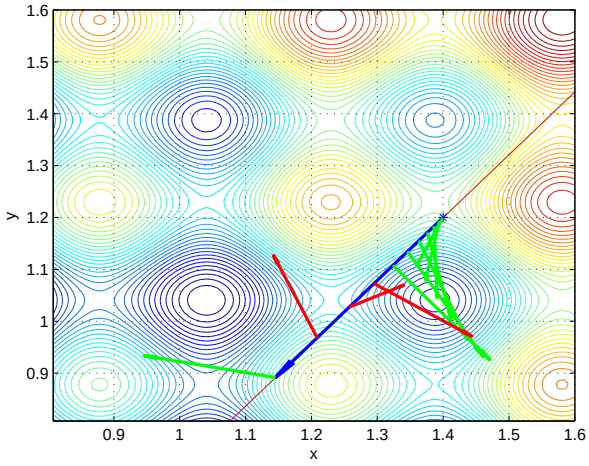


Figure 11.5: Illustration of the gradient information

11.3.3 Accelerating: A way of choosing ν

In the above analysis we chose a constant value for ν. This value plays important role regarding the efficiency of the line search. Keep in mind that the proposed algorithm performs up to ν function evaluation per iteration. The upper bound of ν function calls is reached when taking the full step. Experience from line search algorithms, especially of the Newton class, suggest that when close to the minimum the line search usually takes full steps.

In order to avoid the further computational cost, we devised another way of optimally choosing the value of ν. This estimation is based on the magnitude of the derivative at the starting point of the local search and significantly reduces the value of ν when close to minimum. The requirement that large derivatives should correspond to small steps, may be expressed as:

$$h = \frac{1}{1 - s^T \nabla f(x)}$$

Note that as $s^T \nabla f(x) \to \infty, \quad h \to 0$ and that as $s^T \nabla f(x) \to 0, \quad h \to 1$ Then we may demand that $\lambda_1 = h$, i.e.

$$\frac{\mu - 1}{\mu^n - 1} \min(1, \frac{\max(1, |x|)}{|s|}) = \frac{1}{1 - s^T \nabla f(x)}$$

from which we may determine n. In addition we must safeguard n so that $n \in [1, 10]$, the upper bound (10) being there for both numerical and reasons of efficiency.

All the above are displayed in Algorithm 11.43.

11.4 Experiments and comparison

In order to estimate the efficiency of the proposed line search both in terms of (a) quality of the solution and (b) function calls, we conducted a plethora of experimental comparisons.

Algorithm 11.43 New local search: Version 3 choosing ν (Initialize)

1. **Initialize:**

$scale \leftarrow 1,\ fc \leftarrow 0,\ gc \leftarrow 0$

$h \leftarrow \frac{1}{1-d^T \nabla f(x)}$

$sc \leftarrow \min\left(1, \max\left(1, \frac{||x||}{||\nabla f(x)||}\right)\right)$

$\nu' \leftarrow \nu \ \ , \nu \leftarrow \left\lfloor \min\left(\nu', \frac{\log(1+(\mu-1)\frac{sc}{h})}{\log(\mu)}\right)\right\rfloor$

$\nu = \max\left(\left\lfloor (\nu' - \nu) \cdot e^{0.1(-iter+1)} + \nu \right\rfloor, 1\right)$

2. **Main Step:**

11.4.1 Efficiency vs. Cost

First, we compared a classical backtracking algorithm to the proposed methodology, on several test functions and random starting points. For all experiments BFGS update was used to obtain the hessian approximation. The goal for each algorithm is to obtain, from the same starting point, the same minimum that the infinitesimal gradient descent would. We measure the starting points that lead to the correct minimum, and also the total number of iterations (line search iterations) and the total number of function calls.

Table 11.1 holds the results for the Armijo type backtracking local search. Notice that almost half of the starting points fail to lead to the correct minimum, by the application of the local search. On the other hand it took 498111 total function evaluations to locate these minima. In Table 11.2 we present the same results for the case of our proposed line search methodology. We can see that the percentage of correct classified starting points, is equal to 88.9%, much more improved than the previous case. The price we pay for this performance is a total of 1294234 function calls which is more than double from the Armijo case. We must comment here that the criterion for selecting starting ν worked very well since we expect almost ten times the cost of the Armijo case.

11.4.2 The proposed search in a global framework

Density Clustering

Typical Distance Clustering

Table 11.1: Results for the armijo type backtracking line search

Armijo Type Local Search					
Function	Correct	Error	Iters.	Fun. Calls	% Success
Ackley	555	445	15704	24978	55,50%
Giunta	588	442	12013	15855	57,09%
Guillin	477	523	17764	29974	47,70%
Levy3	1285	715	20783	31624	64,25%
Rastrigin	599	401	9240	12822	59,90%
Griewank	695	305	10244	13422	69,50%
Bird	704	296	14104	19960	70,40%
Levy5	1272	728	21476	31377	63,60%
Rot. Quad	539	461	12001	18408	53,90%
Holder	597	403	12235	17430	59,70%
Liang	561	439	11671	18912	56,10%
Piccioni	726	274	24874	44702	72,60%
Shekel	145	155	3757	7493	48,33%
M0	717	1283	132739	142455	35,85%
Lager	581	419	11342	16537	58,10%
Tube	727	273	10034	13719	72,70%
Mich	178	322	11472	16222	35,60%
Dejong	301	199	18417	22221	60,20%
Sum / Ave.	11247	8083	369870	498111	57.8%

Table 11.2: Results the proposed line search, $\nu = 10$, $\mu = 1.1$

Proposed local Search n=10, m=1.1					
Function	Correct	Error	Iters.	Fun. Calls	%
Ackley	878	122	12498	63081	87,80%
Giunta	905	95	10647	71457	90,50%
Guillin	891	109	11276	69199	89,10%
Levy3	1901	990	20619	145414	95,48%
Rastrigin	1000	0	9187	56269	100,00%
Griewank	996	4	9785	69632	99,60%
Bird	910	90	11907	76377	91,00%
Levy5	1811	189	20220	31377	90,55%
Rot. Quad	908	92	10884	68136	90,80%
Holder	993	7	11739	73539	99,30%
Liang	698	301	11660	78622	69,87%
Piccioni	996	4	8914	77807	99,60%
Shekel	246	54	3407	20818	82,00%
M0	1188	812	20659	132656	59,40%
Lager	884	116	11226	71135	88,40%
Tube	1000	0	9040	60031	100,00%
Mich	355	145	12706	56807	71,00%
Dejong	484	16	17469	71877	96,80%
Sum / Ave	17044	3146	223843	1294234	88.90 %

Table 11.3: Results the proposed line search, $\nu = 20$, $\mu = 1.1$

Local Local Search n=20, m=1.1					
Function	Correct	Error	Iters.	Fun. Calls	%
Ackley	763	237	11928	100858	76,30%
Giunta	911	89	9981	120457	91,10%
Guillin	992	8	10566	115394	99,20%
Levy3	1987	13	19312	205563	99,35%
Rastrigin	973	27	8567	94587	97,30%
Griewank	908	20	9288	116712	97,84%
Bird	915	85	11145	120009	91,50%
Levy5	1949	51	19064	203544	97,45%
Rot. Quad	911	89	10260	111870	91,10%
Holder	958	42	11092	121370	95,80%
Liang	914	85	10971	113345	91,49%
Piccioni	749	251	8991	108966	74,90%
Shekel	249	51	3170	32622	83,00%
M0	1683	317	20388	224573	84,15%
Lager	939	61	10651	109857	93,90%
Tube	1000	0	8565	102273	100,00%
Mich	382	118	12543	89934	76,40%
Dejong	472	28	17070	117007	94,40%
Sum / Ave	17655	1572	213552	2208941	90,8%

Table 11.4: Results the proposed line search, $\nu = 30$, $\mu = 1.1$

Local Local Search n=30, m=1.1					
Function	Correct	Error	Iters.	Fun. Calls	%
Ackley	733	267	13462	46754	73,30%
Giunta	910	90	9744	171806	91,00%
Guillin	1000	0	10261	162841	100,00%
Levy3	1982	18	18675	281502	99,10%
Rastrigin	973	27	8265	135661	97,30%
Griewank	980	20	9078	166388	98,00%
Bird	916	84	10802	167742	91,60%
Levy5	1953	47	18252	279071	97,65%
Rot. Quad	909	91	9943	158114	90,90%
Holder	957	43	10828	171573	95,70%
Liang	923	76	10634	150537	92,39%
Piccioni	265	735	8597	143226	26,50%
Shekel	249	51	3066	45994	83,00%
M0	1692	308	19803	224573	84,60%
Lager	942	58	10298	153491	94,20%
Tube	1000	0	8362	146565	100,00%
Mich	397	103	11092	258622	79,40%
Dejong	473	27	16717	162302	94,60%
Sum / Ave.	17254	2045	207879	3026762	88,92 %

Table 11.5: Results the proposed line search, $\nu = 10$, $\mu = 1.3$

Local Local Search n=10, m=1.3					
Function	Correct	Error	Iters.	Fun. Calls	%
Ackley	874	126	11555	61583	87,40%
Giunta	912	95	12891	83760	90,57%
Guillin	896	109	10211	55838	89,15%
Levy3	1910	90	19882	139418	95,50%
Rastrigin	1000	0	9031	58571	100,00%
Griewank	998	2	9403	72245	99,80%
Bird	913	87	10782	78446	91,30%
Levy5	1811	189	19282	21956	90,55%
Rot. Quad	903	97	9964	69611	90,30%
Holder	995	5	10554	63490	99,50%
Liang	702	298	11782	87584	70,20%
Piccioni	997	3	9021	81703	99,70%
Shekel	250	50	3783	24281	83,33%
M0	1200	800	19826	132526	60,00%
Lager	901	99	10435	69524	90,10%
Tube	1000	0	8991	50724	100,00%
Mich	356	144	11282	55817	71,20%
Dejong	489	12	16822	53632	97,60%
Sum / Ave	17107	2206	215497	1260710,6	89,23%

Table 11.6: Results from density clustering global optimization algorithm

	Backtracking + Armijo			**Proposed Local Search**		
	nloc	nliter	funcalls	nloc	nliter	funcalls
Ackley	6206	84678	154830	5746	63368	345122
Giunta	2399	30741	46645	1818	21133	143252
Guillin Hills	124908	2166570	3927158	111144	1269789	7909498
Levy3	23184	254416	459374	17628	188569	1362860
Rastrigin	34684	381181	659551	14600	141949	967831
Griewank	21428	240413	411082	16921	180315	1277663
Levy5	44963	520083	867921	52824	563539	3933380
Rotated	7206	91897	158144	6007	69330	444215
Holder	642	9382	17492	507	6996	42525
Bird	8780	139203	212122	8140	111726	734121
Piccioni	22619	481950	946957	13924	127377	1109969
Shekel	401	5335	13284	415	5268	36305
	297420	4405849	7874560	**249674**	**2749359**	18306741

Table 11.7: Results from typical distance clustering global optimization algorithm

	Backtracking + Armijo			Proposed Local Search		
	nloc	nliter	funcalls	nloc	nliter	funcalls
Ackley	1482	17721	105300	1253	11621	134148
Giunta	1754	22627	101070	1018	11597	114404
Guillin Hills	7833	135865	246273	6712	116422	295439
Levy3	7550	76766	491131	5811	58747	687385
Rastrigin	8655	90680	635306	5807	52862	628857
Griewank	9254	94480	578331	7975	77468	903184
Levy5	8628	94510	648951	7255	73603	935423
Rotated	4046	52795	262429	3666	43508	418137
Holder	469	8547	39160	300	5324	46395
Bird	6928	109437	476559	7266	87867	907879
Piccioni	13208	287511	1230074	11839	106798	1525484
Shekel	490	6551	44156	357	4397	50250
DeJong	51333	1896462	4443491	6787	240666	1856475
Lagermann	20697	234094	1103016	24797	281443	2453179
Tube1	max	9664392	38755168	max	8835538	84807362
Michalewicz	420440	9803089	24960763	420901	1,1E+07	59374939
	562767	22595527	74121178	511744	20790077	155138940

Chapter 12

Stopping rules

For a broad class of global optimization problems, it can never be verified in finite time that the global optimum is identified with certainty. Therefore a need emerges for stopping rules which decide if the expected benefit of further searching outweighs the required computational effort.

Stopping rules have to decide for the path between the Scylla of computational efficiency and the Charybdis of the completeness warranty. In other words their objective is to collect the complete set of the existent local minima with the least computational effort. The ideal case would be to stop the search as soon as all the minima have been discovered. Since this is not possible, further searching is necessary to ensure that there are no left–out minima, a fact that inevitably leads to a compromise. So the stopping rules, depending on the specific problem at hand, negotiate either for efficiency or for a degree of completeness.

Some of the desirable properties of stopping rules ([Boender, Rinnooy Kan and Vercellis]) are:

1. *Sample dependent*: The actual objective function values and their location, or the number of times that local optima are identified by a local search procedure, should be taken into account by a decision rule to terminate a search.

2. *Problem dependent*: Maximal use should be made of available prior information. This information may concern, for instance, the number of local optima and the size of the regions of attraction, or the tail of the distribution of function values.

3. *Method dependent*: If some general algorithmic properties of the applied method are known, these should be incorporated in the stopping rule.

4. *Loss dependent*: Stopping rules should take into account the seriousness of the cost incurred if the search is terminated before the global optimum is identified.

5. *Resource dependent*: Evidently the computational effort should be kept as small as possible.

The above requirements can be met by postulating an appropriate probabilistic model of the sampling information.

12.1 Stopping rule for multistart-like algorithms

The task of locating all the local minima of a continuous function inside a box-bounded domain, is frequently required in several scientific as well as practical problems. We are interested in stochastic methods based on *Multistart*, a brief review of which follows.

The Multistart Algorithm

Step–0: Set $i = 0$ and $X^* = \emptyset$

Step–1: Sample x at random from S

Step–2: Apply a deterministic local search procedure (LS) starting at x and concluding at a local minimum x^*.

Step–3: Check if a new minimum is discovered
If $x^* \notin X^*$ then

increment: $i \leftarrow i + 1$

set: $x_i^* = x^*$

add: $X^* \leftarrow X^* \cup \{x_i^*\}$
Endif

Step–4: If a stopping rule applies, STOP

Step–5: Go to Step-1

Good stopping rules (Step–5)are important and should combine reliability and economy. A reliable rule is one that stops only when all minima have been collected with certainty. An economical rule is one that does not waste a large number of local searches to detect that all minima have been found. Several stopping rules have been developed in the past, most of them based on Bayesian considerations ([166, 12, 10, 14]) and they have been successfully used in practical applications. A review analyzing the topic of stopping rules is given in the book by *Törn* and *Žilinskas* ([149]). We refer also to Hart ([68]) noting however that his stopping rules aim to terminate the search as soon as possible once the global minimum is found and they are not designed for the retrieval of all the local minima. In this chapter we present a new stopping rule based on an a-priori hypothesis concerning local minima. This hypothesis render the stopping criterion most suitable for stochastic clustering global optimization methods. It would be helpful at this point to

state a few definitions and terms to be used in the rest of the article. Let w be the number of minima of a given function, k be the number of local searches of a multistart-like (i.e. clustering) algorithm and m be the number or recovered minima up to the $k-$th local search.

12.2 Widely used Stopping Rules

If by w we denote the number of recovered local minima after having performed a number of t local searches, then the estimate of the fraction of the uncovered space is given by ([166]):

$$P(w) = \frac{w(w+1)}{t(t-1)}. \tag{12.1}$$

The corresponding rule is then:

$$\text{Stop when } P(w) \leq \epsilon \tag{12.2}$$

ϵ being a small positive number. Boender et al [12] showed that the estimated number of local minima is given by:

$$w_{\text{est}} = \frac{w(t-1)}{t-w-2} \tag{12.3}$$

and the associated rule becomes:

$$\text{Stop when } w_{\text{est}} - w \leq \frac{1}{2} \tag{12.4}$$

In another rule ([14]), the probability that all the local minima have been observed is given by:

$$\prod_{i=1}^{w} \left(\frac{t-1-i}{t-1+i} \right) \tag{12.5}$$

leading to the rule:

$$\text{Stop when } \prod_{i=1}^{w} \left(\frac{t-1-i}{t-1+i} \right) > \tau \tag{12.6}$$

τ tends to 1 from below.

12.2.1 Recent Stopping rules [85]

Double-box stopping rule The covered portion of the search domain is a key element in preventing wasteful applications of the local search procedure. A relative measure for the region that has been covered is given by:

$$C = \sum_{i=1}^{w} \frac{m(A_i)}{m(S)} \tag{12.7}$$

where w is the number of the local minima discovered so far. The rule would then instruct to stop further searching when $C \to 1$.

The quantity $\frac{m(A_i)}{m(S)}$ is not known and generally cannot be calculated, however asymptotically it can be approximated by the fraction $\frac{L_i}{L}$, where L_i is the number of points, started from which, the local search led to the local minimum x_i^*, and $L = \sum_{i=1}^{w} L_i$, is the total number of sampled points (or equivalently, the total number of local search applications). An approximation for C may then be given by:

$$C \simeq \tilde{C} = \sum_{i=1}^{w} \frac{L_i}{L} \tag{12.8}$$

However the quantity $\sum_{i=1}^{w} \frac{L_i}{L}$ is by definition equal to 1, and as a consequence the covered space can not be estimated by the above procedure. To circumvent this, a larger box S_2 is constructed that contains S and such that $m(S_2) = 2 \times m(S)$. At every iteration, 1 point in S is collected, by sampling uniformly from S_2 and rejecting points not contained in S. Let the number of points that belong to $A_0 \equiv S_2 - S$ be denoted by L_0. The total number of sampled points is then given by $L = L_0 + \sum_{i=1}^{w} L_i$ and the relative coverage may be rewritten as:

$$C = \frac{\sum_{i=1}^{w} m(A_i)}{m(S)} = 2 \sum_{i=1}^{w} \frac{m(A_i)}{m(S_2)} \tag{12.9}$$

The quantity $\frac{m(A_i)}{m(S_2)}$ asymptotically is approximated by $\frac{L_i}{L}$, leading to:

$$C \simeq \tilde{C} = 2 \sum_{i=1}^{w} \frac{L_i}{L} \tag{12.10}$$

After k iterations, let the accumulated number of points sampled from S_2 be M_k, k of which are contained in S. The quantity then: $\delta_k \equiv \frac{k}{M_k}$ has an expectation value $< \delta >_k = \frac{1}{k} \sum_{i=1}^{k} \delta_i$ that asymptotically, i.e. for large k, tends to $\frac{m(S)}{m(S_2)} = \frac{1}{2}$.

The variance is given by $\sigma_k^2(\delta) = < \delta^2 >_k - < \delta >_k^2$ and tends to zero as $k \to \infty$. This is a smoother quantity than $< \delta >_k$), and hence better suited for a termination criterion. We permit iterating without finding new minima until $\sigma^2(\delta) < p\sigma_{last}^2(\delta)$, where $\sigma_{last}(\delta)$ is the standard deviation at the iteration during which the most recent minimum was found, and $p \in (0, 1)$ is a parameter that controls the compromise between an exhaustive search ($p \to 0$) and a search optimized for speed ($p \to 1$). The suggested value for general use is $p = 0.5$. Hence the algorithm may be stated as :

1. Initially set $\alpha = 0$.
2. Sample from S_2 until a point falls in S as described above.
3. Calculate $\sigma^2(\delta)$.
4. Apply an iteration of Multistart (i.e. steps 2 and 3).
5. If a new minimum is found, set: $\alpha = p\sigma^2(\delta)$ and repeat from step 2.
6. STOP if $\sigma^2(\delta) < \alpha$, otherwise repeat from step 2.

The Observables Stopping Rule This scheme is based on probabilistic estimates for the number of times each of the minima is being rediscovered by the local search. Let $L_1, L_2, \cdots, L_w$ be the number of local searches that ended–up to the local minima $x_1^*, x_2^*, \cdots, x_w^*$ (indexed in order of their appearance). Let $m(A_1), m(A_2), \cdots, m(A_w)$ be the measures of the corresponding regions of attraction, and let $m(S)$, be the measure of the bounded domain S. x_1^* is discovered for the first time with one application of the local search. Let n_2 be the number of the subsequent applications of the local search procedure spent, until x_2^* is discovered for the first time. Similarly denote by $n_3, n_4, \cdots, n_w$ the incremental number of local search applications to discover $x_3^*, x_4^*, \cdots, x_w^*$, i.e., x_2^* is found after $1 + n_2$ local searches, x_3^* after $1 + n_2 + n_3$, etc. $n_2, n_3, \cdots$ are counted during the execution of the algorithm, i.e. they are observable quantities. Considering the above and taking into account that we sample points using a uniform distribution, the expected number $L_J^{(w)}$ of local search applications that have ended–up to x_J^* at the time when the w^{th} minimum is discovered for the first time, is given by:

$$L_J^{(w)} = L_J^{(w-1)} + (n_w - 1)\frac{m(A_J)}{m(S)}. \tag{12.11}$$

The apriori probability that a local search procedure starting from a point sampled at random, concludes to the local minimum x_J^* is given by the ratio $m(A_J)/m(S)$, while the posteriori probability (observed frequency) is correspondingly given by $L_J/\sum_{i=1}^{w} L_i$. On the asymptotic limit the posteriori reaches the apriori probability, which implies $m(A_i)/m(A_j) = L_i/L_j$, which in turn permits substituting in eq. (12.11) L_i in place of $m(A_i)$ leading to:

$$\begin{aligned} L_J^{(w)} &= L_J^{(w-1)} + (n_w - 1)\frac{L_J}{\sum_{i=1}^{w} L_i} \\ &= L_J^{(w-1)} + (n_w - 1)\frac{L_J}{\sum_{i=1}^{w} n_i} \end{aligned} \tag{12.12}$$

with $n_1 = 1$, $J \leq w - 1$ and $L_w^{(w)} = 1$. Now consider that after having found w minima, an additional number of K local searches are performed without discovering any new minima. We denote by $\mathcal{L}_J^{(w)}(K)$ the expected number of times the J^{th} minimum is found at that moment. One readily obtains:

$$\mathcal{L}_J^{(w)}(K) = \mathcal{L}_J^{(w)}(K-1) + \frac{L_J}{K + \sum_{i=1}^{w} n_i} \tag{12.13}$$

with $\mathcal{L}_J^{(w)}(0) = L_J^{(w)}$.

The quantity

$$E_2(w, K) \equiv \frac{1}{w}\sum_{J=1}^{w}\left(\frac{\mathcal{L}_J^{(w)}(K) - L_J}{\sum_{l=1}^{w} L_l}\right)^2 \tag{12.14}$$

tends to zero asymptotically, hence a criterion based on the variance $\sigma^2(E_2)$ may be stated as:

Stop if $\sigma^2(E_2) < p\sigma^2_{last}(E_2)$

where $\sigma^2_{last}(E_2)$ is the variance of E_2 calculated at the time when the last minimum was retrieved. The value of the parameter p has the same justification as in the Double Box rule and the suggested value is again $p = 0.5$, although the user may choose to modify it according to his needs.

The Expected Minimizers Stopping Rule This technique is based on estimating the expected number of existing minima of the objective function in the specified domain. The search stops when the number of recovered minima, matches this estimate. Note that the estimate is updated iteratively as the algorithm proceeds. Let P^l_m denote the probability that after m draws, l minima have been discovered. Here by "draw" we mean the application of a local search, initiated from a point sampled from the uniform distribution. Let also π_k denote the probability that with a single draw the minimum located at x^*_k is found. This probability is apriori equal to $\pi_k = \dfrac{m(A_k)}{m(S)}$. The P^l_m probability can be recursively calculated by:

$$P^l_m = \left(1 - \sum_{i=1}^{l-1} \pi_i\right) P^{l-1}_{m-1} + \left(\sum_{i=1}^{l} \pi_i\right) P^l_{m-1} \tag{12.15}$$

Note that $P^0_1 = 0,$ and $P^1_1 = 1$. Also $P^l_m = 0$ if $l > m$, $P^0_m = 0,\ \forall m \geq 1$. The rational for the derivation of eq. (12.15) is as follows. The probability that at the m^{th} draw l minima are recovered, is connected with the probabilities at the level of the $(m-1)^{th}$ draw, that either $l-1$ minima are found (and the l^{th} is found at the next, i.e. the m^{th}, draw) or l minima are found (and no new minimum is found at the m^{th} draw). The quantity $\sum_{i=1}^{l} \pi_i$ is the probability that one of the l minima is found in a single draw, likewise the quantity $1 - \sum_{i=1}^{l-1} \pi_i$ is the probability that none of the $l-1$ minima is found in a single draw. Combining these observations the recursion above is readily verified. Since P^l_m denote probabilities they ought obey the closure:

$$\sum_{l=1}^{m} P^l_m = 1. \tag{12.16}$$

To prove the above let us define the quantity $s_l = \sum_{i=1}^{l} \pi_i$. Perform a summation over l on both sides of eq. (12.15) and obtain:

$$\sum_{l=1}^{m} P^l_m = \sum_{l=1}^{m} P^{l-1}_{m-1} - \sum_{l=1}^{m} s_{l-1} P^{l-1}_{m-1} + \sum_{l=1}^{m} s_l P^l_{m-1} \tag{12.17}$$

Note that since $P^0_{m-1} = 0$ and $P^m_{m-1} = 0$ the last two sums in eq. (12.17) cancel, and hence we get: $\sum_{l=1}^{m} P^l_m = \sum_{l=1}^{m-1} P^l_{m-1}$. This step can be repeated to show that

$$\sum_{l=1}^{m} P^l_m = \sum_{l=1}^{m-1} P^l_{m-1} = \ldots = \sum_{l=1}^{m-k} P^l_{m-k} = \sum_{l=1}^{1} P^l_1 = P^1_1 = 1$$

The expected number of minima after m draws is then given by:

$$< L >_m \equiv \sum_{l=1}^{m} l P_m^l$$

and its variance by:

$$\sigma^2(L)_m = \sum_{l=1}^{m} l^2 P_m^l - \left(\sum_{l=1}^{m} l P_m^l \right)^2 \tag{12.18}$$

The quantities π_i are unknown apriori and need to be estimated. Naturally the estimation will improve as the number of draws grows. A plausible estimate $\pi_i^{(m)}$ for approximating π_i after m draws, may be given by:

$$\pi_i^{(m)} \equiv \frac{L_i^{(m)}}{m} \to \frac{m(A_i)}{m(S)} = \pi_i \tag{12.19}$$

where $L_i^{(m)}$ is the number of times the minimizer x_i^* is found after m draws. Hence eq. (12.15) is modified and reads:

$$P_m^l = \left(1 - \sum_{i=1}^{l-1} \pi_i^{(m-1)} \right) P_{m-1}^{l-1} + \left(\sum_{i=1}^{l} \pi_i^{(m-1)} \right) P_{m-1}^l \tag{12.20}$$

The expectation $< L >_m$ tends to w asymptotically. Hence a criterion based on the variance $\sigma^2(L)_m$, that asymptotically tends to zero, may be proper. Consequently, the rule may be stated as:

Stop if $\sigma^2(L)_m < p\sigma^2(L)_{last}$,

where again $\sigma^2(L)_{last}$ is the variance at the time when the last minimum was found and the parameter p is used in the same manner as before. The suggested value for p is again $p = 0.5$.

12.3 Proposed stopping rule idea

Suppose that one can calculate theoretically the relation between the number of recovered minima m and the number of local searches k for a problem that has w distinct local minima. Suppose that this is a relation of the sort

$$N \equiv N^{(k)}(w), \quad N \to w \text{ as } k \to \infty \tag{12.21}$$

Imagine now that one applies multistart-based algorithm and plots the number of recovered minima versus the number of local searches.

One then at the k_0-th local search, may compare the *experimental* curve with the *theoretical* one and find which w is the one that produces the best match. If this is possible then at k_0-th local search we will now the number of expected local minima and hence a very efficient stopping rule may emerge.

12.3.1 Setting up the problem

Initially we will consider the multistart process. Then the results will generalize with any multistart-like global optimization algorithm. A point is sampled from a uniform distribution and a local search follows that concludes to a local minimum. We model this problem of trying to collect all the local minima inside e region with the following one:

> *Consider a box containing w different balls. The balls are numbered sequentially $1, 2, 3, \ldots, w$. We pick a ball at random examine its number, and we put it back in the box. This is one iteration*[1]*. If the ball number has not been drawn previously we update the distinct ball count m, otherwise we don't.*

This problem is direct analogy to our original one. Suppose that at iteration k, the probability that m balls (minima) are found is denoted by $p_m^{(k)}$. Then the expected number of distinct balls is given by:

$$< N >^{(k)} = \sum_{i=1}^{k} i \cdot p_i^{(k)} = p_1^{(k)} + 2p_2^{(k)} + \cdots + kp_k^{(k)} \tag{12.22}$$

Naturally $p_i^{(k)}$ depends on w (the number of balls) and hence so will $< N >^{(k)}$.
The rule to estimate the expected number of balls then would be

$$\min_w \sum_{k=k_0}^{k_1} \left(N_{exp}^{(k)} - N_w^{(k)} \right)^2 \to w^* \tag{12.23}$$

It remains to find a way to calculate $p_i^{(k)}$.

12.3.2 Calculation of probabilities $p_i^{(k)}$

We will now switch to the original global optimization problem and try to develop a recursion for the calculation of $p_i^{(k)}$. The most obvious relation would be:

$$p_i^{(k+1)} = \alpha p_i^{(k)} + \beta p_{i-1}^{(k)} \tag{12.24}$$

The above translates as: The probability that at the $(k+1)-$th local search i minima are recovered is related to

- the probability that in the previous iteration ($k-$th), i minima were already recovered and in the $(k+1)-$th no new minimum is found (this is with probability α),
- the probability that in the $k-$th iteration $(i-1)$ minima were found and in the $(k+1)-$th iteration one more minimum (new) is found (with probability β).

[1] Local search in global optimization framework

The task of calculating $p_i^{(k)}$ is now reduced to the task of defining the probabilities α and β. The most simple and straightforward way is to make the following assumption.

> *The probability of locating a local minimum, among the w distinct ones, by applying a local search is $p = \frac{1}{w}$.*

or in more simple words

> *All minima are retrieved (by applying a local search) with uniform probability.*

The above assumption although it seems irrational in the multistart framework, it makes sense in the concept of stochastic clustering algorithms were we (optimally) aim to perform *one local search per minimum.*

Using then the uniform assumption we derive the following probabilities:

- $\alpha = \frac{i}{w}$
- $\beta = \frac{w-(i-1)}{w}$

12.3.3 An illustration of the criterion

We will present a simple run of the multistart algorithm involving our stopping criterion to show how the expected number of minima found coincides to the real number of distinct minima found. We use the Ackley'a function with 121 minima for this illustration. The expected number of minima found is calculated every $n_{chunk} = 100$ iterations. Every n_{chunk} iterations, we calculate the mean square error of the real number of distinct minima found at the i-th iteration vs. the expected number of minima. That is

$$d_{MSE} = \frac{1}{iter} \sum_{i=0}^{iter} \left(< N^{(i)} > - N_{found}^{(i)} \right)^2$$

The iterations are shown in Table 12.1:

An illustration of the above is shown in Figure 12.1. Real number of minima is plotted using the continuous line and the expected number of minima using the dotted line. Observe that at the 800-th an 900-th iteration the two curves begin to fit perfectly.

12.4 Experimental evaluation

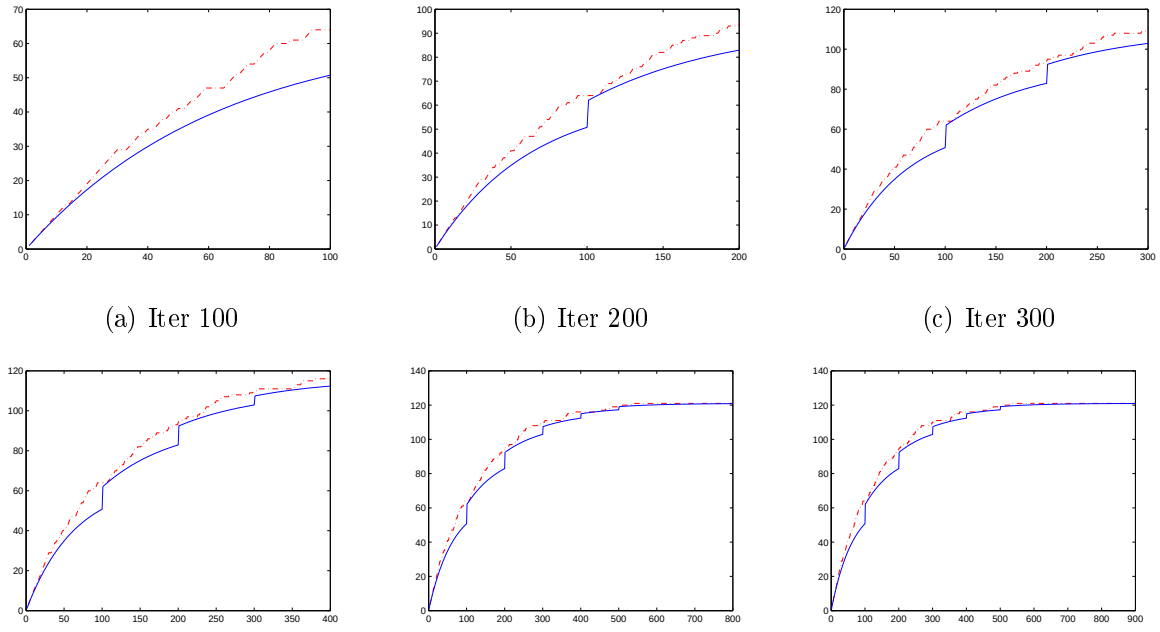
In order to test the efficiency of our proposed stopping criterion we have test it against three well established rules: Zielninksi's rule presented in Equation 12.2, Rinnoy-Kan rule in Equation 12.4 and Tsoulos-Lagaris double box rule.

All rules were implemented in Matlab and tested in a simple multistart framework. We expect better behavior when the proposed stopping rule would be included in a method that tries to equalize the probabilities of finding a minimum such as Minfinder or Adapt. The rules were tested using their default parameters,

Table 12.1: The MSE of the expected number of minima vs. the real minima found and its variance

Iteration	Minima Found	MSE	Variance
100	64	78.949828	205.044716
200	94	72.086818	3 6.688531
300	110	52.139962	9.248131
400	116	28.281410	2.063600
500	119	10.676961	0.443810
600	121	9.770836	0.095597
700	121	5.789973	0.018181
800	121	2.525016	0.003458
900	121	1.101163	0.000658

Figure 12.1: Illustration of the approximation of the expected number to the real number of minima



(a) Iter 100 (b) Iter 200 (c) Iter 300

(d) Iter 400 (e) Iter 800 (f) Iter 900

The test-best consists of 18 highly multimodal test functions that are commonly used in the bibliography. Each, experiment was conducted twenty times and the mean numbers of local minima, of local searches and total function calls is reported. First order derivatives are employed and are included in the total function call counter. The results are reported in Table 12.2. It is clear from the results that for the function with uniformly distributed minima of equal regions of attraction (Ackley, Rastrigin, Griewank, Tube, Holder, Piccioni etc.) our stopping rule is superior, since it performs the minimum number of local searches.

On the other hand, from the results of Rastrigin and Griewank, it is also obvious that the proposed stopping rule depends solely on the distribution and the number of local minima. Since the number of minima was almost equal for these functions and they are distributed uniformly in the search space, our stopping rule reached exactly 1500 local searches for both cases.

Table 12.2: Stopping rule results

Function	Zielinski			Rinnoy-Kan			Tsoulos-Lagaris			Proposed		
Rastrigin(121)	121	3843	66863	121	14886	254412	121	2129	36903	121	1500	25905
Ackley(49)	49	1566	42498	49	2502	67686	48.6	1079	29081	48.6	615	16457
Griewank(123)	123	3906	66436	123	15378	261801	123	1842	31414	123	1500	25742
Levy 3(130)	130	4128	79330	130	17163	329956	130	2078	40043	130	1605	30877
Levy 5(130)	130	4128	92642	130	17163	383216	130	2206	49075	130	1605	35718
Lagermann(64)	63.8	2035	39807	64	4227	82401	63.95	2859	55803	62.8	845	16607
R-Gaussians(94)	92.65	2947	78023	93.2	8875.8	235069	92.6	4503	119098	91.65	1185	31340
Giunta(36)	36	1155	17007	36	1371	20208	35.95	432	6405	36	500	7324
Guillin Hills	303.45	9612	190830	369.9	20000	396869	371	20000	396951	369.7	20000	396951
M0(152)	151.45	4806	71885	154.45	20000	299674	152.55	11265	168760	152	1920	28989
M5(441)	440.9	13959	1215763	440.85	20000	1742052	440.85	18623	1621518	439	5800	505057
Shekel (10)	10	333	10183	10	123	3776	10	158.2	4808,25	10	200[a]	6064
Bird(25)	24.95	805	22764	24.8	668	19019	22.8	659	18639	24.85	420	12025
Tube1(45)	45	1440	18457	45	2118	27140	45	483	6174	45	600	7863
Dejong(64)	62.65	1997	103142	63	4099	211941	62.2	1134	58701	63	800	41508
Holder(180)	180	5709	111733	180	20000	391459	180	2881	56357	180	2300	45052
Piccioni(37)	37	1187	25792	37	1446	31472	37	1036	22457	37	520	10997

[a]The minimum number of local searched needed to start evaluate our stopping rule

Chapter 13

Appendix - Test Functions

13.1 Ackley's test function ([1])

The number of existing minima in $[-5,5]^2$ is 121.

$$f(x) = -\alpha e^{-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n} cos(cx_i)} - \alpha e^1$$

$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = \frac{\alpha\ b\ x_i\ e^{-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}}}{n\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} + \frac{c\ sin(cx_i)e^{\frac{1}{n}\sum_{i=1}^{n} cos(cx_i)}}{n}$$

13.2 Bird's test function ([104])

This function has 173 minima in $[-50,50]^2$.

$$f(x_1,x_2) = sin(x_1)\ e^{(1-cos(x_2))^2} + cos(x_2)\ e^{(1-sin(x_1))^2} + (x(1)-x(2))^2$$

$$g_1(x) = \frac{\partial f(x)}{\partial x_1} = cos(x_1)\ e^{(1-cos(x_2))^2} - 2cos(x_2)(1-sin(x1))cos(x_1)e^{(1-sin(x_1))^2} + 2(x_1-x_2)$$

$$g_2(x) = \frac{\partial f(x)}{\partial x_2} = 2sin(x_1)(1-cos(x_2))sin(x_2)e^{(1-cos(x_2))^2} - sin(x_2)e^{(1-sin(x_1))^2} - 2(x_1-x_2)$$

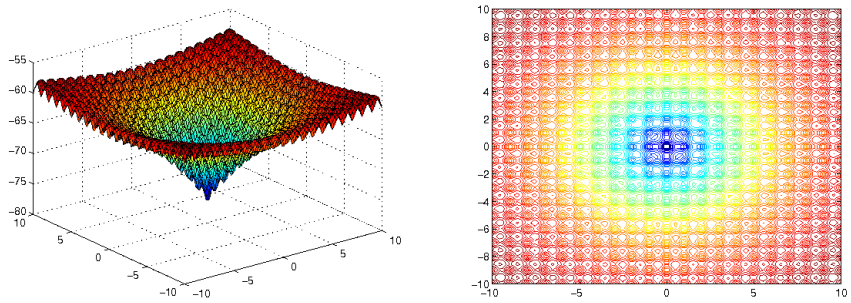
13.3 Bohachevsky 's test function ([15])

This function has 25 minima in $[-10,10]^2$

$$f(x_1,x_2) = x_1^2 + 2x_2^2 - 0.3\ cos(3\pi x_1) - 0.4\ cos(4\pi x_2) + 0.7$$

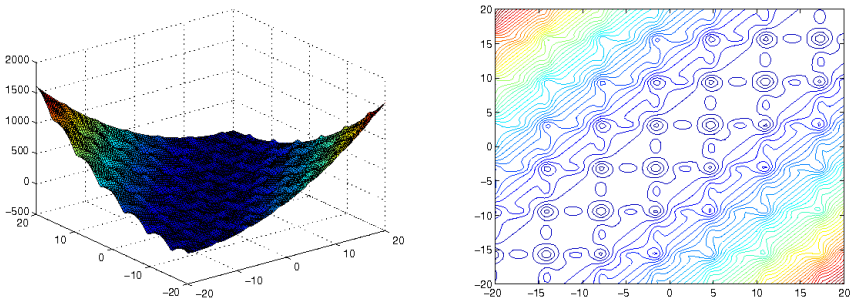
$$g_1(x) = \frac{\partial f(x)}{\partial x_1} = 2x_1 + \frac{9\ \pi}{10} sin(3\ \pi x_1)$$

$$g_2(x) = \frac{\partial f(x)}{\partial x_2} = 4x_2 + \frac{8\ \pi}{5} sin4\ \pi x_2$$



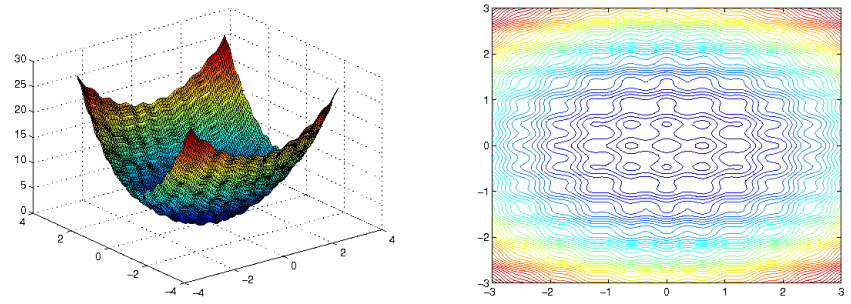
(g) Surface plot (h) Contour plot

Figure 13.1: Ackley's test function



(a) Surface plot (b) Contour plot

Figure 13.2: Birds's test function



(a) Surface plot (b) Contour plot

Figure 13.3: Bohachevsky's test function

13.4 Carrom table test function ([104])

This function has 169 minima in $[-5,5]^2$

$$f(x_1,x_2) = -\frac{1}{30}\left(\cos x_1\ \cos x_2\ e^{\left(1-\frac{\left(x_1^2+x_2^2\right)^{0.5}}{\pi}\right)}\right)^2$$

$$\begin{aligned} g_1(x) &= \frac{\partial f(x)}{\partial x_1} = \frac{\cos x_1 \cos^2 x_2}{15}\left(\sin x_1\ e^{\left(1-\frac{\left(x_1^2+x_2^2\right)^{0.5}}{\pi}\right)^2} + \frac{\cos x_1\ e^{\left(1-\frac{\left(x1^2+x2^2\right)^{0.5}}{\pi}\right)^2}}{\pi\ x_1\ \sqrt{x_1^2+x_2^2}}\right) \\ g_2(x) &= \frac{\partial f(x)}{\partial x_2} = \frac{\cos x_2 \cos^2 x_1}{15}\left(\sin x_2\ e^{\left(1-\frac{\left(x_1^2+x_2^2\right)^{0.5}}{\pi}\right)^2} + \frac{\cos x_2\ e^{\left(1-\frac{\left(x1^2+x2^2\right)^{0.5}}{\pi}\right)^2}}{\pi\ x_2\ \sqrt{x_1^2+x_2^2}}\right) \end{aligned}$$

13.5 Giunta's test function ([54])

This test function has 196 minima inside $[-20,20]^2$.

$$f(x_1,x_2) = 0.6 + \sin y_1 + \sin^2 y_1 + \frac{1}{50}\sin 4y_1 + \sin y_2 + \sin^2 y_2 + \frac{1}{50}\sin 4y_2$$

where $y_1 = \frac{16}{15}x_1 - 1$ and $y_2 = \frac{16}{15}x_2 - 1$.

$$\begin{aligned} g_1(x) &= \frac{\partial f(x)}{\partial x_1} = \frac{16}{15}\cos y_1 + \frac{32}{15}\sin y_1 \cos y_1 + \frac{32}{375}\cos 4y_1 \\ g_2(x) &= \frac{\partial f(x)}{\partial x_2} = \frac{16}{15}\cos y_2 + \frac{32}{15}\sin y_2 \cos y_2 + \frac{32}{375}\cos 4y_2 \end{aligned}$$

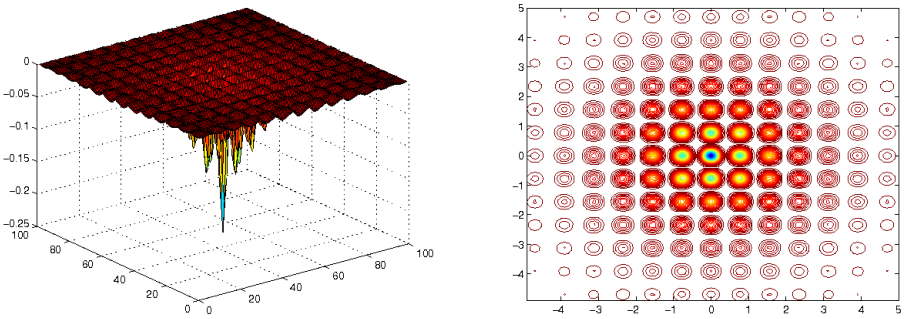
where $y_1 = \frac{16}{15}x_1 - 1$ and $y_2 = \frac{16}{15}x_2 - 1$.

13.6 Griewank's test function ([63])

This function has 529 minima inside $[-100,100]^2$.

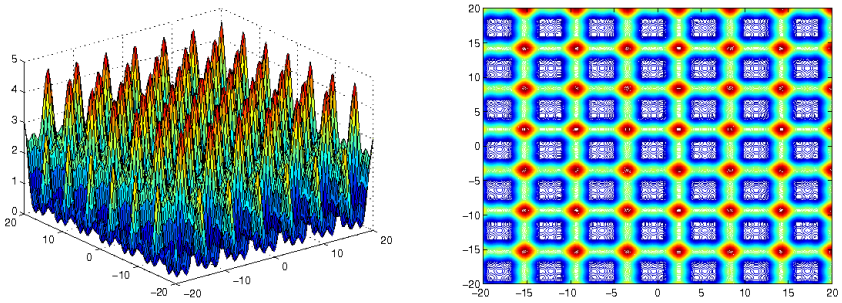
$$f(x) = \frac{1}{200}\sum_{i=0}^{n} x_i^2 - \prod_{i=1}^{n}\cos\frac{x_i}{\sqrt{i}} + 1$$

$$g_i(x) = \frac{\partial f(x)}{\partial x_1} = \frac{2\ x_i}{4000} + \frac{1}{\sqrt{i}}\sin\left(\frac{x_i}{\sqrt{i}}\right)\prod_{k=1,k\neq i}^{n} cos\frac{x_k}{\sqrt{k}}$$



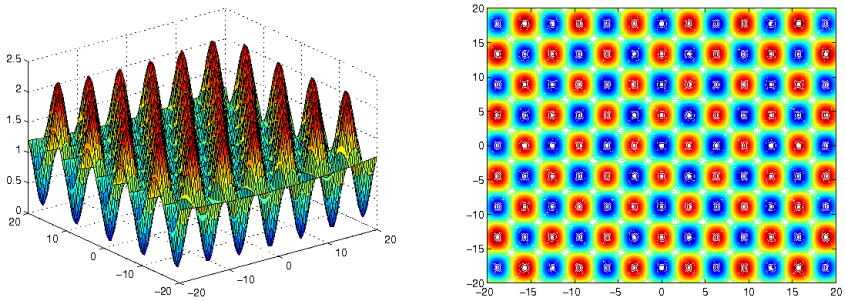
(a) Surface plot (b) Contour plot

Figure 13.4: Carrom table test function



(a) Surface plot (b) Contour plot

Figure 13.5: Giunta's test function



(a) Surface plot (b) Contour plot

Figure 13.6: Griewanks's test function

13.7 Guillin Hills's test function ([151])

This test function possesses 25 minima inside $[0,1]^2$.

$$f(x) = 3 + \sum_{i=1}^{n} \frac{c_i(x_i+9)}{x_i+10} \sin\left(\frac{\pi}{1-x_i+\frac{1}{2k}}\right)$$

where $c_i = 2,\ i = 1,\ldots,n$ and $k = 5$.

$$\begin{aligned} g_i(x) = \frac{\partial f(x)}{\partial x_i} &= \frac{c_i}{x_i+10}\sin\left(\frac{\pi}{1-x_i+\frac{1}{2k}}\right) \\ &- \frac{c_i(x_i+9)}{(x_i+10)^2}\sin\left(\frac{\pi}{1-x_i+\frac{1}{2k}}\right) \\ &+ \frac{c_i(x_i+9)}{x_i+10}\frac{\pi}{\left(1-x_i+\frac{1}{2k}\right)^2}\cos\left(\frac{\pi}{1-x_i+\frac{1}{2k}}\right) \end{aligned}$$

where $c_i = 2,\ i = 1,\ldots,n$ and $k = 5$.

13.8 Holder test function ([104])

This function has 85 minima inside $[-20,20]^2$.

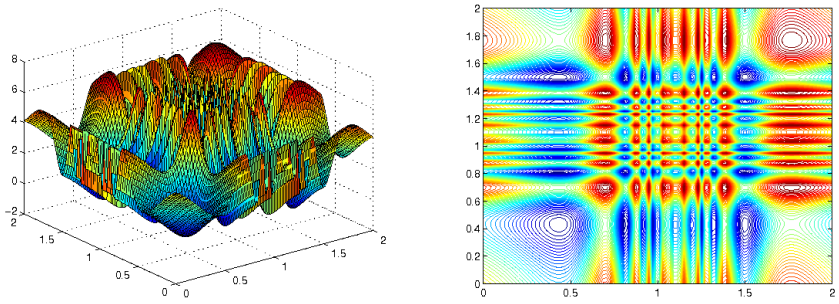
$$f(x_1,x_2) = -\cos x_1 \cos x_2 e^{1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}}$$

$$\begin{aligned} g_1(x) = \frac{\partial f(x)}{\partial x_1} &= \sin x_1\ \cos x_2\ e^{1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}} \\ &+ \cos x_1\ \cos x_2\ \frac{x_1 e^{1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}}}{\pi\sqrt{x_1^2+x_2^2}} \\ g_2(x) = \frac{\partial f(x)}{\partial x_2} &= \cos x_1\ \sin x_2\ e^{1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}} \\ &+ \cos x_1\ \cos x_2\ \frac{x_2 e^{1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}}}{\pi\sqrt{x_1^2+x_2^2}} \end{aligned}$$

13.9 Langermanns's test function ([122])

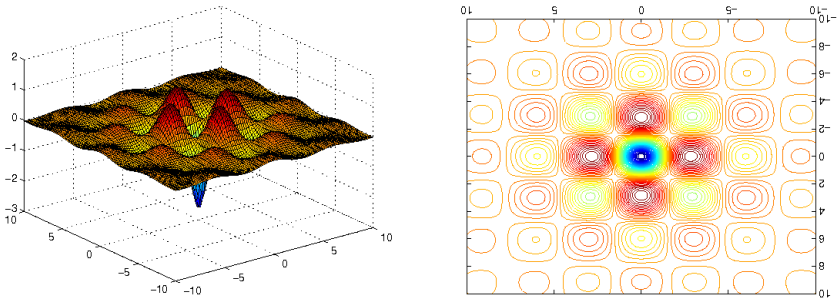
This test function has 270 minima inside $[0,7]^2$.

$$f(x_i) = \sum_{k=0}^{5} c_k e^{\sigma_k}\ \cos\lambda_k$$



(a) Surface plot (b) Contour plot

Figure 13.7: Guillin Hills test function



(a) Surface plot (b) Contour plot

Figure 13.8: Holder-like test function

In current implementation $a = (3, 5, 2, 1, 7)^T$, $c = (1, 2, 5, 2, 3)^T$

$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = \sum_{k=0}^{5} \left(\frac{-2c_k(x_i - a_k)}{\pi} \; e^{\sigma_k} \; \cos \lambda_k - 2c_i \pi (x_i - a_k) \; e^{\sigma_k} \; \sin \lambda_k \right)$$

where $\sigma_k = \sum_{i=1}^{n} -\frac{(x_i - a_k)^2}{\pi}$ and $\lambda_k = \sum_{i=1}^{n} \pi (x_i - a_k)^2$.

13.10 Levy's 3rd test function ([88])

This test function has 527 minima inside $[-10, 10]^2$.

$$f(x_1, x_2) = \sum_{k=1}^{5} k \cos\left((k-1)x_1 + k\right) \sum_{k=1}^{5} k \cos\left((k+1)x_2 + k\right)$$

$$\begin{aligned} g_1(x) = \frac{\partial f(x)}{\partial x_1} &= \sum_{k=1}^{5} -k(k-1) \sin\left((k-1)x_1 + k\right) \sum_{k=1}^{5} k \cos\left((k+1)x_2 + k\right) \\ g_2(x) = \frac{\partial f(x)}{\partial x_2} &= \sum_{k=1}^{5} -k(k+1) \sin\left((k+1)x_2 + k\right) \sum_{k=1}^{5} k \cos\left((k-1)x_1 + k\right) \end{aligned}$$

13.11 Levy's 5th test function ([88])

This test function has 508 minima inside $[-10, 10]^2$.

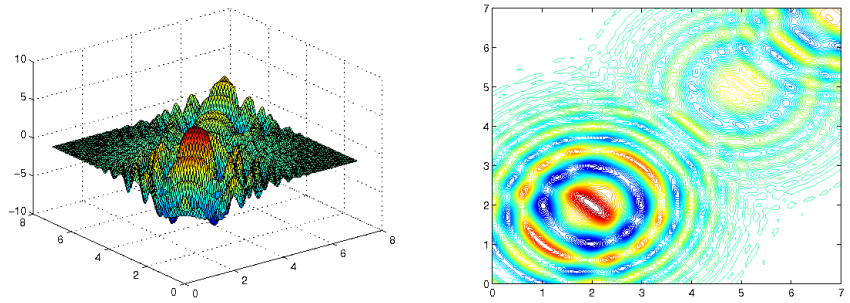
$$f(x_1, x_2) = f_{Levy3}(x_1, x_2) + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2$$

$$\begin{aligned} g_1(x) = \frac{\partial f(x)}{\partial x_1} &= \frac{\partial f_{Levy3}(x)}{\partial x_1} + 2(x_1 + 1.42513) \\ g_2(x) = \frac{\partial f(x)}{\partial x_2} &= \frac{\partial f_{Levy3}(x)}{\partial x_2} + 2(x_2 + 0.80032) \end{aligned}$$

13.12 Liang's test function [90]

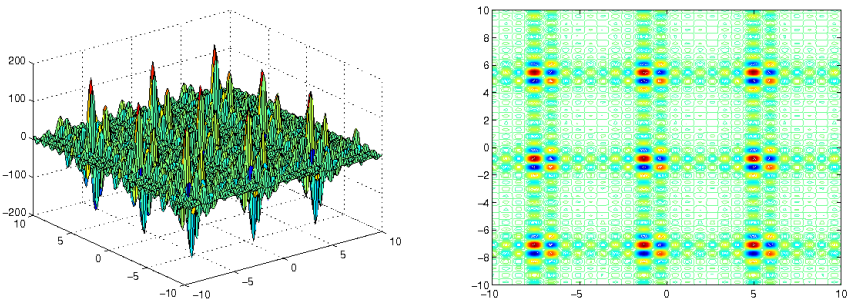
This test function has 236 local minima inside $[1, 4]^2$.

$$\begin{aligned} f(x_1, x_2) = &- \left(x_1 \sin(20x_2) + x_2 \sin(20x_1)\right)^2 \cosh\left(\sin(10x_1)x_1\right) \\ &- \left(x_1 \cos(20x_2) - x_2 \sin(10x_1)\right)^2 \cosh\left(\cos(10x_2)x_2\right) \end{aligned}$$



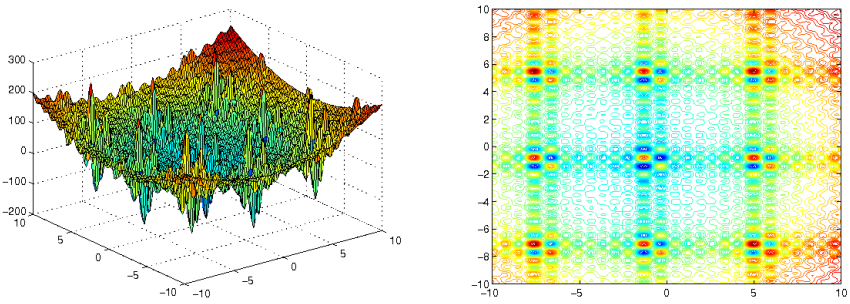
(a) Surface plot (b) Contour plot

Figure 13.9: Lagermanns's test function



(a) Surface plot (b) Contour plot

Figure 13.10: Levy's No 3 test function



(a) Surface plot (b) Contour plot

Figure 13.11: Levy's No 5 test function

$$
\begin{aligned}
g_1(x) = \frac{\partial f(x)}{\partial x_1} &= -2(x_1 \sin(20x_2) + x_2 \sin(20x_1))\ \cosh(\sin(10x_1)x_1)\ (\sin(20x_2) + 20x_2 \cos(20x_1)) \\
&- (x_1 \sin(20x_2) + x_2 \sin(20x_1))^2 \sinh(\sin(10x_1)x_1)(10\cos(10x_1)x_1 + \sin(10x_1)) \\
&- 2(x_1 \cos(10x_2) - x2 \sin(10x_1)) \cosh(\cos(20x_2)x_2)(\cos(10x_2) - 10x_2 \cos(10x_2)) \\
g_2(x) = \frac{\partial f(x)}{\partial x_2} &= -2(x_1 \sin(20x_2) + x_2 \sin(20x_1)) \cosh(\sin(10x_1)x_1)(20x_1 \cos(20x_2) + \sin(20x_1)) \\
&- 2(x_1 \cos(10x_2) - x_2 \sin(10x_1)) \cosh(\cos(20x_2)x_2)(-10x_1 sin(10x_2) - sin(10x_1)) \\
&- (x_1 \cos(10x_2) - x_2 \sin(10x_1))^2 \sinh(\cos(20x_2)x_2)(-20\sin(20x_2)x_2 + \cos(20x_2))
\end{aligned}
$$

13.13 Piccioni's test function ([94])

This test function has 28 minima inside $[-5, 5]^2$.

$$f(x) = -10 \sin(\pi x_1)^2 - \sum_{i=1}^{n-1} (x_i - 1)^2 (1 + 10 \sin(\pi x_{i+1})) - (x_n - 1)^2$$

$$
\begin{aligned}
g_1(x) &= \frac{\partial f(x)}{\partial x_1} = -20\pi \sin(\pi x_1) \cos(\pi x_1) - 2(x_1 - 1) \\
g_i(x) &= \frac{\partial f(x)}{\partial x_i} = -2(x_i - 1)(1 + 10 \sin(\pi x_{i+1})) - (x_{i-1} - 1)^2 10\pi \cos(\pi x_i), \quad i = 2 \ldots n-2 \\
g_n(x) &= \frac{\partial f(x)}{\partial x_n} = -(x_{n-1} - 1)^2 10\pi \cos(\pi x_n) - 2(x_n - 1)
\end{aligned}
$$

13.14 Rastrigin's test function ([130])

This test function has 49 minima inside $[-1, 1]^2$.

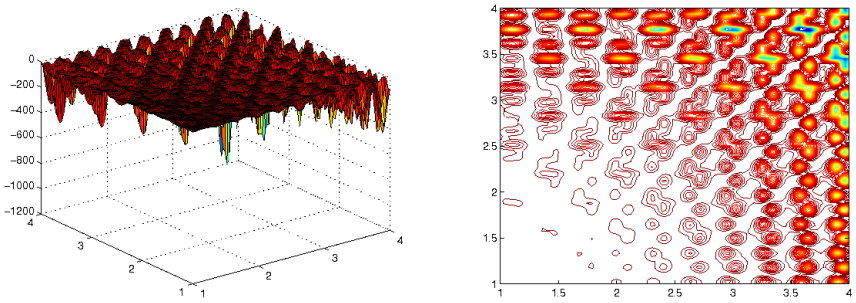
$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right)$$

$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = \frac{\partial f(x)}{\partial x_i} = 2x_i + 20\pi \sin(2\pi x_i)$$

13.15 Voglis's Test Function

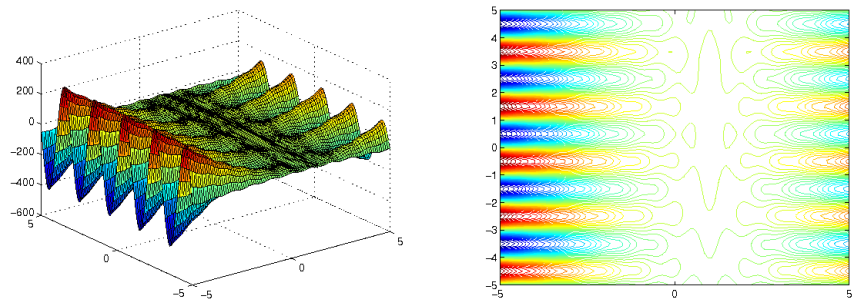
This test function has 61 minima inside $[-25, 25]^2$.

$$f(x) = \alpha_0 \left(\frac{1}{2} x^T Q_0 x + x^T d_0\right) + \sum_{i=1}^{80} \alpha_k e^{-\frac{1}{2} x^T Q_k x + x^T d_k}$$



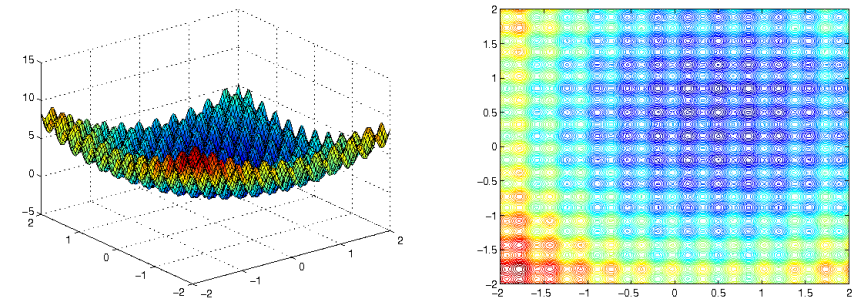
(a) Surface plot (b) Contour plot

Figure 13.12: Liangs's test function



(a) Surface plot (b) Contour plot

Figure 13.13: Piccioni's test function



(a) Surface plot (b) Contour plot

Figure 13.14: Rastrigin's test function

$$g_i(x) = \frac{\partial f(x)}{x_i} = \alpha_0 \left(Q_0 x + d_0\right) + \sum_{i=1}^{80} \alpha_k \left(-Q_k x + d_k\right) e^{-\frac{1}{2} x^T Q_k x + x^T d_k}$$

Function dimension $n = 2$, Q_j specific positive definite $2x2$ matrices, d_j 2-dimensional vectors and α_j appropriate scaling constants.

13.16 Schaffer's Test Function ([104])

This test function has 95 minima inside $[-3, 3]^2$.

$$f(x_1, x_2) = 0.5 + \frac{\sin(x_1^2 + x_2^2)^2 - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2} + 0.1 \sin(10 x_1) + 0.1 \sin(10 x_2)$$

$$\begin{aligned} g_1(x) &= \frac{\partial f(x)}{\partial x_1} = 4x_1 \frac{\sin(x_1^2 + x_2^2)\cos(x_1^2 + x_2^2)}{(1 + 0.001x_1^2 + 0.001x_2^2)^2} - 0.004x_1 \frac{sin(x_1^2 + x_2^2)^2 - 0.5}{(1 + 0.001x_1^2 + 0.001x_2^2)^3} \\ g_2(x) &= \frac{\partial f(x)}{\partial x_2} = 4x_2 \frac{\sin(x_1^2 + x_2^2)\cos(x_1^2 + x_2^2)}{(1 + 0.001x_1^2 + 0.001x_2^2)^2} - 0.004x_2 \frac{sin(x_1^2 + x_2^2)^2 - 0.5}{(1 + 0.001x_1^2 + 0.001x_2^2)^3} \end{aligned}$$

13.17 Shubert's Test Function ([142])

This test function has 400 minima inside $[-10, 10]^2$.

$$f(x) = -\sum_{i=1}^{n} \sum_{j=1}^{5} j \sin((j+1)x_i + j)$$

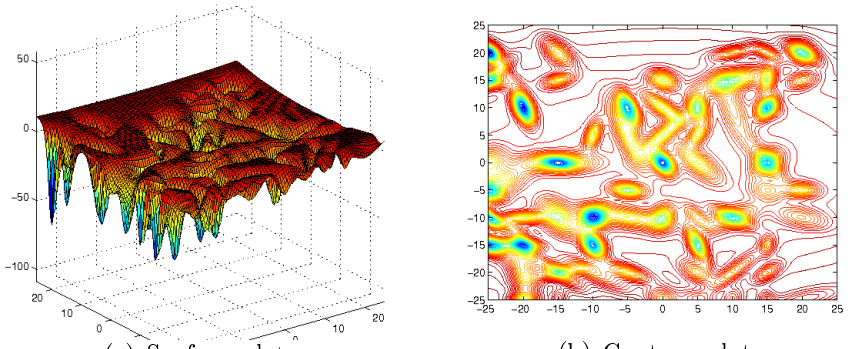
$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = -\sum_{j=1}^{5} j(j+1) \cos((j+1)x_i + j)$$

13.18 M0 Test Function ([142])

This test function has 66 minima inside $[-5, 1]^2$.

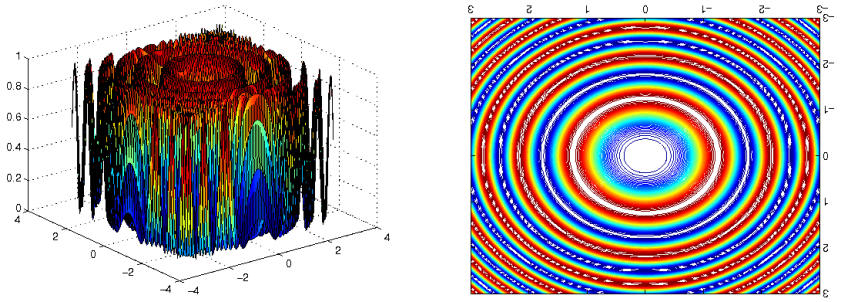
$$f(x) = \sin(2.2\pi x_1 + \frac{\pi}{2}) \frac{2 - x_2}{2} \frac{3 - x_1}{2} + \sin(\frac{\pi}{2} x_2^2 + \frac{\pi}{2}) \frac{2 - x_2}{2} \frac{3 - x_1}{2}$$

$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = -\sum_{j=1}^{5} j(j+1) \cos((j+1)x_i + j)$$



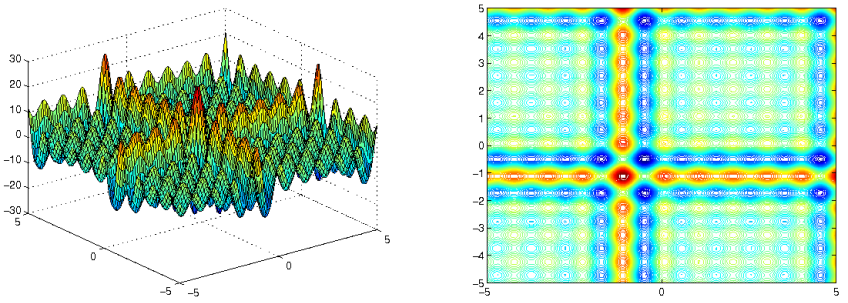
(a) Surface plot (b) Contour plot

Figure 13.15: Voglis 's test function



(a) Surface plot (b) Contour plot

Figure 13.16: Schaffer's test function



(a) Surface plot (b) Contour plot

Figure 13.17: Shubert's test function

13.19 M3 Test Function ([142])

This test function has 26 minima inside $[-2, 2]^2$.

$$f(x) = -(x_2^2 - 4.5x_2^2)x_1x_2 - 4.7\cos(3x_1 - x_2^2(2 + x_1))\sin(2.5\pi * x_1) + (0.3 * x1)^2$$

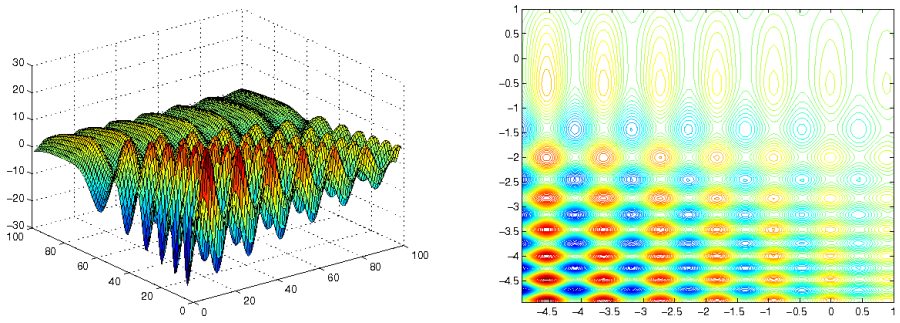
$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = -\sum_{j=1}^{5} j(j+1)\cos((j+1)x_i + j)$$

13.20 Siam Problem 4 Function ([143])

This test function has 600 minima inside $[-1, 1]^2$.

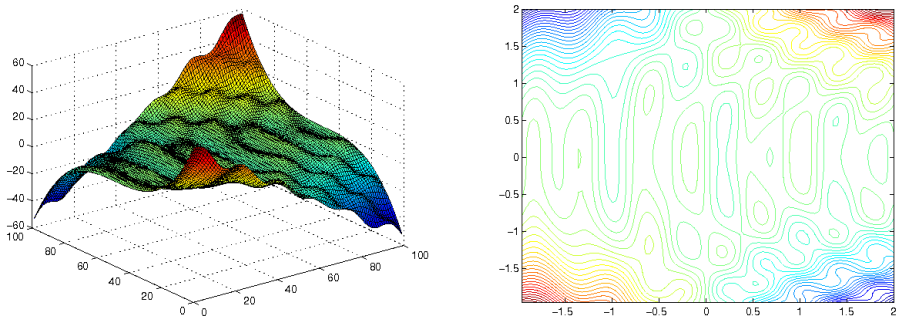
$$f(x) = \exp(\sin(x_1)) + \sin(60\exp(x_2)) + \sin(70\sin(x_1)) + \sin(\sin(80x_2)) - \sin(10(x_1 + x_2)) + \frac{x_1^2 + x_2^2}{4};$$

$$g_i(x) = \frac{\partial f(x)}{\partial x_i} = -\sum_{j=1}^{5} j(j+1)\cos((j+1)x_i + j)$$



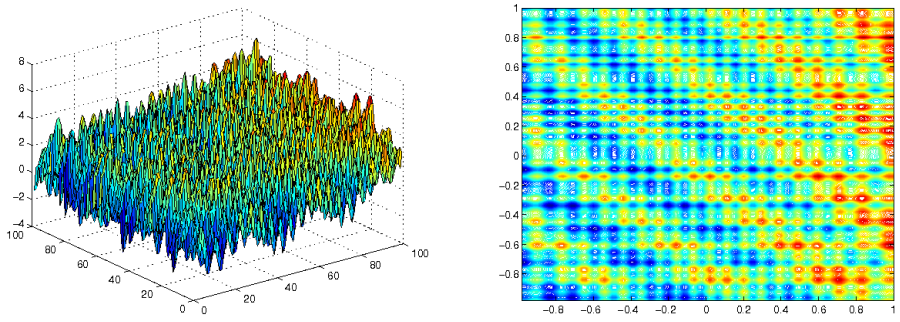
(a) Surface plot (b) Contour plot

Figure 13.18: M0 test function



(a) Surface plot (b) Contour plot

Figure 13.19: M3 test function



(a) Surface plot (b) Contour plot

Figure 13.20: Siam Problem 4 test function

Bibliography

[1] D.H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, 1987.

[2] M. Al-Baali and R. Fletcher. An efficient line search for nonlinear least squares. *Journal of Optimization Theory and Applications*, 48(3):359–377, 1986.

[3] MM Ali and C. Storey. Topographical multilevel single linkage. *Journal of Global Optimization*, 5(4):349–358, 1994.

[4] T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press New York, 1996.

[5] M.S. Bazaraa, H.D. Sherali, and CM Shetty. *Nonlinear programming: theory and algorithms*. Wiley-Interscience, 2006.

[6] RW Becker and GV Lago. A global optimization algorithm. In *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, pages 3–12, 1970.

[7] D.P. Bertsekas. *Constrained Optimization and Lagrange Multipliers*. Academic Press, 1982.

[8] D.P. Bertsekas. *Nonlinear programming*. 1995.

[9] D. Bertsimas, C. Darnell, and R. Soucy. Portfolio construction through mixed-integer programming at Grantham, Mayo, Van Otterloo and Company. *Interfaces*, pages 49–66, 1999.

[10] B. Betro and F. Schoen. Optimal and sub-optimal stopping rules for the Multistart algorithm in global optimization. *Mathematical Programming*, 57(1):445–458, 1992.

[11] A. Bjorck and G. Dahlquist. *Numerical methods*. Prentice Hall Professional Technical Reference, 1990.

[12] C.G.E. Boender and A.H.G. Rinnooy Kan. Bayesian stopping rules for multistart global optimization methods. *Mathematical Programming*, 37(1):59–80, 1987.

[13] CGE Boender, AHG Rinnooy Kan, GT Timmer, and L. Stougie. A stochastic method for global optimization. *Mathematical programming*, 22(1):125–140, 1982.

[14] C.G.E. Boender and H.E. Romeijn. Stochastic methods. *Handbook of global optimization*, pages 829–869, 1995.

[15] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Generalized simulated annealing for function optimization. *Technometrics*, pages 209–217, 1986.

[16] Schutte J. F. Bolton, H. P. J. and Groenwold A. A. Multiple Parallel Local Searches in Global Optimization. *Mathematical Programming*, 1908:88–95, 2000.

[17] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, and S. Lucidi. A new version of the Price's algorithm for global optimization. *Journal of Global Optimization*, 10(2):165–184, 1997.

[18] S. Breedveld, P.R.M. Storchi, M. Keijzer, and B.J.M. Heijmen. Fast, multiple optimizations of quadratic dose objective functions in IMRT. *Physics in Medicine and Biology*, 51(14):3569–3580, 2006.

[19] R.P. Brent. *Algorithms for Minimization Without Derivatives*. Prentice Hall, 1973.

[20] CG Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[21] H.M. Bucker, A. Rasch, and A. Vehreschild. Automatic generation of parallel code for Hessian computations. *Lecture Notes in Computer Science*, 4315:372, 2008.

[22] U. Burkert and N.L. Allinger. *Molecular mechanics*. An American Chemical Society Publication, 1982.

[23] D.R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., 1997.

[24] Dutta P. Bandyopadhyay P. Sarkar P. Chaudhury, P. and S.P. Bhattacharyya. A random walk to local minima and saddle points on a potential energy surface. a strategy based on simulated annealing. *Chemical Physics Letters*, 1996.

[25] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in amultidimensional complex space. *IEEE transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[26] T.F. Coleman and J. Liu. An interior Newton method for quadratic programming. *Mathematical Programming*, 85(3):491–523, 1999.

[27] A.R. Conn and N.I.M. Gould. *Trust-region methods*. Society for Industrial Mathematics, 2000.

[28] A.R. Conn, N.I.M. Gould, and P.L. Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Springer, 1992.

[29] M. D Apuzzo, M. Marino, P.M. Pardalos, and G. Toraldo. A parallel implementation of a potential reduction algorithm for box-constrained quadratic programming. *Lecture notes in computer science*, pages 839–848, 2000.

[30] L. Dagum, R. Menon, and S.G. Inc. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.

[31] W.C. Davidon. Variable Metric Method For Minimization. Technical report, ANL-5990, Argonne National Lab., Lemont, Ill., 1959.

[32] L. Davis et al. *Handbook of genetic algorithms*. Van nostrand reinhold New York, 1991.

[33] Weintraub H.J.R. Demeter, D.A. and J.J. Knittel. The local minima method (lmm) of pharmacophore determination: A protocol for predicting the bioactive conformation of small, conformationally flexible molecules. *Journal of Chemical Information and Computer Sciences*, 1996.

[34] J.E. Dennis and H.H.W. Mei. Two new unconstrained optimization algorithms which use function and gradient values. *Journal of Optimization Theory and Applications*, 28(4):453–482, 1979.

[35] J.E. Dennis, R.B. Schnabel, and J.E. Dennis. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall Englewood Cliffs, NJ, 1983.

[36] R.A. Ding, Z.; Kennedy. On the whereabouts of local minima for blind adaptive equalizers. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 1992.

[37] LCW Dixon and M. Jha. Parallel algorithms for global optimization. *Journal of Optimization Theory and Applications*, 79(2):385–395, 1993.

[38] RC Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particleswarm optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, 2000.

[39] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[40] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Second Edition, 1987.

[41] R. Fletcher and M.J.D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163, 1963.

[42] R. Fletcher and CM Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

[43] R. Fletcher and C. Xu. Hybrid methods for nonlinear least squares. *IMA Journal of Numerical Analysis*, 7(3):371–389, 1987.

[44] C.A. Floudas. *Deterministic global optimization: theory, methods and applications*. Kluwer Academic Pub, 2000.

[45] C.A. Floudas and P.M. Pardalos. *A collection of test problems for constrained global optimization algorithms*. Springer, 1990.

[46] C.A. Floudas and P.M. Pardalos. *Optimization in computational chemistry and molecular biology: local and global approaches*. Kluwer Academic Pub, 2000.

[47] D.B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*. Wiley-IEEE Press, 2006.

[48] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8:159–416, 1994.

[49] A. Friedlander and J.M. Martínez. On the maximization of a concave quadratic function with box constraints. *SIAM Journal on Optimization*, 4:177, 1994.

[50] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU scientific library*. Network Theory Ltd., 2002.

[51] P. Gilbert and Varadhan R. The numderiv package. http://cran.r-project.org/web/packages/numDeriv/, 2006.

[52] P.E. Gill and W. Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Mathematical Programming*, 7(1):311–350, 1974.

[53] P.E. Gill, W. Murray, and M.H. Wright. *Practical optimization*. Academic Press San Diego, 1981.

[54] A.A. Giunta. *Aircraft multidisciplinary design optimization using design of experiments theory and response surface modeling methods*. PhD thesis, virginia polytechnic institute and state university, 1997.

[55] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[56] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, pages 23–26, 1970.

[57] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33, 1983.

[58] S.M. Goldfeld, R.E. Quandt, and H.F. Trotter. Maximization by quadratic hill-climbing. *Econometrica: Journal of the Econometric Society*, pages 541–551, 1966.

[59] A.A. Goldstein. On steepest descent. *J. Soc. Ind. Appl. Math., Ser. A: Control*, 3:147–151, 1965.

[60] J. Greenstadt. On the relative efficiencies of gradient methods. *Mathematics of Computation*, pages 360–367, 1967.

[61] A. Griewank and G.F. Corliss. Automatic Differentiation of Algorithms: Theory. *Implementation, and Application, SIAM, Philadelphia, Penn*, 1991.

[62] Andreas Griewank and George F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, 1991.

[63] AO Griewank. Generalized descent for global optimization. *Journal of optimization theory and applications*, 34(1):11–39, 1981.

[64] S.R. Gunn. Support vector machines for classification and regression. *ISIS technical report*, 14, 1998.

[65] PE Hadjidoukas. A Lightweight Framework for Executing Task Parallelism on Top of MPI. *Lecture notes in computer science*, pages 287–294, 2004.

[66] C.G. Han, P.M. Pardalos, and Y. Ye. Computational aspects of an interior point algorithm for quadratic programming problems with box constraints. *Large-Scale Numerical Optimization*, pages 92–112, 1990.

[67] W.E. Hart. *Adaptive global optimization with local search*. PhD thesis, UNIVERSITY OF CALIFORNIA, SAN DIEGO, 1994.

[68] W.E. Hart. Sequential stopping rules for random optimization methods with applications to multistart local search. *Siam Journal on Optimization*, 9:270–290, 1998.

[69] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand*, 49(6):409–436, 1952.

[70] D.M. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill Companies, 1972.

[71] J.H. Holland. *Adaptation in natural and artificial systems*. MIT Press Cambridge, MA, USA, 1992.

[72] R. Hooke and TA Jeeves. Direct Search Solution of Numerical and Statistical Problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[73] R. Horst and H. Tuy. *Global optimization: Deterministic approaches*. Springer, 1996.

[74] Voessner S. Iglehart, D. L. Optimization of a trading system using global search techniques and local optimization. *Journal of Computational Intelligence in Finance*.

[75] M. IMSL. LIBRARY User's Manual. *FORTRAN Subroutines for Mathematical Applications, IMSL Inc., Houston TX*, 1991.

[76] F. Jensen. *Introduction to Computational Chemistry*. John Wiley & Sons, 2006.

[77] Aaron Masino Justin D. Mansell1 and Brian Henderson. Study of local minima in metric adaptive optics. *Active Optical Systems*, 2004.

[78] C.T. Kelley. *Iterative methods for optimization*. Society for Industrial Mathematics, 1999.

[79] I.R. Khan and R. Ohba. New finite difference formulas for numerical differentiation. *Journal of Computational and Applied Mathematics*, 126(1-2):269–276, 2000.

[80] K. Kunisch and F. Rendl. An Infeasible Active Set Method for Quadratic Problems with Simple Bounds. *SIAM Journal on Optimization*, 14:35, 2003.

[81] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural network methods in quantum mechanics. *Computer Physics Communications*, 104(1-3):1–14, 1997.

[82] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partialdifferential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[83] I.E. Lagaris, A. Likas, and D.G. Papageorgiou. Neural-network methods for boundary value problems with irregularboundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

[84] IE Lagaris and VR Pandharipande. Phenomenological two-nucleon interaction operator. *Nuclear Physics, Section A*, 359(2):331–348, 1981.

[85] I.E. Lagaris and I.G. Tsoulos. Stopping rules for box-constrained stochastic global optimization. *Applied Mathematics and Computation*, 197(2):622–632, 2008.

[86] A. Leshem and A.-J. van der Veen. Blind source separation: The location of local minima in the case of finitely many samples. *IEEE Transactions on Signal Processing*, 2008.

[87] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Q. Appl. Math*, 2(2):164–168, 1944.

[88] A.V. LEVY and A. MONTALVO. The tunneling algorithm for the global minimization of functions. *SIAM journal on scientific and statistical computing*, 6(1):15–29, 1985.

[89] J. Li. General explicit difference formulas for numerical differentiation. *Journal of Computational and Applied Mathematics*, 183(1):29–52, 2005.

[90] J.J. Liang and P.N. Suganthan. Dynamic multi-swarm particle swarm optimizer. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 124–129, 2005.

[91] K.H. Liang, X. Yao, and C. Newton. Combining landscape approximation and local search in global optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 2, pages 1514–1520. New York: IEEE Press, 1999.

[92] M. Locatelli. On the multilevel structure of global optimization problems. *Computational Optimization and Applications*, 30(1):5–22, 2005.

[93] M. Locatelli and F. Schoen. Random Linkage: a family of acceptance/rejection algorithms for global optimisation. *Mathematical Programming*, 85(2):379–396, 1999.

[94] S. Lucidl and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *Journal of Optimization Theory and Applications*, 62(2):255–277, 1989.

[95] JN Lyness and CB Moler. Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis*, pages 202–210, 1967.

[96] J.N. Lyness and G. Sande. Algorithm 413: ENTCAF and ENTCRE: evaluation of normalized Taylor coefficients of an analytic function. 1971.

[97] K. Madsen, H.B. Nielsen, and M.Ç. Pınar. Bound constrained quadratic programming via piecewise quadratic functions. *Mathematical Programming*, 85(1):135–156, 1999.

[98] JS Maltz, E. Polak, and TF Budinger. Multistart optimization algorithm for joint spatial and kinetic parameter estimation from dynamic ECT projection data. In *Proc. IEEE Nuc. Sci. Symp. Med. Im. Conf*, volume 3, pages 1567–73, 1998.

[99] C.D. Maranas and C.A. Floudas. Global minimum potential energy conformations of small molecules. *Journal of Global Optimization*, 4(2):135–170, 1994.

[100] D.W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, pages 431–441, 1963.

[101] J.H. Mathews and J.H. Mathews. *Numerical methods for mathematics, science, and engineering*. Prentice Hall Englewood Cliffs, NJ, 1992.

[102] I. MathWorks and I. MathWorks. *MATLAB: The Language of Technical Computing*. MathWorks, 2005.

[103] Z. Michalewicz. *Genetic algorithms+ data structures= evolution programs*. Springer, 1996.

[104] SK Mishra. Some New Test Functions for Global Optimization and Performance of Repulsive Particle Swarm Method.

[105] J.J. Mor. The Levenberg-Marquardt algorithm: implementation and theory. *Lecture notes in mathematics*, 630:105–116, 1977.

[106] J.J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optim.*, 1(1):93–113, 1991.

[107] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases, 1995.

[108] S. G. Nash and A. Sofer. *Practical optimization*. McGraw–Hill, 1996.

[109] JA Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308, 1964.

[110] JA Nelder and R. Mead. A simplex method for function minimization. *The computer journal (Errata)*, 8(4):27, 1965.

[111] J. Nocedal and S.J. Wright. *Numerical optimization*. Springer, 1999.

[112] J. Oliver. Numerical differentiation of analytic functions. *Journal of Computational and Applied Mathematics*, 6(2):145–160, 1980.

[113] E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. 1997.

[114] DG Papageorgiou, IN Demetropoulos, and IE Lagaris. Merlin-3.0 A multidimensional optimization environment. *Computer Physics Communications*, 109(2-3):227–249, 1998.

[115] DG Papageorgiou, IE Lagaris, NI Papanicolaou, G. Petsos, and HM Polatoglou. Merlin a versatile optimization environment applied to the design of metallic alloys and intermetallic compounds. *Computational Materials Science*, 28(2):125–133, 2003.

[116] P.M. Pardalos, H.E. Romeijn, and H. Tuy. Recent developments and trends in global optimization. *Journal of Computational and Applied Mathematics*, 124(1-2):209–228, 2000.

[117] KE Parsopoulos and MN Vrahatis. Modification of the particle swarm optimizer for locating all the global minima. In *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Prague, Czech Republic, 2001*, page 324. Springer Verlag Wien, 2001.

[118] KE Parsopoulos and MN Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2):235–306, 2002.

[119] KE Parsopoulos and MN Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, 2004.

[120] A.F. Perold. Large-scale portfolio optimization. *Management Science*, pages 1143–1160, 1984.

[121] J. Phillips. *The NAG Library: a beginners guide*. Oxford University Press, Inc. New York, NY, USA, 1987.

[122] H. Pohlheim. GEATbx: Genetic and evolutionary algorithm toolbox for use with matlab-documentation. *Technical University Ilmenau, Germany*, 1996.

[123] E. Polak and G. Ribiere. Note sur la convergence de methodes de directions conjuguees. *Revue Francaise dŏInformatique et de Recherche Operationnelle*, 16:35–43, 1969.

[124] M.J.D. Powell. Rank One Methods For Unconstrained Optimization. Technical report, TP–372, Atomic Energy Research Establishment, Harwell (England), 1969.

[125] MJD Powell. A new algorithm for unconstrained optimization. *Nonlinear Programming*, pages 31–65, 1970.

[126] M.J.D. Powell. A view of unconstrained optimization. *Optimization in Action, London*, 1976.

[127] M.J.D. Powell. *TOLMIN: A Fortran Package for Linearly Constrained Optimization Calculations.* University of Cambridge, Department of Applied Mathematics and Theoretical Physics, 1989.

[128] WL Price. A controlled random search procedure for global optimisation. *The Computer Journal*, 20(4):367–370, 1977.

[129] Deepa-G. Namboori K. Ramachandran, K. I. *Computational Chemistry and Molecular Modeling.* Springer, 2008.

[130] LA Rastrigin. Statistical search methods, 1968.

[131] G.J.E. Rawlins. *Foundations of genetic algorithms.* Morgan Kaufmann, 1991.

[132] T.A. Rijken, VGJ Stoks, RAM Klomp, J.L. de Kok, and JJ De Swart. The Nijmegen NN phase shift analyses. *Nuclear Physics, Section A*, 508:173–183, 1990.

[133] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods part I: Clustering methods. *Mathematical Programming*, 39(1):27–56, 1987.

[134] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods part II: Multi level methods. *Mathematical Programming*, 39(1):57–78, 1987.

[135] R.T. Rockafellar and S. Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2:21–42, 2000.

[136] K. Schittkowski. QLD: A Fortran Code for Quadratic Programming, User's Guide. *Mathematisches Institut, Universitat Bayreuth, Germany*, 1986.

[137] F. Schoen. Two-phase methods for global optimization. *Handbook of global optimization*, 2:151–178, 2002.

[138] JF Schutte, JA Reinbolt, BJ Fregly, RT Haftka, and AD George. Parallel global optimization with the particle swarm algorithm. *International journal for numerical methods in engineering*, 61(13), 2004.

[139] H.P. Schwefel. *Evolution and optimum seeking*. Wiley New York, 1995.

[140] DF Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, pages 647–656, 1970.

[141] Y. Shi and R.C. Eberhart. Parameter selection in particle swarm optimization. *Lecture notes in computer science*, pages 591–600, 1998.

[142] B.O. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, pages 379–388, 1972.

[143] SIAM. `http://www.siam.org/siamnews/01-02/challenge.pdf`.

[144] W. Spendley, GR Hext, and FR Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, pages 441–461, 1962.

[145] R. Srinivasan. RPC: Remote procedure call protocol specification version 2, 1995.

[146] WH Swann. Direct search methods. *Numerical Methods for Unconstrained Optimization, W. Murray, ed., Academic Press, London and New York*, pages 13–28, 1972.

[147] X. Tian, J.P. Hoeflinger, G. Haab, Y.K. Chen, M. Girkar, and S. Shah. A compiler for exploiting nested parallelism in OpenMP programs. *Parallel Computing*, 31(10-12):960–983, 2005.

[148] A. Törn and S. Viitanen. Topographical global optimization using pre-sampled points. *Journal of Global Optimization*, 5(3):267–276, 1994.

[149] A. Torn and A. Zilinskas. Global optimization. *Springer Lecture Notes In Computer Science; Vol. 350*, page 255, 1989.

[150] A.A. Törn. *A search-clustering approach to global optimization*. Åbo Swedish University School of Economics, 1977.

[151] I.G. Tsoulos and I.E. Lagaris. MinFinder: Locating all the local minima of a function. *Computer Physics Communications*, 174(2):166–179, 2006.

[152] W. Tu and RW Mayne. An approach to multi-start clustering for global optimization with non-linear constraints. *Int. J. Numer. Meth. Engng*, 53:2253–2269, 2002.

[153] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Marti. Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *INFORMS Journal on Computing*, 19(3):328, 2007.

[154] H. van de Waterbeemd, D.A. Smith, K. Beaumont, and D.K. Walker. Property-based design: optimization of drug absorption and pharmacokinetics. *J. Med. Chem*, 44(9):1313–1333, 2001.

[155] C. Voglis and I. E. Lagaris. A rectangular trust-region approach for unconstrained and box-constrained optimization problems. In *International Conference of Computational Methods in Sciences and Engineering*, 2004.

[156] Lagaris I.E.-Lekala M.L. Rampho G.J. Voglis, C. and S.A. Sofianos. Global minimization in few-body systems. *Nuclear Physics*, 2007.

[157] J. Waite. Routines for numerical interpolation, with first and second order differentiation, having non-uniformly spaced points, out to three dimensions. *Comput. Phys. Commun.*, 46:323, 1987.

[158] D.J. Wales and H.A. Scheraga. Global optimization of clusters, crystals, and biomolecules. *Science*, 285(5432):1368, 1999.

[159] L.D. Whitley. *Foundations of genetic algorithms 2*. Morgan Kaufmann, 1993.

[160] L.D. Whitley and Vose M. *Foundations of genetic algorithms 3*. Morgan Kaufmann, 1995.

[161] P. Wolfe. Convergence conditions for ascent methods. *SIAM review*, pages 226–235, 1969.

[162] S. Wolfram. *The mathematica book*. Cambridge university press, 1999.

[163] S.N. Wood. Minimizing model fitting objectives that contain spurious local minima by bootstrap restarting. *Biometrics*, 2004.

[164] S.J. Wright. *Primal-dual interior-point methods*. Society for Industrial Mathematics, 1997.

[165] S. (ed) Yip. *Handbook of Materials modelling*. Springer, 2005.

[166] R. Zieliński. A statistical estimate of the structure of multi-extremal problems. *Mathematical Programming*, 21(1):348–356, 1981.

[167] W.T. Ziemba and R.G. Vickson. *Stochastic optimization models in finance*. World Scientific, 2006.

Author's Publications

Journal Papers

1. **C. Voglis** and I.E. Lagaris, A Global Optimization Approach to Neural Network Training, *Neural, Parallel & Scientific Computations* **14**:231–240, 2006.
(*Global Optimization Application*)

2. M.G. Tsipouras, **C. Voglis** and D.I. Fotiadis, A Framework for Fuzzy Expert System CreationůApplication to Cardiovascular Diseases, *IEEE Transactions on Biomedical Engineering*, (**54**):2089–2105, 2007.
(*Global Optimization Application*)

3. C.Papadopoulos and **C. Voglis**, Drawing Graphs using Modular Decomposition, *Journal of Graph Algorithms and Applications* **11**(2): 481–511, 2007.
(*Global Optimization Application*)

4. **C. Voglis**, I.E. Lagaris, M.L. Lekala, G.J. Rampho and S.A. Sofianos, Global minimization in few-body systems, *Nuclear Physics* **790**:655–658, 2007.
(*Global Optimization Application*)

5. **C. Voglis** and I. E. Lagaris, Towards "Ideal Multistart". A stochastic approach for locating the minima of a continuous function inside a bounded domain, Applied Mathematics and Computation, 2009
(*Presented in Chapter 8*)

6. **C. Voglis** P. Hadjidoukas, D. Papageorgiou and I. E. Lagaris, A Numerical Differentiation Library Exploiting Parallel Architectures, Computer Physics Communications, 2009.
(*Presented in Chapter 6*)

Conference Papers

7. **C. Voglis** and S. A. Paschos, A Study on Intrusion Detection Techniques in a TCP/IP Environment, 3th WSEAS International Multiconference on Circuits, Systems, Communications and Computers, Athens, Greece, 1999.

8. **C. Voglis** and I.E. Lagaris, A Hybrid method for neural network training, 6th International Workshop on Mathematical methods in Scattering Theory and Biomedical Engineering, Tsepelovo, Greece, 2004.
(*Presented in Chapter 5*)

9. **C. Voglis** and I. E. Lagaris, A Rectangular Trust Region Dogleg Approach for Unconstrained and Bound Constrained Nonlinear Optimization, WSEAS International Conference on Applied Mathematics, Corfu, Greece, 2004.
(*Presented in Chapter 4*)

10. **C. Voglis** and I. E. Lagaris, A Rectangular Trust-Region Approach for Unconstrained and Box-Constrained Optimization Problems, International Conference of Computational Methods in Sciences and Engineering, Athens, Greece, 2004.
(*Presented in Chapter 4*)

11. **C. Voglis** and I. E. Lagaris, BOXCQP: An Algorithm for Bound Constrained Convex Quadratic Problems , 1st International Conference "From Scientific Computing to Computational Engineering", Athens, Greece, 2004.
(*Presented in Chapter 3*)

12. M.G. Tsipouras, **C. Voglis**, I.E. Lagaris and D.I. Fotiadis, A Framework for Fuzzy Expert System Creation, 7th International Workshop on Mathematical Methods in Scattering Theory and Biomedical Technology, Nymfaio, Greece, 2005.
(*Global Optimization Application*)

13. C. Papadopoulos and **C. Voglis** Drawing Graphs using Modular Decomposition, 13th Symposium on Graph Drawing GD2005, Springer LNCS 3843:343–354, 2005.
(*Global Optimization Application*)

14. M.G. Tsipouras, **C. Voglis**, I.E. Lagaris, D.I. Fotiadis, Cardiac arrythmia classification using support vector machines, The 3rd European Medical and Biological Engineering Conference, Prague, 2005.
(*Application of method described in Chapter 3*)

15. **C. Voglis** and I. E. Lagaris, Smeenos: A Clustered Particle Swarm Algorithm for Recovering the Local Minima of a Function, Optimization 2007, Porto, Portugal, 2007.
(*Presented in Chapter 10*)

16. **C. Voglis** and I. E. Lagaris, Global Optimization by Adaptively Estimating the Probability for Local Search, Optimization 2007, Porto, Portugal, 2007.
(*Presented in Chapter 8*)

17. N. Kyrgios, **C. Voglis** and I. E. Lagaris, Multistart Optimization with a Trainable Decision Maker for Avoiding High Valued Local Minima, 4th IC-EpsMsO, Athens, 2009.

Short Vita

Costas Voglis

Personal information

Date of birth:	22 March, 1978	**Telephone-2**:	+30 26510 63301
Place of birth:	Ioannina	**Email**:	voglis@cs.uoi.gr
Telephone-1:	+30 6977053095	**Homepage**:	www.cs.uoi.gr/∼voglis
Marital status:	Single	**Nationality**:	Greek

Current Status: PhD Candidate in Computer Science, University of Ioannina

Education

3/2003–6/2010 Ph.D in Computer Science, University of Ioannina
Ph.D Dissertation: *Methods for Local and Global Optimization*
Thesis Advisor: Isaac E. Lagaris

09/1999–09/2001 M.Sc in Computer Science, University of Ioannina
M.Sc Dissertation: *Model Based Intrusion detection*

08/1995–08/1999 B.Sc in Computer Science,University of Ioannina
B.Sc Dissertation: *Intrusion Detection in TCP/IP Networks*

Grants–Funts

1995 Scholarship from the Department of Computer Science , University of Ioannina for the undergraduate programme.

1999–2001 Scholarship from the Department of Computer Science , University of Ioannina for the graduate programme.

9/2004–8/2005 Scholarship supported by the European Union in the framework of the project "Support of Computer Science Studies in the University of Ioannina" of the 3rd Community Support Framework of the Hellenic Ministry of Education

---

Research Interests

- Development and implementation of Global and Local Optimization methods.
- Implementation of parallel algorithms (MPI, OpenMP)
- Simulation of classical systems with the Molecular Dynamics techniques
- Neural Networks for pattern recognition and function approximation.
- Optimization in Inverse Scattering problems.
- Graph Drawing.

Research Activities

- Publications in International Scientific Journals and Conferences.
- Member of IPAN (*Information Processing and Analysis Research Group*) in the Dept. of Computer Science, University of Ioannina.

Working Experience

2006–2007 Collaborator in the EU Program entitled *Open Source*.

5/2008–7/2008 External Collaborator of Unisystems S.A. on the project *Service provider for the production of cards for Digital Tachographs*, Hellenic Ministry of Transport and Communications.

Teaching Experience

10/1999–6/2007 Teaching assistant in undergraduate courses in Dept. of Computer Science, University of Ioannina.

10/2006–6/2007 Laboratory Assistant, Department of TeleInformatics & Management, Technological Educational Institute of Epirus.

2006 Teaching assistant in the graduate course of Nonlinear Optimization

2007	Teaching assistant in the graduate course of Continuous Global Optimization
9/2007	Assistant supervisor for B.Sc Dissertation entitled *Global Optimization Using Bee Colonies*
11/2007	Assistant supervisor for B.Sc Dissertation entitled *Particle Swarm Global Optimization*
3/2008	Assistant supervisor for B.Sc Dissertation entitled *Global Optimization Using Interval Analysis*
5/2008	Assistant supervisor for M.Sc Dissertation entitled *An Application of Normal Distribution Sampling in Global Optimization*

Graduate Courses

- ◇ Topics on Database Systems: Models, Languages and Architecture.
- ◇ Topics on Neural Networks and Fuzzy Logic.
- ◇ Optimization.
- ◇ Computer Aided Design: Algorithms and Systems.
- ◇ Topics on Computers Network and Network Programming.
- ◇ Topics on Biomedical Informatics
- ◇ Semantics of Programming Languages.
- ◇ Machine Learning.

Programming Skills

- Programming in C, Fortran, C++, Java
- Parallel implementation libraries MPI, OpenMP
- Experienced Matlab Development/Modelling
- ASP, PHP

Languages

Greek	Native language
English	Very good

Hobbies

- Music, Travelling, Soccer.