ADDIS ABABA UNIVERSITY

Departement of Mathematics

Final project

ON

Derivative free optimization methods

(Submitted in partial fulfillment of M.Sc. Degree in Mathematics)

Prepared by: FLAGOT YOHANNES

Advisor: Dr. SEMU MITIKU

January, 2012

Addis Ababa

Acknowledgement

I would like to express my gratitude to all those who supported me in any means to complete this project**.** I am grateful to my advisor and instructor Dr. Semu Mitku for his genuine advice, constructive comment, invaluable suggestions and provisions of materials in preparing this project.

Abstract.

Let f be a continuous function on R^n, and suppose f is a smooth nonlinear function. Such functions arise in many applications, and very often minimizers are points at which f is not differentiable. Of particular interest is the case where the gradient and the Hessian cannot be computed for any x. I present a practical, robust algorithm to locally minimize such functions, based on *model sampling or search*. No derivatives information is required by the algorithm.

The algorithms generate a sequence with an initial point. Furthermore, I show that if f has a unique solution, then the set of all cluster points generated by the algorithm converges globally. Numerical results are presented demonstrating the robustness of the algorithm.

Contents

1. Introduction 5
   1.1. Background 7
   1.2. History of Derivative Free Optimization Methods 8
   1.3. Features of DFO 9
2. Methods for Derivative free optimization 10
   2.1. The Nelder–Mead method 12
      2.1.1. Algorithm for Nelder–Mead method 14
   2.2. The Golden Section method 16
      2.2.1. Algorithm for golden section method 19
   2.3. The Trust Region method 20
      2.3.1. Algorithm for trust-region method 27
3. Test functions and experimental setup 28
   3.1. Example1 29
      3.1.1. Solve Example1 by using Nelder-Mead method 29
      3.1.2. Solve Example1 by using golden section method 30
      3.1.3. Solve Example1 by using Trust Region method 31
   3.2. Example2 32
      3.2.1. Solve Example2 by using Nelder-Mead method 32
      3.2.2. Solve Example2 by using golden section method 33
      3.2.3. Solve Example2 by using Trust Region method 35
   3.3. Example3 36
      3.3.1. Solve Example3 by using Nelder-Mead method 36
      3.3.2. Solve Example3 by using golden section method 37
      3.3.3. Solve Example3 by using Trust Region method 38
4. Conclusion 40
5. Appendices 41
   5.1. Matlab code for Nelder–Mead method 41
   5.2. Matlab code for golden section method 45
   5.3. Matlab code for thrust region method by using powell method 49
6. References 52

1. Introduction

It is well known that extensive useful information is contained in the derivatives of any function one wishes to optimize. However, for a variety of reasons there have always been many instances where (at least some) derivatives are unavailable or numerically unreliable.

Derivative free optimization (DFO) methods are designed for solving nonlinear optimization problems where the derivatives of the involved function are not available.

Increasing complexity in mathematical modeling, higher sophistication of scientific computing, and an abundance of legacy codes are some of the reasons why derivative-free optimization is currently an area of great demand. There is a high demand from practitioners for such algorithms because this kind of problems occur relatively frequently in the industry.

Optimization problems, in which the derivatives cannot be computed, arise in modern physical, chemical and econometric measurements and in engineering applications, where computer simulation is employed for the evaluation of the objective functions.

In applications either the evaluation of the objective function is very difficult or expensive, or the derivatives of such functions are not available.

Currently DFO work when the dimensions are relatively at small scale (less than 100 variables), their objective function is relatively expensive to compute and derivatives of such functions are not available and cannot be estimated efficiently.

There may also be some noise in the function evaluation procedures. Such optimization problems arise, for example, in engineering design, where the objective function evaluation is a simulation package treated as a black box.

This project is mostly devoted to the study of derivative-free algorithms for unconstrained optimization problems, which we will write in the form

$$\min_{x \in R^n} f(x)$$

Where f is a smooth nonlinear objective function from R^n into R and is bounded below. We assume that the gradient and the Hessian cannot be computed for any x.

The methods we will consider do not rely on derivative information of the objective function or constraints, nor are the methods designed explicitly to approximate these derivatives. Rather, they build models of the functions based on sample function values or they directly exploit a sample set of function values without building an explicit model.

We are interested in algorithms that are globally convergent to stationary points (of first- or second-order type), in other words, algorithms that regardless of the starting point are able to generate sequences of iterates asymptotically approaching stationary points.

There are mainly four classes of derivative-free optimization methods. The first class of DFO algorithms are the direct search or pattern search methods which are based on the exploration of the variable space by using sample points from a predefined class geometric pattern and use either the Nelder–Mead simplex algorithm or parallel direct search algorithm. They do not exploit the inherent smoothness of the objective function and require, therefore, a very large number of function evaluations. They can be useful for non-smooth problems. In this project we use Nelder-mead simplex algorithm rather than pattern search method for investigation. The second class of DFO's is a line search method which consists of a sequence of $n + 1$ dimensional searches. These methods are the oldest and most widely used methods in unconstrained optimization and use golden section search method or other methods. The combination of finite difference techniques coupled with quasi-Newton method constitutes the third class of the algorithms. The last class of the methods is based on modeling the objective function by multivariate interpolation in combination with the trust-region techniques. The main idea of these methods is building a polynomial model, interpolating the objective function at all points at which its value is known. The model is then minimized over the trust region and a new point, is computed. The objective function evaluated at this new point thus possibly enlarges the interpolation set. This newly computed point is checked as to whether the objective function is improved and the whole process repeated until convergence is achieved.

The main criterion for assessment and comparison of DFO algorithms is the number of function evaluations they require for minimization.

After the introduction, we begin the first part of the project Nelder-mead simplex method and its algorithm. The next part is golden section search method and its algorithm. The last one is trust-region method and its algorithm. After we discus all these three methods, then we compare the performance of each of them on some selected test problems.

1.1. Background

In derivative-free optimization, the goal is to optimize a function defined on a subset of R^n for which derivative information is neither symbolically available nor numerically computable. Since the beginning of the field in 1961 with the algorithm by Hooke & Jeeves [9], numerous algorithms and software implementations have been proposed. Recently the interest in derivative-free optimization grew rapidly, motivated by practical applications in physics, chemistry, biology, engineering design, medical science, finance, and operations research.

Many approaches and algorithms exist already, and the field undergoes rapid growth. Currently, there are various approaches to derivative-free optimization coming from three different research communities that in the past interacted only loosely. These approaches have led to three main classes of algorithms:

Direct methods are deterministic but make no model assumptions. They include the simplex method by Nelder & Mead [3], pattern search (e.g., Torczon [11]), and generating set search (e.g., Kolda et al. [13]).

Indirect or model-based methods are also deterministic, and include methods using quadratic model fits (e.g., SNOBFIT by Huyer & Neumaier [14]), trust-region methods based on model updates (e.g., NEWUOA by Powell [15]), other response surface methods (e.g. DACE by Sacks et al. [16]) and surrogate function methods (e.g., EGO by Jones et al. [18]), implicit filtering (e.g., Gilmore & Kelley [10]) and multilevel coordinate search (MCS) by Huyer & Neumaier [17].

Stochastic methods use random choices in their strategy and include, e.g., simulated annealing (Van Laarhoven & Aarts et al. [21]), genetic algorithms (e.g., real-coded CHC by Eshelman & Schaffer [24]) and their modern evolutionary improvements (e.g., DE by Storn & Price [22] and G CMA-ES by Auger & Hansen [23, 25]), stochastic clustering algorithms (e.g., GLOBAL by Csendes et al. [27]), particle swarm methods (e.g., Eberhart & Kennedy [29]), ant colony optimization (e.g., Dorigo & St¨utzle [28]) and tabu search (e.g., Glover [31]). There are also various hybrid methods, e.g. between direct search and particle swarms (Fan & Zahara[30]). Stochastic approaches can be useful for dealing with rough functions or functions that contain multiple local optima.

Although gradient-free methodologies have been in existence for many years, they have become widely used in only the last 20 years or so [30]. This relatively recent uptake can be attributed to several factors, including the wide availability of large numbers of cores (combined with algorithms that parallelize easily), the significant theoretical results achieved in

this period, and the successful application of derivative-free techniques in a number of areas. Examples can be found in molecular geometry [10], aircraft design [31], hydrodynamics [32, 33] and medicine [34, 35].

1.2. History of Derivative Free Optimization Methods

Although it is not exactly known when the idea of derivative free methods for minimization was first introduced, we see that the approach of using the direct search methods arose in 1950's [16]. A detailed review on the historical developments of derivative free optimization methods can be found in [16].

The idea of employing available objective function values $f(x)$ for building the quadratic model by interpolation was firstly introduced by Winfield in 1960's [12]. This model is assumed to be valid in a neighborhood of the current iterate, which is described as a trust-region, whose radius is iteratively adjusted. The model is then minimized within the trust-region, hopefully yielding a point with a lower function value. As the algorithm proceeds and more objective functional values become available, the set of points defining the interpolation model is updated in such a way that it always contains the points closest to the current iterate.

Winfield recognizes the difficulty that interpolation points must have certain geometric properties, although he does not seem to consider what happens if these properties are not obtained.

Twenty five years later, Powell proposed a method for constrained optimization, whose idea is close to that of Winfield. In his proposal, the objective function and constraints are approximated by linear multivariate interpolation. He explored Winfield's idea further by describing an algorithm for unconstrained optimization using a quadratic multivariate interpolation model of the objective function in a trust-region framework.

The crucial difference between Powell's and Winfield's proposals is that in the first one, the set of interpolation points is updated in a way that geometric properties of these points are preserved [16]. This means that the differences between points of this set are guaranteed to remain "sufficiently" linearly independent. Adding this condition to the interpolation set was good for avoiding the difficulties associated with earlier proposals.

The first convergence theorems for methods of this type were presented by Conn, Scheinberg and Toint in [14]. They also described some alternative techniques to strengthen the geometric properties of the set of the interpolation points. In [19], we find the discussion

on the studies of a derivative free optimization method via support vector machines (see [36] for details) to improve the convergence properties of the method with the approach of low tolerance to noise, good initial reduction in the objective function exploiting the geometry of the interpolation points.

The appearance of min-type, max-type or other non differentiable functions has given rise to non-smooth analysis and optimization [13].

1.3. Features of DFO

There are three features present in all globally convergent derivative-free algorithms:

1. They incorporate some mechanism to impose descent away from stationary. The same is done by derivative-based algorithms to enforce global convergence, so this imposition is not really new. It is the way in which this is imposed that makes the difference. Direct-search methods of directional type, for instance, achieve this goal by using positive bases or spanning sets and moving in the direction of the points of the pattern with the best function value. Simplex-based methods ensure descent from simplex operations like reflections, by moving in the direction away from the point with the worst function value. Methods like the implicit-filtering method aim to get descent along negative simplex gradients, which are intimately related to polynomial models. Trust-region methods, in turn, minimize trust-region sub problems defined by fully linear or fully quadratic models, typically built from polynomial interpolation or regression. In every case, descent is guaranteed away from stationary by combining such mechanisms with a possible reduction of the corresponding step size parameter. Such a parameter could be a mesh size parameter (directional direct search), a simplex diameter (simplicial direct search), a line-search parameter, or a trust-region radius.
2. They must guarantee some form of control of the geometry of the sample sets where the function is evaluated. Essentially, such operations ensure that any indication of stationarity (like model stationarity) is indeed a true one. Not enforcing good geometry explains the lack of convergence of the original Nelder–Mead method. Examples of measures of geometry are (i) the cosine measure for positive spanning sets; (ii) the normalized volume of simplices (both to be kept away from zero); (iii) the poisedness constant, to be maintained moderately small and bounded from above when building interpolation models and simplex derivatives.

3. They must drive the step size parameter to zero. We know that most optimization codes stop execution when the step size parameter passes below a given small thresh-old. In derivative-based optimization such terminations may be premature and an indication of failure, perhaps because the derivatives are either not accurate enough or wrongly coded. The best stopping criteria when derivatives are available are based on some form of stationarity indicated by the first-order necessary conditions. In derivative-free optimization the step size serves a double purpose: besides bounding the size of the minimization step it also controls the size of the local area where the function is sampled around the current iterate. For example, in direct-search methods the step size and the mesh size (defining the pattern) are the same or constant multiples of each other. In a model-based derivative-free method, the size of the trust region or line-search step is typically intimately connected with the radius of the sample set. Clearly, the radius of the sample set or mesh size has to converge to zero in order to ensure the accuracy of the objective function representation. It is possible to decouple the step size from the size of the sample set; however, so far most derivative-free methods (with the exception of the original Nelder–Mead method) connect the two quantities. In fact, the convergence theory of derivative-free methods that we will see that the sequence (or a subsequence) of the step size parameters do converge to zero. It is an implicit consequence of the mechanisms of effective algorithms and should not (or does not have to) be enforced explicitly. Thus, a stopping criterion based on the size of the step is a natural one.

There are three important characterizing problems of derivative free optimization. Firstly, evaluating the objective function at a given vector x is computationally very expensive. This kind of expensive evaluations causes extensive linear algebra calculations in an algorithm when the optimal value of these evaluations is being looked for. The second feature is about the computation of derivatives. In this type of nonlinear optimization problems, the natures of the objective function prevent us from the computation of any associated derivatives (gradient or Hessian). Finally, the objective function value is usually computed with some noise which puts additional requirements on the minimization's robustness.

2. Methods for Derivative free optimization

This section introduces the different algorithms that will be compared in this project. Consider an optimization problem

$$\min_{x \in R^n} f(x)$$

Several strategies can be considered when we are faced with the problems which do not allow utilization of direct derivatives of gradients [2]. The first is to apply existing direct search optimization methods, like the well-known and widely used simplex reflection algorithm of [3] or its modern variants, or the parallel direct search algorithm of [11] and [19]. This first approach has the advantage of requiring little effort from the user; however, it may require substantial computing resources like a great number of function evaluations. This is because it does not take into account the advantage of the objective function's smoothness well enough.

The second approach is using automatic differentiation tools. Automatic differentiation is utilized to define computer programs which calculate the derivatives of a function by some procedures. These computer programs calculate the Jacobean of vector-valued functions which are from n-dimensional to m-dimensional Euclidean space, i.e., from R^n to R^m. On the other hand, if the function is scalar-valued, i.e., from R^n to R , then the computer program should calculate the gradient (and Hessian) of the function [26]. However, such tools are not preferred in solving problems which we consider. This is mainly because in the automatic differentiation tools approach, the function to be differentiated is required to be the result of a callable program which cannot be treated as a black box.

A third possibility is to make use of finite difference approximation of the derivatives (gradients and possibly Hessian matrices). In general, given the cost of evaluating the objective function, evaluating its Hessian by finite differences is much too expensive. One can utilize quasi-Newton Hessian approximation techniques instead. Incorporating finite differences for computing gradients in conjunction with the quasi-Newton Hessian approximation techniques has proved to be helpful and sometimes surprisingly efficient.

Indeed, the additional function evaluations required in the calculation of the derivatives may be very costly and, most importantly, finite differencing can be unreliable in the presence of noise if the differentiation step is not adapted according to the noise level.

The objective functions in the problems we consider are obtained from some simulation procedures; therefore, automatic differentiation tools are not applicable as mentioned above. This forces one to consider algorithms without preceding the approximation of the derivatives of the objective function at a given value.

We will look at discrete gradients from non-smooth optimization, where the approach can be interpreted as an approximation or mimicking of derivatives.

There are two important components of derivative free methods. Sampling better points in the iteration procedure is the first one of these components. The other one is searching appropriate subspaces where the chance of finding a minimum is relatively high.

In order to be able to use the extensive convergence theory for derivative based methods, these derivative free methods need to satisfy some properties. For instance, to guarantee the convergence of a derivative free method, we need to ensure that the error in the gradient converges to zero when the trust-region or line-search steps are reduced. Hence, a descent step will be found if the gradient of the true function is not zero at the current iterate (for details see [18]). To show this, one needs to prove that the linear or quadratic approximation models satisfy Taylor-like error bounds on the function value and the gradient.

Finally, for our approach to derivative free optimization given by non-smooth optimization, we shall use so-called discrete gradients [7, 8].

2.1. The Nelder–Mead method

The Nelder–Mead algorithm is a direct-search method in the sense that it evaluates the objective function at a finite number of points per iteration and decides which action to take next solely based on those function values and without any explicit or implicit derivative approximation or model building. Every iteration in R^n is based on a simplex in R^n or $n+1$ linearly independent vertices $y = \{y_0, y_1, \dots, y_n\}$ ordered by increasing values of f. The most common Nelder–Mead iterations perform a reflection, an expansion, or a contraction (the latter can be inside or outside the simplex). In such iterations the worst vertex y_n is replaced by a point in the line that connects y_n and y_c,

$$y = y_c + \delta(y_c - y_n)\,, \delta \in R$$

Where $y_c = \sum_{i=0}^{n-1} \frac{y_i}{n}$ is the center of the best n vertices. The value of δ indicates the type of iteration. For instance, when $\delta = 1$ we have a (genuine or isometric) reflection $\delta = \delta^r$, when $\delta = 2$ an expansion $\delta = \delta^e$, when $\delta = 1/2$ outside contraction $\delta = \delta^{oc}$, and when $\delta = -1/2$ an inside contraction $\delta = \delta^{ic}$. In Figure 2.1, we plot these four situations.

Nelder–Mead iteration can also perform a simplex shrink, which rarely occurs in practice. When a shrink is performed all the vertices in y are thrown away except the best one y_0. Then n new vertices are computed by shrinking the simplex at y_0, i.e., by computing, for instance,

$y_0 + \gamma^s(y_i - y_0),\ i = 1, \ldots, n\ \ \gamma^s \in (0,1)$ See Figure 2.2.

We note that the "shape" of the resulting simplexes can change by being stretched or contracted, unless a shrink occurs.

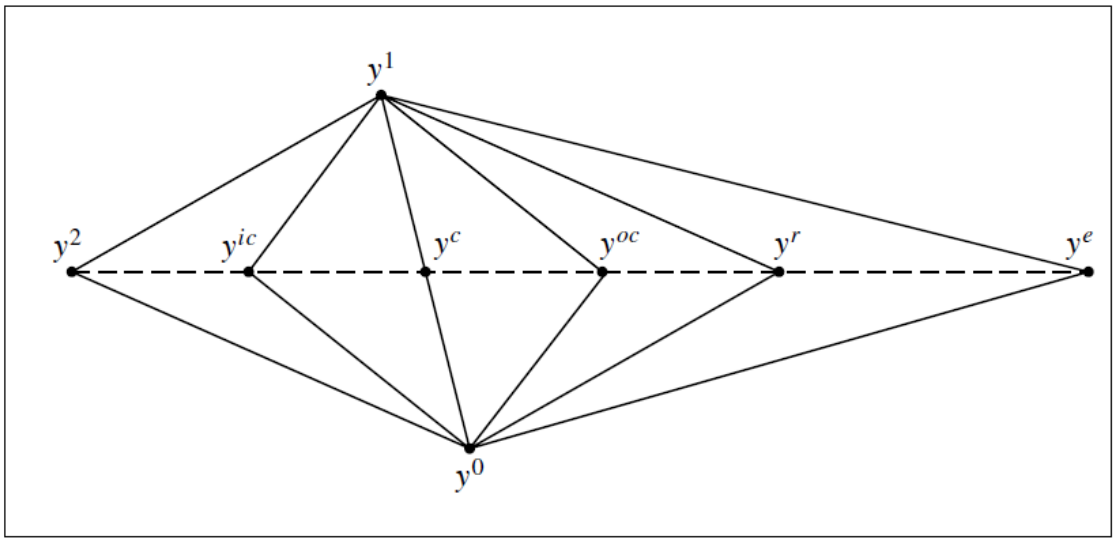


Figure 2.1. Reflection, expansion, outside contraction, and inside contraction of a simplex, used by the Nelder–Mead method.

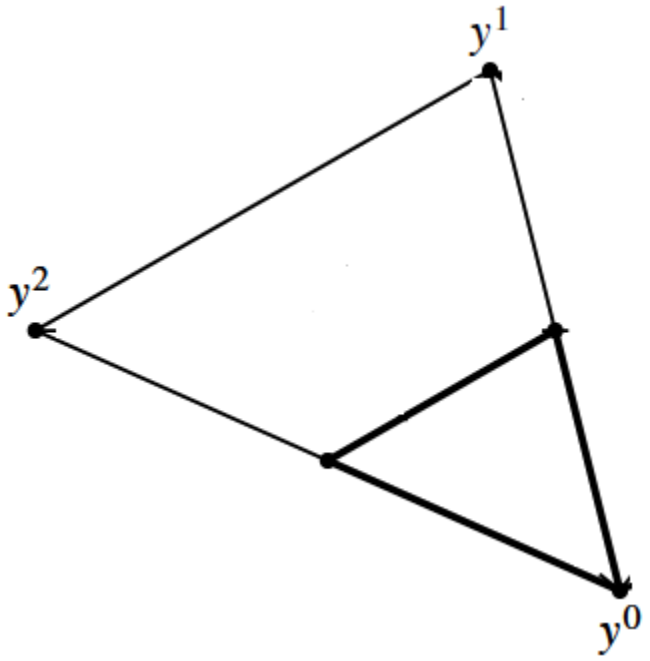


Figure 2.2. Shrink of a simplex, used by the Nelder–Mead method.

Note that, except for shrinks, the emphasis is on replacing the worst vertex rather than improving the best. It is also worth mentioning that the Nelder–Mead method does not parallelize well since the sampling procedure is necessarily sequential (except at a shrink).

2.1.1. Algorithm for Nelder–Mead method

Initialization: Choose an initial simplex of vertices $y_0 = \{y_0^0, y_0^1, y_0^2, \dots, y_0^n\}$.

Evaluate f at the points in y_0. Choose constants:

$$0 < \gamma^s < 1\,, \quad -1 < \delta^{ic} < 0 < \delta^{oc} < \delta^r < \delta^e$$

The Nelder–Mead method Algorithm standard choices for the coefficients used are

$$\gamma^s = \frac{1}{2}\,, \quad \delta^{ic} = -\frac{1}{2}\,, \quad \delta^{oc} = \frac{1}{2}\,, \quad \delta^r = 1 \ and \quad \delta^e = 2$$

For $k = 0,1,2, \dots$, perform the following steps

Set $y = y_k$.

1. **Order:** Order the $n+1$ vertices of $y = \{y^0, y^1, \dots, y^n\}$ so that

$$f^0 = f(y^0) \le f^1 = f(y^1) \le \cdots \le f^n = f(y^n)$$

2. **Reflect:** Reflect the worst vertex y^n over the centroid

   $y^c = \sum_{i=0}^{n-1} \frac{y^i}{n}$ of the remaining n vertices by evaluating

   $$y^r = y^c + \delta^r(y^c - y^n).$$

   Evaluate $f^r = f(y^r)$. If $f^0 \leq f^r < f^{n-1}$, then replace y^n by the reflected point y^r and update the vertices:

   $$y_{k+1} = \{ y^0, y^1, \dots, y^{n-1}, y^r \}$$

3. **Expand:** If $f^r < f^0$, then calculate the expansion point $y^e = y^c + \delta^e(y^c - y^n)$ and evaluate $f^r = f(y^r)$.

   If $f^e \leq f^r$, replace y^n by the expansion point y^e and update the vertices:

   $y_{k+1} = \{ y^0, y^1, \dots, y^{n-1}, y^e \}$ Otherwise, replace y^n by the reflected point y^r and update the vertices: $y_{k+1} = \{ y^0, y^1, \dots, y^{n-1}, y^r \}$

4. **Contract:** If $f^r \geq f^{n-1}$, then a contraction is performed between the best of y^r and y^n.

   a. **Outside contraction:** If $f^r < f^n$, perform an outside contraction

      $y^{oc} = y^c + \delta^{oc}(y^c - y^n)$ and evaluate $f^{oc} = f(y^{oc})$. If $f^{oc} \leq f^r$, then replace y^n by the outside contraction point y^{oc} and update the vertices:

      $y_{k+1} = \{ y^0, y^1, \dots, y^{n-1}, y^{oc} \}$ Otherwise, perform a shrink.

   b. **Inside contraction:** If $f^r \geq f^n$, perform an inside contraction

      $y^{ic} = y^c + \delta^{ic}(y^c - y^n)$ and evaluate $f^{ic} = f\left(y^{ic}\right)$. If $f^{ic} < f^n$, then replace y^n by the inside contraction point y^{ic} and update the vertices:

      $y_{k+1} = \{ y^0, y^1, \dots, y^{n-1}, y^{ic} \}$ Otherwise, perform a shrink.

5. **Shrink:** Evaluate f at the n point

   $y^0 + \gamma^s(y^i - y^0),\ i = 1, \dots, n,$ and replace $y^1, \dots, y^n$ by these points, update the vertices: $y_{k+1} = \{y^0 + \gamma^s(y^i - y^0),\ i = 1, \dots, n,\}$

A stopping criterion could consist of terminating the run when the diameter of the simplex becomes smaller than a chosen tolerance $\Delta tol > 0$ (for instance, $\Delta tol = 10^{-5}$).

The Nelder–Mead algorithm performs the following number of function evaluations per iteration:

1 If the iteration is a reflection,

2 If the iteration is an expansion or contraction,

$n + 2$ if the iteration is a shrink.

The most general property of the Nelder–Mead algorithm is stated in the next theorem.

Theorem Consider the application of the Nelder–Mead method to a function f which is bounded from below on R^n.

1. The sequence $\{f_k^0\}$ is convergent.
2. If only a finite number of shrinks occur, then all the $n + 1$ sequences $\{f_k^i\} i = 1,2, \dots, n$ converge and their limits satisfy $f_*^0 \leq f_*^1 \leq f_*^2 \leq \cdots \leq f_*^n$

   Moreover, if there is an integer $j \in \{0,2, \dots, n-1\}$ for which $f_*^j < f_*^{j+1}$(a property called broken convergence), then for sufficiently large k the change index is such that $k^* > j$
3. If only a finite number of non shrinks occur, then all the simplex vertices converge to a single point.

Proof The proof of the first and second assertions is essentially based on the fact that monotonically decreasing sequences bounded from below are convergent.

2.2. The Golden Section method

Definition: Interval of uncertainty

Consider $f(\lambda),\ a \leq \lambda \leq b$ since the exact location of $minf(\lambda)$, over $[a,b]$ is not known, this interval is called interval of uncertainty, i.e. $[a,b]$ is called interval of uncertainty of a minimum λ in $[a,b]$ though its exact value is not known

Theorem: let $f: R \to R$ be strictly quasi convex over $[a,b],\ \lambda,\mu \in [a,b]$ such that $\lambda < \mu$,

if $f(\lambda) > f(\mu)$ then $f(z) \geq f(\mu)\ \forall z \in [a,\lambda)$

(If $f(\lambda) \leq f(\mu)$ then $f(z) \geq f(\lambda) \quad \forall z \in (\mu,b]$)

Proof: Suppose $f(\lambda) > f(\mu)$ and let $z \in [a,\lambda)$ suppose $f(z) < f(\mu)$ since λ can be a convex combination of z and μ and by the strict quasi convexity of f we have

$f(\lambda) < max\{f(z), f(\mu)\} = f(\mu) \ \Rightarrow\ f(\lambda) < f(\mu)$ which is a contradiction to the assumption $f(\lambda) > f(\mu)$

Hence $f(z) \geq f(\mu)\ \forall z \in [a,\lambda)$ thus the interval of uncertainty will be reduced to $[\lambda,b]$ similarly for the next part □

For a strictly quasi convex function f let, at iteration point k, the interval of uncertainty is $[a_k,b_k]$. By the above theorem the new interval of uncertainty $[a_{k+1},b_{k+1}]$ is given by $[\lambda_k,b_k]$ if $f(\lambda_k) > f(\mu_k)$ and $[a_k,\mu_k]$ if $f(\lambda_k) \leq f(\mu_k)$

1. The length of the new interval of uncertainty $b_{k+1} - a_{k+1}$ doesn't depend on the outcome of k^{th} iteration i.e. whether $f(\lambda_k) > f(\mu_k)$ or $f(\lambda_k) \leq f(\mu_k)$. Therefore, we must have $b_k - \lambda_k = \mu_k - a_k$, hence if λ_k is of the form

   $\lambda_k = a_k + (1-\alpha)(b_k - a_k)$ where $\alpha \in (0,1)$ then μ_k must be of the form, $\mu_k = a_k + \alpha(b_k - a_k)$ so that $b_{k+1} - a_{k+1} = \alpha(b_k - a_k)$.

2. As λ_{k+1} and μ_{k+1} are selected for the purpose of a new iteration, either λ_{k+1} coincides with μ_k or μ_{k+1} coincides with λ_k. If this is so during iteration $k+1$ only one extra observation is required.

   Case-1: if $f(\lambda_k) > f(\mu_k)$, then $a_{k+1} = \lambda_k$ and $b_{k+1} = b_k$ to satisfy $\lambda_{k+1} = \mu_k$

   apply $\lambda_k = a_k + (1-\alpha)(b_k - a_k)$ with k replaced by $k+1$ we get

$$\mu_k = \lambda_{k+1} = a_{k+1} + (1-\alpha)(b_{k+1} - a_{k+1}) = \lambda_k + (1-\alpha)(b_k - \lambda_k).$$

Substituting $\lambda_k = a_k + (1-\alpha)(b_k - a_k)$, $\mu_k = a_k + \alpha(b_k - a_k)$ in this equation we get

$\alpha^2 + \alpha - 1 = 0$

Case-2: if $f(\lambda_k) \leq f(\mu_k)$ then $a_{k+1} = a_k$ and $b_{k+1} = \mu_k$ to satisfy $\mu_{k+1} = \lambda_k$

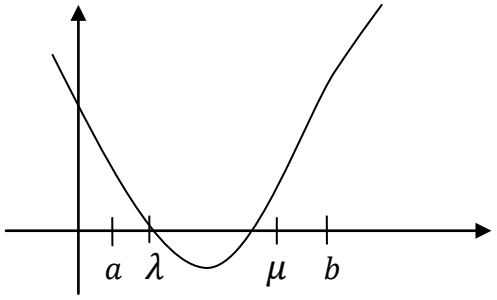
apply $\mu_k = a_k + \alpha(b_k - a_k)$ with k replaced by $k+1$ we get

$\lambda_k = \mu_{k+1} = a_{k+1} + \alpha(b_{k+1} - a_{k+1}) = a_k + \alpha(\mu_k - a_k)$. Substituting

$\lambda_k = a_k + (1-\alpha)(b_k - a_k)$, $\mu_k = a_k + \alpha(b_k - a_k)$ in this equation we get again $\alpha^2 + \alpha - 1 = 0$

Which has a positive root $\alpha = 0.618$; hence the interval of uncertainty will be reduced by a factor of $\alpha = 0.618$ (the golden number/ratio)

3. At the first iteration two observation are required but latter on only one is required i.e. either $\lambda_{k+1} = \mu_k$ or $\mu_{k+1} = \lambda_k$



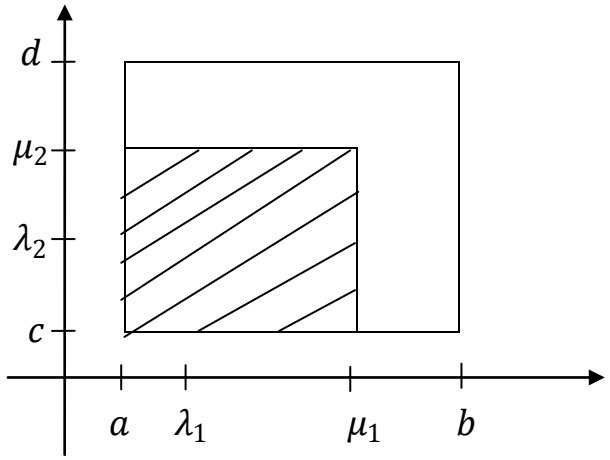
If initial interval of uncertainty is $[a, b]$ then the second interval of uncertainty become $[a, \mu]$ if $f(\lambda) < f(\mu)$ or $[\lambda, b]$ if $f(\lambda) > f(\mu)$.

The algorithm works for R^n that means if $x \in R^n$ i.e. $x = (x_1, \dots, x_n)$ we define interval of uncertainty for each x_i where $i = 1, \dots, n$

For example if $n = 2$ interval of uncertainty is rectangle. Initially for each x_i $i = 1,2$ interval of uncertainty become $x_1 = [a, b]$ $x_2 = [c, d]$ and calculate $\mu = (\mu_1, \mu_2)$ and $\lambda =$

(λ_1, λ_2) then, the second interval of uncertainty can be $x_1 = [a, \mu_1]\, x_2 = [c, \mu_2]$ if $f(\lambda) < f(\mu)$ or $x_1 = [\lambda_1, b]\, x_2 = [\lambda_2, d]$ if $f(\lambda) > f(\mu)$.

This shows by the next figure.



rectangle $abcd$ is initial interval of uncertainty and the shaded part is the second interval of uncertainty for $f(\lambda) < f(\mu)$.

2.2.1. Algorithm for golden section method

Step 0: initialization Choose allowable length $l > 0$ let $[a_1, b_1]$ be the initial interval of uncertainty, let $\lambda_1 = a_1 + (1 - \alpha)(b_1 - a_1)$ and

$\mu_1 = a_1 + \alpha(b_1 - a_1)$, with $\alpha = 0.618$.

Evaluate $f(\lambda_1)$ and $f(\mu_1)$

Let $k = 1$

Step 1: if $b_k - a_k < l$ stop; the optimal solution lies in $[a_k, b_k]$ otherwise, if

$f(\lambda_k) > f(\mu_k)$ go to step 2 and if $f(\lambda_k) \leq f(\mu_k)$ go to step 3

Step 2: let $a_{k+1} = \lambda_k$ and $b_{k+1} = b_k$ furthermore

let $\lambda_{k+1} = \mu_k$ and $\mu_{k+1} = a_{k+1} + \alpha(b_{k+1} - a_{k+1})$ and evaluate $f(\mu_{k+1})$ and go to step 4

Step 3: let $a_{k+1} = a_k$ and $b_{k+1} = \mu_k$ furthermore

let $\mu_{k+1} = \lambda_k$ and $\lambda_{k+1} = a_{k+1} + (1-\alpha)(b_{k+1} - a_{k+1})$ and evaluate $f(\lambda_{k+1})$ and go to step 4

Step 4: replace k by $k+1$ and go to step 1

2.3. The Trust Region method

The concept of the trust-region first appears in a paper of Levenberg (1944) and Marquardt (1963) for solving nonlinear least squares problems. Trust-region methods are iterative numerical procedures like the line-search methods in which an approximation of the objective function $f(x)$ by a model $m_k(x_k + s)$ is computed in a neighborhood of the current iterate x_k, which we refer to as the trust-region. The model $m_k(x_k + s)$ should be constructed so that it is easier to handle than $f(x)$ itself. Let us assume for this that our function f is of class c^2.

We solve the following sub problem to obtain the next iteration at each step k of a trust-region method,

$$\begin{cases} \quad min \quad m_k(x_k + s) = f(x_k) + < g(x_k), s > + \frac{1}{2} < s, H(x_k)s > \\ subject\ to \qquad B_k = \{x_k + s : s \in R^n \ and \ \|s\| \leq \Delta_k\}, \end{cases} \tag{1}$$

where $\Delta_k > 0$ is the trust-region radius and $\| \, . \, \|$ is defined to be the Euclidean norm. These sub problems are constrained optimization problems in which the objective function and the constraint are both quadratic. The constraint is a quadratic inequality constraint and can be written as $-s^T s + \Delta_k^2 \geq 0$. In fact, usually, the model $m_k(x_k + s)$ is a quadratic function which is truncated from a Taylor series for f around the point x_k:

$$m_k(x_k + s) = f(x_k) + < g(x_k), s > + \frac{1}{2} < s, H(x_k)s > \qquad k \in N_0$$

We note that one can choose any other norm in the formulations. In this project we will consider the Euclidean norm $\| \, . \, \| = \| \, . \, \|_2$ since it makes some computations easier. For using other kinds of norms we refer to [26].

Hence, our trust-region for the model $m_k(x_k + s)$ is bounded neighborhoods of the current iterate x_k:

$$B_k = \{x_k + s : s \in R^n \ and \ \|s\| \leq \Delta_k\}$$

After constructing the model $m_k(x_k + s)$ and its trust-region, then, one seeks a trial step s to the next iteration $x_{k+1} = x_k + s$ which will result in a reduction for the model while the size of the step is bounded by B_k.

Then, the objective function is evaluated at $x_k + s$ to compare its value to the one predicted by the model at this point. If the sufficient reduction predicted by the model is accomplished by the objective function, $x_k + s$ is accepted as the next iterate and the trust-region is possibly expanded to include this new point (i.e., Δ_k increases). If the reduction in the model is a poor predictor of the actual reduction of the objective function, then the trial point is rejected.

We conclude that the trust-region is too large and the size of the trust-region is reduced (i.e., Δ_k decreases), with the hope that the model provides a better prediction in the smaller region. The illustration of the trust-region steps can be seen in Figure 6.3 (cf. [36]).

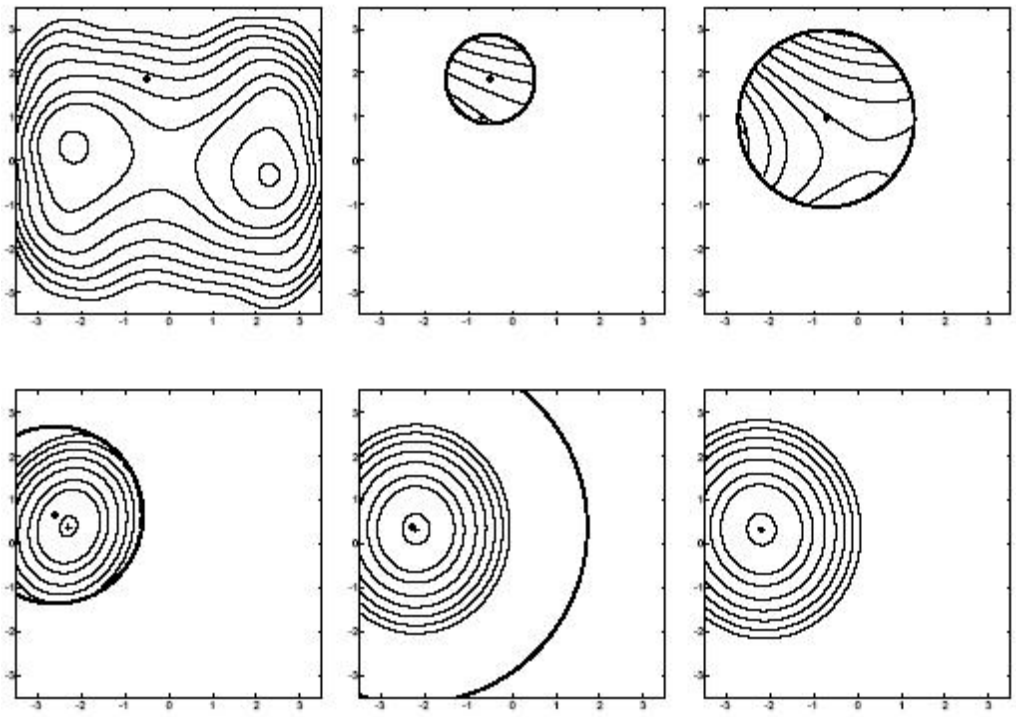


Figure 6.3.1: Six iterations of a trust-region algorithm [36].

Outline and Properties of the Trust-Region Algorithm

In a trust-region algorithm, a strategy for determining the trust-region radius Δ_k at each iteration is needed to be developed. The trust-region radius can be determined by looking at the agreement between the model function m_k and the objective function f at previous iterations. Given a step s_k, we define the ratio

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} = \frac{\text{actual reduction}}{\text{predicted reduction}} \qquad (2)$$

There are various definitions for ρ_k in the mathematics literature, but we shall prefer (2) in particular here. We note that the denominator of ρ_k, namely, the predicted reduction, is always nonnegative since the step s_k is computed from the sub problem (1) over a region that includes the step $s = 0$.

In fact, ρ_k can be called a measure of how good the model $m_k(s)$ predicts the reduction in the f value. If ρ_k is closer to zero or negative, then the actual prediction is much smaller than the predicted one. This indicates that the model cannot be trusted in this region with radius Δ_k. Thus, s_k will be rejected and Δ_k will be reduced. On the other hand, if ρ_k is close to 1, an adequate prediction is obtained. We safely expand the trust-region since the model can be trusted over a wider region, i.e., Δ_k should be increased. If ρ_k is positive but not close to 1, then the trust-region radius is not changed.

Let us state the following result

Theorem For our minimization problem, let x_0 be a given initial point and $\{x_k\}_{k\in N}$ be defined by the trust-region algorithm as stated above. We assume:

(a) The lower level set $L^f(x_0) := \{x \in R | f(x) \leq f(x_0)\}$ is bounded (hence, by (b), compact).

(b) $f \in C^2$ i.e., f, ∇f and $\nabla^2 f$ are continuous in $L^f(x_0)$.

Then, it is fulfilled:

$$\lim_{k\to\infty} \nabla f(x_k) = 0_n$$

Proof See the book of Nash and Sofer [38].

This theorem implies convergence of the gradients to zero; however, it does not say that $\{x_k\}_{k\in N}$ tends to a local minimize. To obtain a stronger theorem which states convergence towards a local solution x^*, additional assumptions must be imposed on the problem.

Here, a brief review will be helpful to follow the integration of trust-region methods into DFO algorithm.

Main steps of a typical trust-region method

1. Given a current iterate, build a good local approximation model.

2. Choose a neighborhood around the current iterate where the model is 'trusted'to be accurate. Minimize the model in this neighborhood.
3. Determine if the step is successful by evaluating the true objective function at the new point comparing the true reduction in value of the objective with the reduction predicted by the model.
4. If the step is successful, accept the new point as the next iterate. Increase the size of the trust-region, if the success is really significant. Otherwise, reject the new point and reduce the size of the trust-region.
5. Repeat until convergence.

Quadratic Interpolation

Consider the problem of interpolating a given or suitably chosen function $f(x)$ from R^n into R by a quadratic polynomial $Q(x)$ at a chosen set of points $y = \{y_1, \dots, y_p\} \subseteq R^n$. The quadratic polynomial $Q(x)$ is an interpolation of the function $f(x)$ with respect to the set y if

$$Q(y_j) = f(y_j) \;\; j = 1,2, \dots, p$$

such that f is known at all the finitely many elements of y . Here, we note that Q is our model which we defined as m_k in above within the context of trust-region methods; i.e., $m_k(x) = Q(x)$.

Suppose that the space of quadratic polynomials is spanned by a set of basis functions $\{\Phi_i(.)\}_{i=1}^{p}$. Then, any quadratic polynomial can be written in terms of these basis functions, i.e., $Q(x) = \sum_{i=1}^{p} \alpha_i \, \Phi_i(x)$, where the coefficient vector $\alpha_i, i = 1, \dots, p$ is to be determined. We need

$$p = 1 + n + n + \frac{n(n-1)}{2} = \frac{1}{2}(n+1)(n+2)$$

points to find all of the interpolation parameters. If we have $\frac{1}{2}(n+1)(n+2)$points, we can ensure that the quadratic model is entirely determined by the following system of equations. When this is the case, the system of linear equations

$$\sum_{i=1}^{p} \alpha_i \, \Phi_i(y_i) = f(y_i) \qquad\qquad \forall y_i \in y$$

can be solved to derive interpolation parameters.

The parameter or coefficient matrix of this system is looks as follows:

$$m = \begin{pmatrix} \Phi_1(y_1) & \cdots & \Phi_1(y_p) \\ \vdots & \ddots & \vdots \\ \Phi_p(y_1) & \cdots & \Phi_p(y_p) \end{pmatrix}$$

For a given set of points and a set of function values, an interpolation polynomial exists and is unique if and only if the above matrix is nonsingular. Theoretically, this means that the system can be solved, but in practice the solvability of this system depends on whether the matrix is ill-conditioned or not.

From the above arguments, we conclude that if we manage to determine the quadratic polynomial uniquely, then we have

$$p = \frac{1}{2}(n+1)(n+2)$$

However, we need to be aware of the fact that not any $\frac{1}{2}(n+1)(n+2)$ points in R^n can be interpolated by a quadratic polynomial. Obviously, although 3 distinct points can be interpolated by a quadratic function in univariate interpolation, this is not the case in multivariate interpolation. In fact, 3 points will not be enough to obtain a quadratic interpolation polynomial whenever the dimension of the interpolation space is greater than one. By inspection, one can see that 6 points are necessary to obtain a unique quadratic interpolation of a function in two dimensions. However, an interpolation set y of six points lying on one line cannot be interpolated by a quadratic function. Therefore, the points of y must satisfy a geometric condition to ensure the existence and uniqueness of the quadratic model. This geometric condition is known as the poisedness of the point set.

Definition A set of points y is called poised, with respect to a given subspace of polynomials, if the considered function $f(x)$ can be interpolated at the points of y by the polynomials from this subspace, i.e., if there always exists a suitable interpolating polynomial in that subspace.

Remark In DFO, poisedness is a necessary geometric condition on the interpolation set y that ensures the existence and uniqueness of the quadratic model $Q(x)$ wanted and used in DFO algorithm.

We illustrate the implied geometric character of poisedness by the following examples.

Example supposes $n = 2$ and y is a set of six points on a unit circle. Then, $y \subset R^2$ cannot be interpolated by a polynomial of the form

$$a_0 + a_1x_1 + a_2x_2 + a_{1,1}x_1^2 + a_{1,2}x_1x_2 + a_{2,2}x_2^2$$

Hence, y is not poised with respect to the space of quadratic polynomials. On the other hand, y can be interpolated by a polynomial of the form

$$a_0 + a_1x_1 + a_2x_2 + a_{1,1}x_1^2 + a_{1,2}x_1x_2 + a_{1,1}x_1^3$$

Therefore, y is poised in an appropriate subspace of the space of cubic polynomials.

We present one more example for the non-poised case to have a well understanding of the notion of poisedness from [12]:

Example Consider the two quadrics $q1(x,y) = 2x + x^2 - y^2$ and $q2(x,y) = x^2 + y^2$, whose intersection curve I projects in the (x,y)-plane to the conics $C(x,y) = 0$ with $C(x,y) = x - y^2$. Namely,

$$2x + x^2 - y^2 = x^2 + y^2,$$
$$\Leftrightarrow \quad 2x = 2y^2,$$
$$\Leftrightarrow \quad x = y^2$$

If one tries to interpolate a height function using points on I, uniqueness of the interpolant is not achieved since any quadric in the pencil of $q1$ and $q2$ goes through I.

Definition A set of points y is called well-poised, if it remains poised under small perturbations.

For example, if n = 2, six points almost on a line may define a poised set. However, since some small perturbation of the points might make them aligned, it is not a well-poised set.

As we mentioned before, a set of points is poised if m is nonsingular with respect to the space of quadratic polynomials. If we look for an understanding of this by the fact that an interpolation polynomial exists and is unique if and only if m is nonsingular, then we conclude: y is poised if the determinant of m is nonvanishing, i.e., if

$$\delta(\mathrm{y}) = \det(\mathrm{m}) \neq 0$$

The measure of poisedness in a DFO algorithm can be explained by a methodology based on Newton fundamental polynomials [33]. In DFO, the approach of handling the poisedness in combination with the Newton fundamental polynomials is a distinctive issue. This is so, because it allows us not only to choose a good interpolation set from a given set of sample points but also to find a new sample point which improves the poisedness of the interpolation set. If we had no such a useful tool, then, removing a point from the set would have caused the

conditioning of the coefficient matrix to get worse in the updating step for the interpolation set of DFO. There is also a detailed work on DFO in which the quadratic approximation model is determined by Lagrange interpolation polynomials in [9] instead of the Newton fundamental polynomials.

Let us focus on the Newton fundamental points [33]: The points y in our interpolation set $y = \{y_1, \dots, y_p\} \subseteq R^n$ are organized into $d+1$ blocks, where $y^l \ (l = 0,1,2, \dots, d)$ is the l^{th} block, containing $| y^l | = \binom{l+n-1}{l}$ points.

Definition A single Newton fundamental polynomial of degree l corresponds to each point $(y_i)^l \in y^l$ satisfying the following conditions:

$N_i^l (y_j)^m = \delta_{i,j}\, \delta_{lm}$ for all $(y_j)^m \in y^l$ with $m \in \{0,1,2, \dots, l\}$

Here, we refer to Kronecker's symbol for $i, j = 0,1,2, \dots, l$:

$$\delta_{i,j} := \begin{cases} 1, & if\ i = j \\ 0, & else \end{cases}$$

Consider the set of interpolation points being partitioned into three disjoint blocks y^0, y^1, y^2, which correspond to the constant term, the linear terms and the quadratic terms of a quadratic polynomial, respectively

Hence, y^0 has a single element, y^1 has n elements and y^2 has $\frac{n(n+1)}{2}$ elements. The basis $\{N_i(.)\}$ of NFP is also partitioned into three blocks $\{N_i^0(.)\}, \{N_i^1(.)\}\ and\ \{N_i^2(.)\}$ with the appropriate number of elements in each block.

The unique element of $\{N_i^0(.)\}$ is a polynomial of degree zero, each of the n elements of $\{N_i^1(.)\}$ is a polynomial of degree one and, finally, each of the $\frac{n(n+1)}{2}$ elements of $\{N_i^2(.)\}$ is a polynomial of degree two.

The basis elements and the interpolation points are set in one-to-one correspondence, so that the points from block y^l correspond to polynomials from block $\{N_i^l(.)\}$. A Newton fundamental polynomial $\{N_i(.)\}$ and a point y_i are in correspondence with each other if and only if the value of that polynomial at that point is one and its value at any other point in the

same block or in any previous block is zero. In other words, if y_i corresponds to N_i, then, $N_i(y_i) = 1$ and $N_i(y_j) = 0$ for all other indices j.

Example Consider the quadratic interpolation on a plane. We require six interpolation points using three blocks:

$y^0 = \{(0,0)\},\ y^1 = \{(1,0),(0,1)\},\ and\ \ y^2 = \{(2,0),(1,1),(0,2)\}$

corresponding to the initial basis functions $1, x_1, x_2, x_1^2, x_1x_2\ and\ x_2^2$, respectively.

Applying the procedure from [53], we find the basis of NFP:

$$N_1^0 = 1, \qquad N_1^1 = x_1, \qquad N_2^1 = x_2, \qquad N_1^2 = \frac{x_1^2 - x_1}{2}, \qquad N_2^2 = x_1x_2, \qquad N_3^2 = \frac{x_2^2 - x_2}{2}$$

Now, we are ready to present an outline of the DFO algorithm

2.3.1. Algorithm for trust-region method

Step 1: Initialization

For a given x_s and $f(x_s)$ choose an initial interpolation set y. Determine $x_0 \in y$ such that

$$f(x_0) = \min_{y_i \in y} f(y_i).$$

Choose an initial trust region radius $\Delta_k > 0$. Set $\mathrm{k} = 0$.

Choose parameters $0 < \mu_0 < \mu_1 < 1 \quad and \quad 0 < \gamma_0 \le \gamma_1 < 1 \le \gamma_2$

Step 2: Model building

Using the poised interpolation set y and basis of NFP, build the model $m_k(x_k + s)$:

Minimize the model within the trust-region

$$B_k = \{x_k + s : s \in R^n\ and\ \|s\| \le \Delta_k\},$$

Compute the point x_k^+ such that

$$m_k(x_k^+) = \min_{B_k} m_k(x)$$

Compute $f(x_k^+)$ and the ratio

$$\rho_k = \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}$$

Step 3: Update the interpolation set

-Successful step:

If $\rho_k \geq \mu_0$, include x_k^+ in y by dropping one of the existing interpolation points.

-Unsuccessful step:

If $\rho_k < \mu_0$, and y is inadequate in $x \in B_k$ improve the geometry.

Step 4: Update the trust-region radius

-Successful step:

If $\rho_k \geq \mu_1$, then set $\Delta_{k+1} \in [\Delta_k, \gamma_2 \Delta_k]$

-Unsuccessful step:

If $\rho_k < \mu_0$, and y is adequate in B_k then set $\Delta_{k+1} \in [\gamma_0 \Delta_k, \gamma_2 \Delta_k]$

Otherwise, set $\Delta_{k+1} = \Delta_k$,

Step 5: Update the current iterate

Determine $\hat{x}_k$ such that

$$m_k(\hat{x}_k) = \min_{y_i \in y, y_i \neq x_k} m_k(y_i)$$

Then, if

$$\hat{\rho}_k = \frac{f(x_k) - f(\hat{x}_k)}{m_k(x_k) - m_k(x_k^+)} \geq \mu_0,$$

set $x_{k+1} = \hat{x}_k$ otherwise $x_{k+1} = x_k$. Increment k by one and go to Step 1.

3. Test functions and experimental setup

We choose two simple problems to illustrate some of the differences between the above three methods for derivative free optimization. The above three algorithms are implemented on matlab environment and the matlab codes are written in appendices.

3.1. Example1

$$min\ \ f(x,y) = x^2 - 4x + y^2 - y - xy$$

3.1.1. Solve Example1 by using Nelder-Mead method

Start with three vertices $y_1 = \begin{pmatrix}0\\0\end{pmatrix}$ $y_2 = \begin{pmatrix}1.2\\0\end{pmatrix}$ $y_3 = \begin{pmatrix}0\\0.8\end{pmatrix}$

n	y_1	y_2	y_3	f_1	f_2	f_3
0	$\begin{pmatrix}1.2\\0\end{pmatrix}$	$\begin{pmatrix}0\\0.8\end{pmatrix}$	$\begin{pmatrix}0\\0\end{pmatrix}$	-3.3600	-0.1600	0
1	$\begin{pmatrix}1.8\\1.2\end{pmatrix}$	$\begin{pmatrix}1.2\\0\end{pmatrix}$	$\begin{pmatrix}0\\0.8\end{pmatrix}$	-5.8800	-3.3600	-0.1600
2	$\begin{pmatrix}1.8\\1.2\end{pmatrix}$	$\begin{pmatrix}3\\0.4\end{pmatrix}$	$\begin{pmatrix}1.2\\0\end{pmatrix}$	-5.8800	-4.4400	-3.3600
3	$\begin{pmatrix}3.6\\1.6\end{pmatrix}$	$\begin{pmatrix}1.8\\1.2\end{pmatrix}$	$\begin{pmatrix}3\\0.4\end{pmatrix}$	-6.2400	-5.8800	-4.4400
4	$\begin{pmatrix}3.6\\1.6\end{pmatrix}$	$\begin{pmatrix}2.4\\2.4\end{pmatrix}$	$\begin{pmatrix}1.8\\1.2\end{pmatrix}$	-6.2400	-6.2400	-5.8800
5	$\begin{pmatrix}3.6\\2.4\end{pmatrix}$	$\begin{pmatrix}3.6\\1.6\end{pmatrix}$	$(2.4,2.4)$	-6.7200	-6.2400	-6.2400
6	$\begin{pmatrix}3\\2.2\end{pmatrix}$	$\begin{pmatrix}3.6\\2.4\end{pmatrix}$	$\begin{pmatrix}3.6\\1.6\end{pmatrix}$	-6.9600	-6.7200	-6.2400
7	$\begin{pmatrix}3\\2.2\end{pmatrix}$	$\begin{pmatrix}3.45\\1.95\end{pmatrix}$	$\begin{pmatrix}3.6\\2.4\end{pmatrix}$	-6.9600	-6.7725	-6.7200
8	$\begin{pmatrix}3\\2.2\end{pmatrix}$	$\begin{pmatrix}2.85\\1.75\end{pmatrix}$	$\begin{pmatrix}3.45\\1.95\end{pmatrix}$	-6.9600	-6.9525	-6.7725
9	$\begin{pmatrix}3\\2.2\end{pmatrix}$	$\begin{pmatrix}3.1875\\1.9625\end{pmatrix}$	$\begin{pmatrix}2.85\\1.75\end{pmatrix}$	-6.9600	-6.9564	-6.9525

10	$\begin{pmatrix}2.9719\\1.9156\end{pmatrix}$	$\begin{pmatrix}3\\2.2\end{pmatrix}$	$\begin{pmatrix}3.1875\\1.9625\end{pmatrix}$	-6.9945	-6.9600	-6.9564

After 33 iterations, 64 function evaluations and 0.5771 times and by using 10^{-9} chosen tolerance we get the solution, $f = -7 \ \ at \ \ x = 3 \ and \ y = 2$

3.1.2. Solve Example1 by using golden section method

Start with this interval

$x \in (2.5,3.5)$ $y \in (1.5,2.5)$

n	a_n	b_n	λ_n	μ_n	$f(\lambda_n)$	$f(\mu_n)$
0	$\begin{pmatrix}2.5\\1.5\end{pmatrix}$	$\begin{pmatrix}3.5\\2.5\end{pmatrix}$	$\begin{pmatrix}2.882\\1.882\end{pmatrix}$	$\begin{pmatrix}3.118\\2.118\end{pmatrix}$	-6.9861	-6.9861
1	$\begin{pmatrix}2.5\\1.5\end{pmatrix}$	$\begin{pmatrix}3.118\\2.118\end{pmatrix}$	$\begin{pmatrix}2.7361\\1.7361\end{pmatrix}$	$\begin{pmatrix}2.882\\1.882\end{pmatrix}$	-6.9303	-6.9861
2	$\begin{pmatrix}2.7361\\1.7361\end{pmatrix}$	$\begin{pmatrix}3.118\\2.118\end{pmatrix}$	$\begin{pmatrix}2.882\\1.882\end{pmatrix}$	$\begin{pmatrix}2.9721\\1.9721\end{pmatrix}$	-6.9861	-6.9992
3	$\begin{pmatrix}2.882\\1.882\end{pmatrix}$	$\begin{pmatrix}3.118\\2.118\end{pmatrix}$	$\begin{pmatrix}2.9721\\1.9721\end{pmatrix}$	$\begin{pmatrix}3.0278\\2.0278\end{pmatrix}$	-6.9992	-6.9992
4	$\begin{pmatrix}2.9721\\1.9721\end{pmatrix}$	$\begin{pmatrix}3.118\\2.118\end{pmatrix}$	$\begin{pmatrix}3.0278\\2.0278\end{pmatrix}$	$\begin{pmatrix}3.0623\\2.0623\end{pmatrix}$	-6.9992	-6.9961
5	$\begin{pmatrix}2.9721\\1.9721\end{pmatrix}$	$\begin{pmatrix}3.0623\\2.0623\end{pmatrix}$	$\begin{pmatrix}3.0065\\2.0065\end{pmatrix}$	$\begin{pmatrix}3.0278\\2.0278\end{pmatrix}$	-7.0000	-6.9992
6	$\begin{pmatrix}2.9721\\1.9721\end{pmatrix}$	$\begin{pmatrix}3.0278\\2.0278\end{pmatrix}$	$\begin{pmatrix}2.9934\\1.9934\end{pmatrix}$	$\begin{pmatrix}3.0065\\2.0065\end{pmatrix}$	-7.0000	-7.0000
7	$\begin{pmatrix}2.9934\\1.9934\end{pmatrix}$	$\begin{pmatrix}3.0278\\2.0278\end{pmatrix}$	$\begin{pmatrix}3.0065\\2.0065\end{pmatrix}$	$\begin{pmatrix}3.0147\\2.0147\end{pmatrix}$	-7.0000	-6.9998
8	$\begin{pmatrix}2.9934\\1.9934\end{pmatrix}$	$\begin{pmatrix}3.0147\\2.0147\end{pmatrix}$	$\begin{pmatrix}3.0015\\2.0015\end{pmatrix}$	$\begin{pmatrix}3.0065\\2.0065\end{pmatrix}$	-7.0000	-7.0000

9	$\begin{pmatrix}2.9934\\1.9934\end{pmatrix}$	$\begin{pmatrix}3.0065\\2.0065\end{pmatrix}$	$\begin{pmatrix}2.9984\\1.9984\end{pmatrix}$	$\begin{pmatrix}3.0015\\2.0015\end{pmatrix}$	-7.0000	-7.0000
10	$\begin{pmatrix}2.9984\\1.9984\end{pmatrix}$	$\begin{pmatrix}3.0065\\2.0065\end{pmatrix}$	$\begin{pmatrix}3.0015\\2.0015\end{pmatrix}$	$\begin{pmatrix}3.0034\\2.0034\end{pmatrix}$	-7.0000	-7.0000

After 16 iterations, 17 function evaluations and 0.3633 times and by using 10^{-3} chosen tolerance we get the solution, $f = -7 \;\; at \;\; x = 3 \;\; and \;\; y = 2$

3.1.3. Solve Example1 by using Trust Region method

Start with $y_0 = \begin{pmatrix}2.55\\1.75\end{pmatrix}$

n	y_n	f_n
0	$\begin{pmatrix}2.55\\1.75\end{pmatrix}$	-6.8475
1	$\begin{pmatrix}2.875\\1.75\end{pmatrix}$	-6.9531
2	$\begin{pmatrix}2.875\\1.9375\end{pmatrix}$	-6.9882
3	$\begin{pmatrix}2.999\\2.0095\end{pmatrix}$	-6.9999
4	$\begin{pmatrix}2.999\\1999.5\end{pmatrix}$	-6.9999
5	$\begin{pmatrix}2.9999\\2.0000\end{pmatrix}$	-6.9999
6	$\begin{pmatrix}2.9999\\1.999\end{pmatrix}$	-7
7	$\begin{pmatrix}3\\2\end{pmatrix}$	-7

8	$\begin{pmatrix}3\\2\end{pmatrix}$	-7
9	$\begin{pmatrix}3\\2\end{pmatrix}$	-7

After 9 iteration and 9 function evaluations we get the solution which is a minimum,

$$f = -7 \;\; at \;\; x = 3 \;\; and \;\; y = 2$$

3.2. Example2

$$min \;\; f(x,y,z) = 3 - 4x + 2x^2 - 4y + 2xy + 2y^2 + 4z - 2xz - 2yz + 2z^2$$

3.2.1. Solve Example2 by using Nelder-Mead method

Start with fore vertices

$$y_1 = \begin{pmatrix}0\\0\\0\end{pmatrix} \;\; y_2 = \begin{pmatrix}0\\1.2\\0\end{pmatrix} \;\; y_3 = \begin{pmatrix}0\\0\\0.8\end{pmatrix} \;\; y_4 = \begin{pmatrix}1\\0\\0\end{pmatrix}$$

n	y_1	y_2	y_3	y_4	f_1	f_2	f_3	f_4
0	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1.2\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}1\\0\\0.8\end{pmatrix}$	1	1.08	3	7.480
1	$\begin{pmatrix}0.6667\\0.8\\-0.8\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1.2\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	0.7956	1	1.08	3
2	$\begin{pmatrix}0.2778\\0.3333\\-0.1333\end{pmatrix}$	$\begin{pmatrix}0.6667\\0.8\\-0.8\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1.2\\0\end{pmatrix}$	0.7825	0.7956	1	1.08
3	$\begin{pmatrix}0.3241\\0.7889\\-0.1556\end{pmatrix}$	$\begin{pmatrix}0.2778\\0.3333\\-0.1333\end{pmatrix}$	$\begin{pmatrix}0.6667\\0.8\\-0.8\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	0.2866	0.7825	0.7956	1

4	$\begin{pmatrix} 0.7114 \\ 0.3204 \\ -0.1815 \end{pmatrix}$	$\begin{pmatrix} 0.3241 \\ 0.7889 \\ -0.1556 \end{pmatrix}$	$\begin{pmatrix} 0.2778 \\ 0.3333 \\ -0.1333 \end{pmatrix}$	$\begin{pmatrix} 0.6667 \\ 0.8 \\ -0.8 \end{pmatrix}$	0.2606	0.2866	0.7825	0.7956
5	$\begin{pmatrix} 0.5522 \\ 0.6404 \\ -0.4784 \end{pmatrix}$	$\begin{pmatrix} 0.7114 \\ 0.3204 \\ -0.1815 \end{pmatrix}$	$\begin{pmatrix} 0.3241 \\ 0.7889 \\ -0.1556 \end{pmatrix}$	$\begin{pmatrix} 0.2778 \\ 0.3333 \\ -0.1333 \end{pmatrix}$	0.0522	0.2606	0.2866	0.7825
6	$\begin{pmatrix} 0.5522 \\ 0.6404 \\ -0.4784 \end{pmatrix}$	$\begin{pmatrix} 0.6550 \\ 0.7082 \\ -0.3410 \end{pmatrix}$	$\begin{pmatrix} 0.7114 \\ 0.3204 \\ -0.1815 \end{pmatrix}$	$\begin{pmatrix} 0.3241 \\ 0.7889 \\ -0.1556 \end{pmatrix}$	0.0522	0.1343	0.2606	0.2866
7	$\begin{pmatrix} 0.5522 \\ 0.6404 \\ -0.4784 \end{pmatrix}$	$\begin{pmatrix} 0.4818 \\ 0.6726 \\ -0.2446 \end{pmatrix}$	$\begin{pmatrix} 0.6550 \\ 0.7082 \\ -0.3410 \end{pmatrix}$	$\begin{pmatrix} 0.7114 \\ 0.3204 \\ -0.1815 \end{pmatrix}$	0.0522	0.1056	0.1343	0.2606
8	$\begin{pmatrix} 0.5522 \\ 0.6404 \\ -0.4784 \end{pmatrix}$	$\begin{pmatrix} 0.6372 \\ 0.4971 \\ -0.2681 \end{pmatrix}$	$\begin{pmatrix} 0.4818 \\ 0.6726 \\ -0.2446 \end{pmatrix}$	$\begin{pmatrix} 0.6550 \\ 0.7082 \\ -0.3410 \end{pmatrix}$	0.0522	0.0822	0.1056	0.1343
9	$\begin{pmatrix} 0.5522 \\ 0.6404 \\ -0.4784 \end{pmatrix}$	$\begin{pmatrix} 0.6372 \\ 0.4971 \\ -0.2681 \end{pmatrix}$	$\begin{pmatrix} 0.4592 \\ 0.4986 \\ -0.3197 \end{pmatrix}$	$\begin{pmatrix} 0.4818 \\ 0.6726 \\ -0.2446 \end{pmatrix}$	0.0522	0.0822	0.0837	0.1056
10	$\begin{pmatrix} 0.6173 \\ 0.4181 \\ -0.4662 \end{pmatrix}$	$\begin{pmatrix} 0.5522 \\ 0.6404 \\ -0.4784 \end{pmatrix}$	$\begin{pmatrix} 0.6372 \\ 0.4971 \\ -0.2681 \end{pmatrix}$	$\begin{pmatrix} 0.4592 \\ 0.4986 \\ -0.3197 \end{pmatrix}$	0.0216	0.0522	0.0822	0.0837

After 51 iterations, 94 function evaluations and 1.0012 times and by using 10^{-7} chosen tolerance we get the solution, $f = 0$ at $x = 0.5 \quad y = 0.5 \quad and \quad z = -0.5$

3.2.2. Solve Example2 by using golden section method

Start with this interval

$$x \in (0,1) \qquad y \in (-1,1) \quad z \in (-1,0)$$

n	a_n	b_n	λ_n	μ_n	$f(\lambda_n)$	$f(\mu_n)$

0	$\begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.3820 \\ -0.2360 \\ -0.6180 \end{pmatrix}$	$\begin{pmatrix} 0.6180 \\ 0.2360 \\ -0.3820 \end{pmatrix}$	1.1112	0.1672
1	$\begin{pmatrix} 0.3820 \\ -0.2360 \\ -0.6180 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.6180 \\ 0.2360 \\ -0.3820 \end{pmatrix}$	$\begin{pmatrix} 0.7639 \\ 0.5278 \\ -0.2361 \end{pmatrix}$	0.1672	0.1409
2	$\begin{pmatrix} 0.6180 \\ 0.2360 \\ -0.3820 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.7639 \\ 0.5278 \\ -0.2361 \end{pmatrix}$	$\begin{pmatrix} 0.8541 \\ 0.7082 \\ -0.1459 \end{pmatrix}$	0.1409	0.3374
3	$\begin{pmatrix} 0.6180 \\ 0.2360 \\ -0.3820 \end{pmatrix}$	$\begin{pmatrix} 0.8541 \\ 0.7082 \\ -0.1459 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	$\begin{pmatrix} 0.7639 \\ 0.5278 \\ -0.2361 \end{pmatrix}$	0.1007	0.1409
4	$\begin{pmatrix} 0.6180 \\ 0.2360 \\ -0.3820 \end{pmatrix}$	$\begin{pmatrix} 0.7639 \\ 0.5278 \\ -0.2361 \end{pmatrix}$	$\begin{pmatrix} 0.6737 \\ 0.3475 \\ -0.3263 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	0.1069	0.1007
5	$\begin{pmatrix} 0.6737 \\ 0.3475 \\ -0.3263 \end{pmatrix}$	$\begin{pmatrix} 0.7639 \\ 0.5278 \\ -0.2361 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	$\begin{pmatrix} 0.7295 \\ 0.4589 \\ -0.2705 \end{pmatrix}$	0.1007	0.1087
6	$\begin{pmatrix} 0.6737 \\ 0.3475 \\ -0.3263 \end{pmatrix}$	$\begin{pmatrix} 0.7295 \\ 0.4589 \\ -0.2705 \end{pmatrix}$	$\begin{pmatrix} 0.6950 \\ 0.3901 \\ -0.3050 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	0.1002	0.1007
7	$\begin{pmatrix} 0.6737 \\ 0.3475 \\ -0.3263 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	$\begin{pmatrix} 0.6869 \\ 0.3738 \\ -0.3131 \end{pmatrix}$	$\begin{pmatrix} 0.6950 \\ 0.3901 \\ -0.3050 \end{pmatrix}$	0.1017	0.1002
8	$\begin{pmatrix} 0.6869 \\ 0.3738 \\ -0.3131 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	$\begin{pmatrix} 0.6950 \\ 0.3901 \\ -0.3050 \end{pmatrix}$	$\begin{pmatrix} 0.7001 \\ 0.4001 \\ -0.2999 \end{pmatrix}$	0.1002	0.1000
9	$\begin{pmatrix} 0.6950 \\ 0.3901 \\ -0.3050 \end{pmatrix}$	$\begin{pmatrix} 0.7082 \\ 0.4164 \\ -0.2918 \end{pmatrix}$	$\begin{pmatrix} 0.7001 \\ 0.4001 \\ -0.2999 \end{pmatrix}$	$\begin{pmatrix} 0.7032 \\ 0.4063 \\ -0.2968 \end{pmatrix}$	0.1000	0.1001

10	$\begin{pmatrix} 0.6950 \\ 0.3901 \\ -0.3050 \end{pmatrix}$	$\begin{pmatrix} 0.7032 \\ 0.4063 \\ -0.2968 \end{pmatrix}$	$\begin{pmatrix} 0.6981 \\ 0.3963 \\ -0.3019 \end{pmatrix}$	$\begin{pmatrix} 0.7001 \\ 0.4001 \\ -0.2999 \end{pmatrix}$	0.1000	0.1000

After 21 iterations, 22 function evaluations and 0.4696 times and by using 10^{-4} we get the solution,

$$f = 0.1 \ \ at \ \ x = 0.7 \quad y = 0.4 \quad and \quad z = -0.3$$

3.2.3. Solve Example2 by using Trust Region method

Start with $y_0 = (1,0.5,0)$

n	y_n	f_n
0	$\begin{pmatrix} 1 \\ 0.5 \\ 0 \end{pmatrix}$	0.5
1	$\begin{pmatrix} 0.75 \\ 0.5 \\ 0 \end{pmatrix}$	0.375
2	$\begin{pmatrix} 0.75 \\ 0.625 \\ 0 \end{pmatrix}$	0.3437
3	$\begin{pmatrix} 0.75 \\ 0.625 \\ -0.3124 \end{pmatrix}$	0.148
4	$\begin{pmatrix} 0.531 \\ 0.625 \\ -0.3124 \end{pmatrix}$	0.053
5	$\begin{pmatrix} 0.531 \\ 0.578 \\ -0.3124 \end{pmatrix}$	0.048
6	$\begin{pmatrix} 0.531 \\ 0.578 \\ -0.445 \end{pmatrix}$	0.013

7	$\begin{pmatrix} 0.488 \\ 0.578 \\ -0.445 \end{pmatrix}$	0.009
8	$\begin{pmatrix} 0.488 \\ 0.533 \\ -0.445 \end{pmatrix}$	0.005
9	$\begin{pmatrix} 0.488 \\ 0.533 \\ -0.489 \end{pmatrix}$	0.0015
10	$\begin{pmatrix} 0.488 \\ 0.533 \\ -0.489 \end{pmatrix}$	0.00147

After 15 iteration and 15 function evaluations we get the solution

$$f = 0 \; at \;\; x = 0.5 \qquad y = 0.5 \quad and \quad z = -0.5$$

3.3. Example3

$$min \;\; f(x,y) = (x-2)^4 + ((x-2)y)^2 + (y+1)^2$$

3.3.1. Solve Example3 by using Nelder-Mead method

Start with three vertices

$$y_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad y_2 = \begin{pmatrix} 0 \\ -1.8 \end{pmatrix} \quad y_3 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

n	y_1	y_2	y_3	f_1	f_2	f_3
0	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -1.8 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0 \end{pmatrix}$	17	29.6	82
1	$\begin{pmatrix} 2 \\ -2.7 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -1.8 \end{pmatrix}$	2.89	17	29
2	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 2 \\ -2.7 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	0.01	2.89	17

3	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 2 \\ -2.7 \end{pmatrix}$	0.01	1.82	2.89
4	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 1.75 \\ -1.8 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -0.9 \end{pmatrix}$	0.01	0.8464	1.82
5	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 1.4375 \\ -1.125 \end{pmatrix}$	$\begin{pmatrix} 1.75 \\ -1.8 \end{pmatrix}$	0.01	0.5162	0.8464
6	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 1.7031 \\ -0.6188 \end{pmatrix}$	$\begin{pmatrix} 1.4375 \\ -1.125 \end{pmatrix}$	0.01	0.1869	0.5162
7	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 2.0586 \\ -0.5766 \end{pmatrix}$	$\begin{pmatrix} 1.7031 \\ -0.6188 \end{pmatrix}$	0.01	0.1805	0.1869
8	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 2.3555 \\ -0.8578 \end{pmatrix}$	$\begin{pmatrix} 2.0586 \\ -0.5766 \end{pmatrix}$	0.01	0.1292	0.1805
9	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 2.2373 \\ -1.0301 \end{pmatrix}$	$\begin{pmatrix} 2.3555 \\ -0.8578 \end{pmatrix}$	0.01	0.0638	0.1292
10	$\begin{pmatrix} 2 \\ -0.9 \end{pmatrix}$	$\begin{pmatrix} 1.8818 \\ -1.0723 \end{pmatrix}$	$\begin{pmatrix} 2.2373 \\ -1.0301 \end{pmatrix}$	0.01	0.0215	0.0638

After 33 iterations, 64 function evaluations and 0.5074 seconds and by using 10^{-7} chosen tolerance we get the solution, $f = 0$ at $x = 2 \quad and \quad y = -1$

3.3.2. Solve Example3 by using golden section method

Start with this interval

$$x \in (1,3) \qquad y \in (-2,0)$$

n	a_n	b_n	λ_n	μ_n	$f(\lambda_n)$	$f(\mu_n)$
0	$\begin{pmatrix} 1 \\ -2 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1.764 \\ -1.2360 \end{pmatrix}$	$\begin{pmatrix} 2.2360 \\ -0.7640 \end{pmatrix}$	0.1439	0.0913
1	$\begin{pmatrix} 1.7640 \\ -1.2360 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 2\ .2360 \\ -0.7640 \end{pmatrix}$	$\begin{pmatrix} 2.5278 \\ -0.4722 \end{pmatrix}$	0.0913	0.4184

2	$\begin{pmatrix}1.7640\\-1.2360\end{pmatrix}$	$\begin{pmatrix}2.5278\\-0.4722\end{pmatrix}$	$\begin{pmatrix}2.0558\\-0.9442\end{pmatrix}$	$\begin{pmatrix}2.2360\\-0.7640\end{pmatrix}$	0.0059	0.0913
3	$\begin{pmatrix}1.7640\\-1.2360\end{pmatrix}$	$\begin{pmatrix}2.2360\\-0.7640\end{pmatrix}$	$\begin{pmatrix}1.9443\\-1.0557\end{pmatrix}$	$\begin{pmatrix}2.0558\\-0.9442\end{pmatrix}$	0.0066	0.0059
4	$\begin{pmatrix}1.9443\\-1.0557\end{pmatrix}$	$\begin{pmatrix}2.2360\\-0.7640\end{pmatrix}$	$\begin{pmatrix}2.0558\\-0.9442\end{pmatrix}$	$\begin{pmatrix}2.1246\\-0.8754\end{pmatrix}$	0.0059	0.0277
5	$\begin{pmatrix}1.9443\\-1.0557\end{pmatrix}$	$\begin{pmatrix}2.1246\\-0.8754\end{pmatrix}$	$\begin{pmatrix}2.0132\\-0.9868\end{pmatrix}$	$\begin{pmatrix}2.0558\\-0.9442\end{pmatrix}$	0.0003	0.0059
6	$\begin{pmatrix}1.9443\\-1.0557\end{pmatrix}$	$\begin{pmatrix}2.0558\\-0.9442\end{pmatrix}$	$\begin{pmatrix}1.9869\\-1.0131\end{pmatrix}$	$\begin{pmatrix}2.0132\\-0.9868\end{pmatrix}$	0.0003	0.0003
7	$\begin{pmatrix}1.9869\\-1.0131\end{pmatrix}$	$\begin{pmatrix}2.0558\\-0.9442\end{pmatrix}$	$\begin{pmatrix}2.0132\\-0.9868\end{pmatrix}$	$\begin{pmatrix}2.0295\\-0.9705\end{pmatrix}$	0.0003	0.0017
8	$\begin{pmatrix}1.9869\\-1.0131\end{pmatrix}$	$\begin{pmatrix}2.0295\\-0.9705\end{pmatrix}$	$\begin{pmatrix}2.0032\\-0.9968\end{pmatrix}$	$\begin{pmatrix}2.0132\\-0.9868\end{pmatrix}$	0	0.0003
9	$\begin{pmatrix}1.9869\\-1.0131\end{pmatrix}$	$\begin{pmatrix}2.0132\\-0.9868\end{pmatrix}$	$\begin{pmatrix}1.9969\\-1.0031\end{pmatrix}$	$\begin{pmatrix}2.0032\\-0.9968\end{pmatrix}$	0	0
10	$\begin{pmatrix}1.9869\\-1.0131\end{pmatrix}$	$\begin{pmatrix}2.0032\\-0.9968\end{pmatrix}$	$\begin{pmatrix}1.9931\\-1.0069\end{pmatrix}$	$\begin{pmatrix}1.9969\\-1.0031\end{pmatrix}$	0	0

After 22 iterations, 23 function evaluations and 0.5616 seconds and by using 10^{-4} we get the solution,

$$f = 0 \ \ at \ \ x = 2 \quad and \quad y = -1$$

3.3.3. **Solve Example3 by using Trust Region method**

Start with $y_0 = (1.5, -2)$

n	y_n	f_n
0	$\begin{pmatrix}2\\-2\end{pmatrix}$	1
1	$\begin{pmatrix}2\\-0.5\end{pmatrix}$	0.25

After 1 iteration and 1 function evaluations we get the solution

$$f = 025\ at\ \ x = 2 \quad and \quad y = -0.5$$

4. Conclusion

Let us briefly summarize the approaches which the line-search, the trust region methods and Nelder-Mead method use for finding the solution of the problem.

The line-search algorithms choose a search direction p_k at the current iteration x_k and, then, take a step for the new iteration with a lower function value along this direction. This step has the length of α. Herewith, line-search finds an approximate value of α by generating a limited number of the trial step lengths.

Trust-region algorithms construct a model function m_k to be used instead of the actual objective function f. This model which is easier to handle than the objective function itself is constructed by means of the gradient and Hessian information of f and by some f-values which are already in our hands. The search directions p which will minimize the model m_k is being looked for in some region around the current iterate x_k. The size of this region is adjusted according to the sufficiency of the decrease in f at every iteration step.

We conclude that the line-search, the trust-region and Nelder-Mead method approaches differ in choosing the direction and the distance of the move to the next iterate. In line-search algorithms, the search direction p_k is being fixed firstly and, then, an appropriate distance α_k is identified. In trust-region algorithms, however, a region around the current iterate is chosen with a maximum radius Δ_k, then a direction is being looked for to obtain the sufficient decrease in f.

Among those three methods thrust region method is fast to get the solution and the number of functional evaluation is also smaller but it is difficult to build the model. Nelder-mead method is simple to get the solution but it evaluates function many time therefore it take many time. Golden section method is also simple but it is not efficiency when the number of variable greater than two.

5. Appendices

5.1. Matlab code for Nelder–Mead method

```
function[x_opt,n_feval]=nelder_mead(x, function_handle,tol)
 rho = 1;

 delta  = 2;

 gama = 0.5;

 sigma = 0.5;

 max_feval = 250;

 [ temp, n_dim ] = size ( x );

 k=0;

 index = 1 : n_dim + 1;

 [ f ]  = evaluate ( x, function_handle );

 n_feval = n_dim + 1;

 [ f, index ] = sort ( f );

 x = x(index,:);

 converged = 0;

 diverged  = 0;

 while ( ~converged && ~diverged )

     display(x);

     display(f);

     k=k+1;

     x_bar = sum ( x(1:n_dim,:) ) / n_dim;

     x_r   = ( 1 + rho ) * x_bar - rho * x(n_dim+1,:);

     f_r   = feval(function_handle,x_r);
```

```
n_feval = n_feval + 1;
if ( f(1) <= f_r && f_r <= f(n_dim) )
     x(n_dim+1,:) = x_r;
     f(n_dim+1  ) = f_r;
elseif ( f_r < f(1) )
     x_e=(1+rho*delta)*x_bar-rho*delta* x(n_dim+1,:);
     f_e = feval(function_handle,x_e);
     n_feval = n_feval+1;
     if ( f_e < f_r )
         x(n_dim+1,:) = x_e;
         f(n_dim+1  ) = f_e;
     else
         x(n_dim+1,:) = x_r;
         f(n_dim+1) = f_r;
     end
elseif ( f(n_dim) <= f_r && f_r < f(n_dim+1) )
     x_c = (1+rho*gam)*x_bar - rho*gam*x(n_dim+1,:);
     f_c = feval(function_handle,x_c);
     n_feval = n_feval+1;
     if (f_c <= f_r)
         x(n_dim+1,:) = x_c;
         f(n_dim+1  ) = f_c;
     else
```

```
            [x,f] = shrink(x,function_handle,sig);
            n_feval = n_feval+n_dim;
        end
    else
        x_c = ( 1 - gam ) * x_bar + gam * x(n_dim+1,:);
        f_c = feval(function_handle,x_c);
        n_feval = n_feval+1;
        if (f_c < f(n_dim+1))
            x(n_dim+1,:) = x_c;
            f(n_dim+1  ) = f_c;
        else
            [x,f] = shrink(x,function_handle,sig);
            n_feval = n_feval+n_dim;
        end
    end
    [ f, index ] = sort ( f );
    x = x(index,:);
    converged = (f(n_dim+1)-f(1) < tolerance);
    diverged = ( max_feval < n_feval );
end
x_opt = x(1,:);
for(i=1:n_dim)
    y(i)=sum(x(:,i))/temp;
```

```
  end
  display(y);
  display(n_feval);
  display(k);
  display(f(n_dim+1));
  return
end
function f = evaluate ( x, function_handle )
  [ temp, n_dim ] = size ( x );
  f = zeros ( 1, n_dim+1 );
  for i = 1 : n_dim + 1
     f(i) = feval(function_handle,x(i,:));
  end
  return
end
function [ x, f ] = shrink ( x, function_handle, sig )
  [temp,n_dim] = size(x);
  x1 = x(1,:);
  f(1) = feval(function_handle,x1);
  for i=2:n_dim+1
     x(i,:) = sig*x(i,:) + (1-sig)*x(1,:);
     f(i) = feval(function_handle,x(i,:));
  end
```

```
  return

end
```

5.2. Matlab code for golden section method

```
function   [xminEstimate,fminEstimate,f_calculations]   =...
goldenSectionSearch(functionToMinimise,x,tol)
gamma =0.6180 ;
k = 1;
n=length(x(:,1));
a=x;
p=zeros(n,1);
q=zeros(n,1);
m=zeros(n,1);
if(n == 1)
j=1;
else
j=2;
end
for i=1:n
if(j==1)
    p(i) = a(i,j) - gamma*(a(i,j)-a(i,1));

    q(i) = a(i,1) + gamma*(a(i,j)-a(i,1));

    m(i)=a(i,j)-a(i,1);

else

   p(i) = a(i,j) - gamma*(a(i,j)-a(i,j-1));

   q(i) = a(i,j-1) + gamma*(a(i,j)-a(i,j-1));

   m(i)=a(i,j)-a(i,j-1);

end

end

fp = feval(functionToMinimise,p);
```

```
fq = feval(functionToMinimise,q);
f_calculations = 2;
while ( m >= tol )
    display(a);
    x=[p,q];
    display(x);
    y=[fp,fq];
    display(y);
    if (fp <= fq)
        for i=1:n
            a(i,j)=q(i);
        end
       q = p;
       fq = fp;
       for i=1:n
           if(j==1)
             p(i) = a(i,j) - gamma*(a(i,j)-a(i,1));
             m(i)=a(i,j)-a(i,1);
           else
             p(i) = a(i,j) - gamma*(a(i,j)-a(i,j-1));
             m(i)=a(i,j)-a(i,j-1);
           end
       end
```

```
        fp = feval(functionToMinimise,p);
    else
        for i=1:n
            a(i,j-1)= p(i);
        end
        p = q;
        fp = fq;
        for i=1:n
            q(i) = a(i,j-1) + gamma*(a(i,j)-a(i,j-1));
            m(i)=a(i,j)-a(i,j-1);
        end
        fq =feval(functionToMinimise,q);
    end
    k = k + 1;
    f_calculations = f_calculations + 1;
end
xminEstimate=zeros(n,1);
for i=1:n
    if(j==1)
        xminEstimate(i) = a(i,j);
    else
        xminEstimate(i) = (a(i,j)+a(i,j-1))/2;
    end
```

```
end

fminEstimate = feval(functionToMinimise,xminEstimate);

sprintf('  The  golden  section  search  algorithm  used  %d
iterations. \n',k)

sprintf('fminEstimate %f',fminEstimate)

sprintf('f_calculations %f',f_calculations)
```

5.3. Matlab code for thrust region method by using powell method

```
%powell
clc;
clear;
e=1e-5;
n=2;
d=eye(n);
p=zeros(n,1);
x(1,:)=p;
m=0;
k=1;
ap=0;
f(1)=powellfun(x(1,:));
r=f(1);
for p=1:n
    for q=1:n
        for i=-5:0.001:5
            x((k+1),:)=x(k,:)+i*d(k,:);
            b=powellfun(x((k+1),:));
            if b<r
                f(k+1)=b;
                r=b;
                a(k)=i;
            end
        end
        x((k+1),:)=x(k,:)+a(k)*d(k,:);
        if k==n
            if abs(x(k+1,:)-x(1,:))<=e
                    lastx=x(k+1,:);
                    lastf=powellfun(lastx);
                    hold
            else
                  for i=1:k
                      b(i)=f(i)-f(i+1);
```

```
end
MIN=b(1);
MAX=b(1);
for i=2:k
    if MIN>b(i)
        MIN=b(i);
    end
    if MAX<b(i)
        MAX=b(i);
    end
end
for i=1:k
    if MAX==b(i)
        m=i-1;
    end
end
fl=powellfun((2*x(k+1,:)-x(1,:)));
if (fl>=f(1))
    f(1)=f(k+1);
    x(1,:)=x(k,:);
    k=1;
else
    for i=1:m
        d(i,:)=d(i,:);
    end
    for i=(m+1):n-1
        d(i,:)=d((i+1),:);
    end
    d(k,:)=(x(k+1,:)-x(1,:));
    fp=f(k+1);
    for i=-5:0.001:5
        xp=x(k+1,:)+i*d(k,:);
        s=powellfun(xp);
        if s<fp
```

```
                                fp=s;
                                s=fp;
                                ap=i;
                            end
                        end
                            x(1,:)=x(k+1,:)+ap*d(k,:);
                            f(1)=powellfun(x(1,:));
                            k=1;
                    end
                end
                display(x((k+1),:));
                display(f(1));
            else
               k=k+1;
            end
        end
    end
lastx=x(1,:)
lastf=f(1)

 ...
```

6. References

1. A.R .Conn and Luis N. Vicente, *Introduction to derivative free optimization,* SIAM J. Optim., (2009).
2. A. R. Conn, K. Scheinberg, P.L. Toint, *On the convergence of derivative-free methods for unconstrained optimization,* in "Approximation Theory and Optimization: Tribute to M.J.D. Powell", A. Iserles, M. Buhmann, eds, Cambridge University Press, (1997),83-108
3. J. A. Nelder and R. Mead, *A simplex method for function minimization*, Comput.J., 7 (1965), pp. 308–313.
4. _____, *On trust region methods for unconstrained minimization without derivatives*, Math. Program, 97 (2003), pp. 605–623.
5. A. R. Conn, N. I. M. Gould, and PH. L. Toint, *Trust-Region Methods*, MPSSIAM Series on Optimization, SIAM, Philadelphia, 2000.
6. C. T. Kelley, *Detection and remediation of stagnation in the Nelder–Mead algorithm using a sufficient decrease condition*, SIAM J. Optim., 10 (1999), pp. 43–55.
7. J. C. Lagarias, J. A. Reeds,M. H. Wright, and P. E. Wright, *Convergence properties of the Nelder–Mead simplex method in low dimensions*, SIAM J. Optim.,9 (1998), pp. 112–147.
8. K. I. M. Mckinnon, *Convergence of the Nelder–Mead simplex method to a non stationary point*, SIAM J. Optim., 9 (1998), pp. 148–158.
9. X. Wanf, *Derivative-free optimization algorithms*, report, Department of Computing and Software, McMaster University, 2003.
10. P. Gilmore and C.T. Kelley, *An implicit filtering algorithm for optimization of functions with many local minima*. SIAM Journal on Optimization 5 (1995), 269–285.
11. V. Torczon, *On the convergence of multidirectional search algorithms*, SIAM Journal on Optimization 1 (1991), 123–145.
12. D. Winfield, *Function and Functional Optimization by Interpolation in data tables*, Ph.D. Thesis, Harvard University, Cambridge, MA, 1969.
13. T.G. Kolda, R.M. Lewis and V.J. Torczon*, Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Review 45 (2003), 385–482.
14. W. Huyer and A. Neumaier. SNOBFIT – Stable Noisy Optimization by Branch and Fit. ACM Transactions on Mathematical Software 35 (2008), Article 9.

15. M.J.D. Powell, *Developments of NEWUOA for minimization without derivatives*. IMA Journal of Numerical Analysis 28 (2008), 649–664.
16. J. Sacks, W.J. Welch, T.J. Mitchell and H.P. Wynn, *Design and analysis of computer experiments (with discussion)*, Statistical Science 4, pp. 409–435, 1989.
17. W. Huyer and A. Neumaier. *Global Optimization by Multilevel Coordinate Search*. Journal of Global Optimization 14 (1999), 331–355.
18. D.R. Jones, M. Schonlau and W.J. Welch: *Efficient Global Optimization of Expensive Black-Box Functions*. Journal of Global Optimization, 13:455–492, 1998.
19. V. Torczon and J.E. Dennis, *Direct search methods on parallel machines*, SIAM J. Optimization 1 (1991) 448-474.
20. NAG Library Chapter Contents. E05 – *Global Optimization of a Function*. NAG Library, Mark22(2009).http://www.nag.co.uk/numeric/FL/nagdoc_fl22/xhtml/E05/e05conts.xml
21. P.J.M. Van Laarhoven and E.H.L. Aarts: Simulated Annealing: *Theory and Applications*. Kluwer, Dordrecht, 1987.
22. R. Storn and K. Price, *Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*, Journal of Global Optimization 11 (1997), 341–359.
23. A. Auger and N. Hansen, *A restart CMA evolution strategy with increasing population size*. pp. 1769–1776 in: The 2005 IEEE Congress on Evolutionary Computation, vol 2(2005).
24. L.J. Eshelman and J. D. Schaffer, *Real-coded genetic algorithm and interval schemata*. pp. 187–202 in: Foundations of Genetic Algorithms Workshop (FOGA-92) (D. Whitley, ed.), Vail, CO, 1993.
25. N. Hansen, *the CMA evolution strategy: A comparing review*. pp. 75–102 in: Towards a new evolutionary computation. Advances on estimation of distribution algorithms (J.A. Lozano et al., eds.), Springer 2006.
26. A. Griewank and G. Corliss, *Automatic Differentiation Algorithms*, SIAM, Philadelphia, PA, 1991.
27. T. Csendes, L. P´al, J.O.H. Sendin, and J.R. Banga, *The GLOBAL Optimization Method Revisited*, Optimization Letters, 2 (2008), 445–454.
28. M. Dorigo and T. St¨utzle, *Ant Colony Optimization*, MIT Press 2004.

29. R. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, Proc. Sixth Int. Symp. Micro Machine and Human Science. Nagoya, Japan, 1995, 39–43.
30. S.S. Fan and E. Zahara, *A hybrid simplex search and particle swarm optimization for unconstrained optimization*. European Journal of Operational Research 181(2007),527-548.
31. F. Glover, Tabu thresholding: *Improved search by nonmonotonic trajectories*. ORSA Journal on Computing 7 (1995), 426–442.
32. N. Hansen, A. Auger, R. Ros, S. Finck, and P. Poˇsik, *Comparing Results of 31 Algorithms from the Black-Box Optimization*, Benchmarking BBOB-2009, Manuscript (2010). http://www.lri.fr/~hansen/gecco09-results-2010.pdf
33. F. Herrera, M. Lozano, and D. Molina, *Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems*. Web document (2010), http://sci2s.ugr.es/eamhco/CFP.php
34. R. Hooke and T.A. Jeeves. *"Direct Search" Solution of Numerical and Statistical Problems*, Journal of the ACM 8 (1961), 212–229.
35. W. Huyer and A. Neumaier. *Global Optimization by Multilevel Coordinate Search*. Journal of Global Optimization 14 (1999), 331–355.
36. A.R. Conn, K. Scheinberg and P.L. Toint*, A derivative free optimization method via support vector machines*, 1999.
37. S. ¨Ozcan, V. Yıldırım, L. Kaya, D. Becher, M. Hecker and G. ¨Ozcengiz, *Phanerochaete Chrysoporium Proteomo and a large-scale study of heavy metal response*, preprint, Department of Biology, METU (2005).
38. S.C. Nash and A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill, 1996.