

DERIVATIVE FREE OPTIMIZATION METHODS:  
APPLICATION IN STIRRER CONFIGURATION  
AND DATA CLUSTERING

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS  
OF  
THE MIDDLE EAST TECHNICAL UNIVERSITY



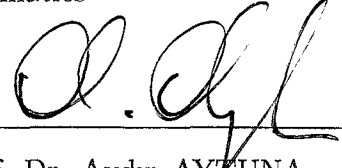
BY

BAŞAK AKTEKE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE  
IN  
THE DEPARTMENT OF SCIENTIFIC COMPUTING

JULY 2005

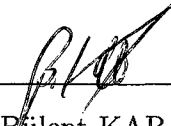
Approval of the Graduate School of Applied Mathematics



Prof. Dr. Aydın AYTUNA,

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Prof. Dr. Bülent KARASÖZEN

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

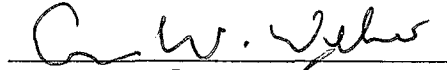


Prof. Dr. Bülent KARASÖZEN

Supervisor

Examining Committee Members

Prof. Dr. Gerhard Wilhelm Weber



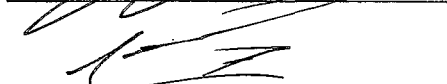
Prof. Dr. Bülent Karasözen



Assist. Prof. Yusuf Uludağ



Assoc. Prof. Dr. Tanıl Ergenç



Dr. Hakan Öktem



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Başak Akteke

Signature: 

# ABSTRACT

## DERIVATIVE FREE OPTIMIZATION METHODS: APPLICATION IN STIRRER CONFIGURATION AND DATA CLUSTERING

Başak Akteke

M.Sc., Department of Scientific Computing

Supervisor: Prof. Dr. Bülent Karasözen

July 2005, 89 pages

Recent developments show that derivative free methods are highly demanded by researches for solving optimization problems in various practical contexts. Although well-known optimization methods that employ derivative information can be very efficient, a derivative free method will be more efficient in cases where the objective function is nondifferentiable, the derivative information is not available or is not reliable. Derivative Free Optimization (DFO) is developed for solving small dimensional problems (less than 100 variables) in which the computation of an objective function is relatively expensive and the derivatives of the objective function are not available. Problems of this nature more and more 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 this thesis, we give an example of the implementation of DFO in an approach for optimizing stirrer configurations, including a parametrized grid generator, a flow solver, and DFO. A derivative free method, i.e., DFO is preferred because the gradient of the objective function with respect to the stirrer's design

variables is not directly available. This nonlinear objective function is obtained from the flow field by the flow solver. We present and interpret numerical results of this implementation. Moreover, a contribution is given to a survey and a distinction of DFO research directions, to an analysis and discussion of these. We also state a derivative free algorithm used within a clustering algorithm in combination with non-smooth optimization techniques to reveal the effectiveness of derivative free methods in computations. This algorithm is applied on some data sets from various sources of public life and medicine. We compare various methods, their practical backgrounds, and conclude with a summary and outlook. This work may serve as a preparation of possible future research.

**Keywords:** Derivative Free Optimization, Quadratic Interpolation, Trust-Region Method, Well-Poisedness, Nonlinear Optimization, Stirrer Configurations, Support Vector Machines, Non-Smooth Optimization, Inverse Problems, Statistical Learning, Data, Clustering.

# ÖZ

## TÜREVSİZ OPTİMİZASYON METOTLARI: KARIŞTIRICI KONFIGÜRASYONLARI VE VERİ SINIFLANDIRMASI UYGULAMASI

Başak Akteke

Yüksek Lisans, Bilimsel Hesaplama Bölümü

Tez Yöneticisi: Prof. Dr. Bülent Karasözen

Temmuz 2005, 89 sayfa

Son yıllardaki gelişmeler, türevsiz optimizasyon tekniklerinin, optimizasyon problemlerini çözmeye araştırmacılar tarafından oldukça fazla talep edildiğini göstermektedir. Her ne kadar, türeve dayalı yaygın optimizasyon metotları, optimizasyon problemlerini çözmeye genel olarak yeterli olsalar da, fonksiyonun türevlenemediği, türev hesaplamasının kolay olmadığı ya da güvenilir olmadığı durumlarda, türevsiz yöntemler, problemin çözümünde çok daha etkin olmaktadır. Derivative Free Optimization (DFO) adlı program fonksiyon hesaplamalarının uğraştırıcı olduğu ve fonksiyonun türevinin elde edilemediği küçük boyutlu (100 değişkenden az) problemleri çözmek için geliştirilmiştir. Bu tür problemlerle, fonksiyon değerlerinin bilgisayar benzetimleri kullanılarak hesaplandığı modern fizik, kimya, ekonometrik ölçümler ve mühendislik uygulamalarında sık sık karşılaşılmaktadır.

Bu tezde, kimyasal karıştırıcıların geometrik konfigürasyonu optimizasyonuna yönelik, parametrize edilmiş ağ üreticisini ve akışkan hesaplayıcını içeren bir DFO uygulaması örnek olarak verilmiştir. Burada türevsiz bir metot olan DFO tercih edilmiştir çünkü karıştırıcının tasarım değişkenlerine göre fonksiyonun

türevi doğrudan elde edilememektedir. Bu uygulama ile ilgili nümerik sonuçlar yorumlanarak sunulmaktadır. Bunlara ek olarak, çeşitli DFO arařtırmalarının analizi ve bunların bir deęerlendirmesi yapılarak bu konuyla ilgili çalıřmalara katkıda bulunulmuřtur.

Ayrıca, türevsiz metotların hesaplamalarda başarılı sonuçlar verdiđini göstermek amacıyla pürüzlü (non-smooth) optimizasyon teknikleri ile birlikte kullanılan, türevsiz bir sınıflandırma algoritması incelenmektedir. Bu amaçla, bu algoritma günlük yaşamda ve tıpta sık sık karşılaşılan çeşitli veri kümelerine uygulanmaktadır. Bu metotları teorik olarak karşılařtırıp, türevsiz optimizasyon alanının řu anki genel görünümünü ve özeti verilmektedir.

**Anahtar Kelimeler:** türevsiz optimizasyon, trust-region metodu, well-poisedness, doğrusal olmayan optimizasyon, karıřtırıcı konfigürasyonları, support vector machines, pürüzlü optimizasyon, ters problemler, istatistiksel öğrenme, veri, sınıflandırma.

To my family





## ACKNOWLEDGMENT

I would like to express my gratitude to all those who supported me in any means to complete this thesis. I am grateful to my supervisor Professor Dr. Bülent Karasözen for giving me the possibility to do the necessary research work and for guiding me through my graduate study in our institute.

I am deeply indebted to Professor Dr. Gerhard-Wilhelm Weber for his help, stimulating suggestions and encouragement in all the time of writing this thesis. He is a kind friend for me rather than a co-supervisor. I thank him for his careful examination of this thesis and for many valuable contributions.

I would like to thank to the members of my committee, Dr. Hakan Öktem, Assoc. Prof. Dr. Tanıl Ergenç and Assist. Prof. Yusuf Uludağ, for their helpful suggestions. Thanks to Dr. Adil Bagirov and Prof. Dr. Alexander Rubinov for sharing their studies with me which enriched this thesis a lot.

I am grateful to my colleagues and my bosses from my company for their patience and support. Many thanks to Fatma Bilge Yılmaz, Meral Sezer, Dr. Ömur Uğur, Süreyya Özögür, Aysun Tezel and Serap Yücel for their encouragement, support and valuable helps.

Special thanks to my brother Halil Akteke for his wise suggestions during the preparation of this thesis. I am grateful to him and his wife Nuran Akteke who gave me the best thesis completion present i.e., Elfin Duru. It is a great pleasure that I have now the opportunity to express my gratitude to my mother Kamile Akteke and my father Muzaffer Akteke for their limitless love and for letting me free in my choices.

Special thanks to Caner Öztürk for being with me all the way. His efforts to ensure my concentration on my studies for preparing this thesis are unforgettable.

# TABLE OF CONTENTS

ABSTRACT .....	iv
ÖZ .....	vi
ACKNOWLEDGEMENTS .....	ix
TABLE OF CONTENTS .....	x
LIST OF TABLES .....	xiv
LIST OF FIGURES .....	xv
CHAPTER	
1 INTRODUCTION .....	1
1.1 Solution Strategies .....	3
1.2 Review about the History of Derivative Free Optimization Methods .....	4
1.3 General Properties of Derivative Free Methods .....	6
1.4 Outline of the thesis .....	7
2 OVERVIEW OF UNCONSTRAINED AND CONSTRAINED DIFFERENTIABLE OPTIMIZATION .....	8

2.1	Unconstrained Differentiable Optimization . . . . .	8
2.1.1	Line-Search Methods . . . . .	9
2.1.2	Trust-Region Methods . . . . .	13
2.1.2.1	Introduction . . . . .	13
2.1.2.2	Outline and Properties of the Trust-Region Algorithm . . . . .	15
2.2	Constrained Differentiable Optimization . . . . .	19
2.2.1	Some Foundations . . . . .	19
2.2.2	Linear Programming . . . . .	22
2.2.2.1	Introduction . . . . .	22
2.2.2.2	Primal-Dual Methods . . . . .	23
2.2.3	Nonlinear Programming . . . . .	25
<b>3</b>	<b>DERIVATIVE FREE OPTIMIZATION . . . . .</b>	<b>28</b>
3.1	What is DFO? . . . . .	28
3.1.1	Trust-Region Framework . . . . .	29
3.1.2	Quadratic Interpolation . . . . .	29
3.1.3	Algorithm . . . . .	35
3.2	Derivative Free Optimization via Support Vector Machines . . . . .	38
3.2.1	Introduction into SVMs . . . . .	38
3.2.2	SVMs in Classification . . . . .	39
3.2.3	SVMs in DFO . . . . .	41
3.2.4	$\Lambda$ - Poisedness . . . . .	45
<b>4</b>	<b>OPTIMIZATION OF STIRRER CONFIGURATIONS . . . . .</b>	<b>47</b>

4.1	Optimization Problem of Stirrer Configurations . . . . .	49
4.2	Numerical Toolbox . . . . .	50
4.2.1	FASTEST 3D . . . . .	50
4.2.2	Grid Generation Tool . . . . .	51
4.2.3	DFO . . . . .	52
4.3	Illustration of the Numerical Toolbox . . . . .	52
4.4	Implementation of the Numerical Tool . . . . .	54
4.4.1	Optimization of the Power Number . . . . .	54
4.4.2	Optimization of the Newton Number . . . . .	56

## 5 DERIVATIVE FREE OPTIMIZATION

### BY NON-SMOOTH ANALYSIS AND

### VARIOUS APPLICATIONS . . . . . 60

#### 5.1 Overview of Clustering . . . . . 60

#### 5.2 Clustering via Non-Smooth Optimization . . . . . 63

##### 5.2.1 Algorithmic Framework . . . . . 63

##### 5.2.2 An Algorithm for Solving Optimization Problems in the Algorithmic Framework . . . . . 66

###### 5.2.2.1 Some Elements of Non-smooth Analysis . . . . . 66

###### 5.2.2.2 Consequence for the Clustering Problem . . . . . 67

#### 5.3 Discrete Gradient Method . . . . . 68

##### 5.3.1 Definition of Discrete Gradient . . . . . 68

##### 5.3.2 The Algorithm . . . . . 70

#### 5.4 Numerical Results . . . . . 71

5.5 Concluding Remark . . . . .	77
6 CONCLUSION . . . . .	78
APPENDIX . . . . .	80
A SOME NOTATION . . . . .	80
B A SUBROUTINE OF DFO FOR STIRRER CONFIGURATION	81
REFERENCES . . . . .	82

# LIST OF TABLES

4.1	Geometrical parameters of stirrer configuration [54]. . . . .	48
4.2	Optimization of power number [54]. . . . .	55
4.3	Optimization of Newton number for $Re=100$ . . . . .	57
5.1	Results for German towns data base (relative errors) [6]. . . . .	72
5.2	Results for the first Bavarian postal zones data set (relative errors) [6]. . . . .	72
5.3	Results for the second Bavarian postal zones data set (relative errors) [6]. . . . .	73
5.4	Results for the first Bavarian postal zones data set (relative errors, running time) [6]. . . . .	74
5.5	Results for the Fishers iris data set (relative errors, running time) [6].	74
5.6	Results for the TSPLIB ( $n=1060$ ) data set (relative errors, running time) [6]. . . . .	75
5.7	Results for the TSBLIB ( $n=3038$ ) zones data set (relative errors, running time) [6]. . . . .	76

# LIST OF FIGURES

2.1 Three possible iterations in a line-search algorithm [19]. . . . .	9
2.2 Six iterations of a trust-region algorithm [19]. . . . .	15
2.3 A confidence ellipsoid and its projections [2]. . . . .	18
3.1 Two quadrics whose intersection curve $I$ projects onto the parabola $C : x = y^2$ [25]. . . . .	32
3.2 Newton fundamental polynomials [14]. . . . .	34
3.3 SVM finds a hyperplane for separating the data set into two classes [45]. . . . .	40
3.4 Support vectors and the margin $\rho$ [45]. . . . .	40
3.5 Linear support vector machines model [19]. . . . .	43
4.1 A typical stirrer tank [72]. . . . .	47
4.2 The configuration of a stirrer tank [54]. . . . .	48
4.3 Flow chart of control script [54]. . . . .	53
4.4 Power curve for standard tank configuration [54]. . . . .	54
4.5 Newton number versus number of the loops [54]. . . . .	56
4.6 Comparison of the velocity field for vertical planes between two baf- fles at Reynolds number 1000 [54]:	
(i) optimized parameter obtained from DFO,	
(ii) standard parameters. . . . .	58
4.7 Dimensionless impeller parameter versus number of the loops [54]:	
(i) dimensionless disk thickness , (ii) clearance , (iii) baffle length. .	59

# CHAPTER 1

## INTRODUCTION

The problem of minimizing a *nonlinear* function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  of several variables when the derivatives of the function are not available is attempted to be solved by the *derivative free methods*. This function may in fact be *smooth*, but also a *non-smooth* function is possible here. A formal statement of the underlying problem can be found in [17] as follows:

$$\begin{aligned} (\mathcal{CP}) \quad & \min && f(x) \\ & \text{such that} && a_i \leq c_i(x) \leq b_i \quad (i = 1, 2, \dots, m), \\ & && x \in \mathbb{F} \subseteq \mathbb{R}^n, \end{aligned}$$

where  $\nabla f(x)$  cannot be computed or just does not exist for every  $x$ . Here,  $\mathbb{F}$  is an arbitrary subset of  $\mathbb{R}^n$ , and  $x \in \mathbb{F}$  is called the *easy* constraint while the functions  $c_i(x)$  ( $i = 1, 2, \dots, m$ ) represent *difficult* constraints [17]. By easy constraints, we mean bound constraints on the variables, linear constraints, or more general nonlinear smooth constraints whose values and the Jacobian matrix can be computed cheaply. Difficult constraints are any nonlinear constraints whose value is expensive to compute and whose derivatives are unavailable [56].

For a small repetition and preparation the following chapters likewise, we recall some basic concepts of differentiable optimization in Chapter 2 firstly.

Then, we will consider the algorithm of DFO (derivative free optimization) in the first section of Chapter 3 for the unconstrained case, i.e., problem  $(\mathcal{CP})$  with  $M = \mathbb{R}^n$ , where  $M$  denotes the feasible set of  $(\mathcal{CP})$ . The solution strategy of DFO for the constrained problems will also be explained in Chapter 3 [17]. Additionally, we present a different approach of DFO with support vector ma-



chines which can be used for solving the constraint problem ( $\mathcal{CP}$ ) in the third section of Chapter 3 [19].

Now, we introduce the *unconstrained optimization problem* formally,

$$(\mathcal{UP}) \quad \text{minimize } f(x),$$

where  $x \in \mathbb{R}^n$  and again  $\nabla f(x)$  cannot be computed for every  $x$ .

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. The main motivation for improving algorithmic solutions to these problems is the high demand from practitioners for such tools.

In [15] and [55], Conn, Scheinberg and Toint state that there are three important features characterizing problems of derivative free optimization. We present them in Chapter 3. In fact, firstly, evaluating the objective function  $f(x)$  at a given vector  $x$  is computationally very expensive. This kind of expensive evaluations cause 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 nature 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.

There are cases where the objective function is not smooth. While trust-region algorithms are more useful for various engineering applications (Chapter 4) and, in particular, data classification, non-smooth optimization is of special importance for *data clustering*. In Chapter 5, we introduce into non-smooth optimization based on the research of Bagirov, Rubinov and Yearwood [6, 7, 8].

Throughout this thesis, we give various and specific motivations and corresponding computations.

## 1.1 Solution Strategies

Several strategies can be considered when we are faced with the problems which do not allow utilization of direct derivatives of gradients [14]. The first is to apply existing *direct search* optimization methods, like the well-known and widely-used *simplex reflection algorithm* of [47] or its modern variants, or the *parallel direct search algorithm* of [63] and [64]. 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 [30]. Automatic differentiation is utilized to define computer programs which calculate the derivatives of a function by some procedures. These computer programs calculate the *Jacobian* of *vector-valued* functions which are from  $n$ -dimensional to  $m$ -dimensional Euclidean space, i.e., from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ . On the other hand, if the function is *scalar-valued*, i.e., from  $\mathbb{R}^n$  to  $\mathbb{R}$ , then the computer program should calculate the *gradient* (and *Hessian*) of the function [31]. However, such tools are not preferred in solving problems which we consider, i.e., (*CP*) or (*UP*). 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 [22]. 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* proceeding 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.

## 1.2 Review about the 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 [69, 70]. 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 function 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. In [70], 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 [51] 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 [52].

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. In Chapter 3, we give a brief discussion on implementation of the support vector machines using DFO.

The appearance of min-type, max-type or other nondifferentiable functions has given rise to *non-smooth* analysis and optimization [13]. In Chapter 4, we introduce into both modern areas of mathematical programming and present recent applications to data clustering.

With the motivations and examples in data classifications and data clustering given in Chapter 3 and 5, respectively, we also guide the interested reader in the emerging fields of supervised and unsupervised statistical learning [36], respectively, and into data mining [24].

## 1.3 General Properties of Derivative Free Methods

Derivative free methods for unconstrained optimization build a linear or quadratic model of the objective function and apply one of the fundamental approaches of continuous optimization, i.e., a trust-region or a line-search, to optimize this model [18]. While derivative based methods typically use a *Taylor*-based model which is an approximation of the objective function, derivative free methods use interpolation, regression or other *sample*-based models. If the problem has constraints, the strategy of derivative free methods is usually to apply sequential quadratic programming methods for the linearization of the constraints.

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 step 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].

## 1.4 Outline of the thesis

Firstly, we briefly present the foundations of differentiable optimization in Chapter 2. We do this because of the need to recall the fundamental approaches of numerical optimization, to introduce by this the basic ideas of trust-region algorithm and, herewith, prepare our introduction of derivative free optimization.

We focus on the derivative free optimization (DFO) developed by Conn, Scheinberg and Toint in Chapter 3. The theoretical aspects of DFO is explained in detail there. We contribute to a state-of-the-art survey and distinction of the various different new DFO research directions, to an analysis and a discussion of them. This contribution is mainly on combining support vector machines (SVM) model with DFO instead of the quadratic interpolation model.

In Chapter 4, we explain how we implement DFO, which is a FORTRAN software package in a modern approach for optimizing stirrer configurations. In this implementation, a *parametrized grid generator*, a *flow solver*, and *DFO v2.0* are integrated within an optimization tool. We present the numerical results obtained by running this tool and present a discussion on them.

In Chapter 5, we show how a derivative free algorithm is used within a clustering algorithm from [6] in combination with non-smooth optimization techniques. This algorithm is applied on various data sets from modern life and, in particular, medicine. Our contribution is to extend the usage of learning methods given by combining support vector machines with DFO, to the clustering algorithm of Bagirov and Yearwood. Herewith, we widen the view towards various statistical learning and data mining applications.

Then, this thesis ends in Chapter 6 with a conclusion and an outlook to possible future research directions.

# CHAPTER 2

## OVERVIEW OF UNCONSTRAINED AND CONSTRAINED DIFFERENTIABLE OPTIMIZATION

### 2.1 Unconstrained Differentiable Optimization

There are basically two fundamental approaches for solving smooth *unconstrained* minimization problems which we introduced at the beginning of Chapter 1 already:

$$(UP) \quad \min \quad f(x),$$

where  $x \in \mathbb{R}^n$ , and  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is a  $C^1$ -function (see Appendix A), i.e., continuously differentiable. These approaches are:

- (i) *line-search methods*, and
- (ii) *trust-region methods*.

For (ii), we shall even assume that  $f$  is a  $C^2$ -function, i.e., twice continuously differentiable. Detailed explanation which we give in this Chapter are based on the following references: [20, 38, 46] and [48].

Our main interest is in trust-region methods, since in derivative free optimization, a trust-region algorithm is used to solve the optimization problems without

derivatives. However, let us for a better understanding explain the line-search methods first.

### 2.1.1 Line-Search Methods

These methods are the oldest and most widely used methods in unconstrained optimization. The strategy in the line-search is to choose a direction  $p_k$  and to search along this direction from the current iterate  $x_k$  for a new iterate with a lower function value. In line-search methods, the *next iteration* is obtained as follows:

$$x_{k+1} := x_k + \alpha_k p_k.$$

Here,  $x_k$  is the current approximation of a solution of  $(UP)$ ,  $p_k$  is the *search direction* at the point  $x_k$ , and  $\alpha_k$  is the *step length* which is a real number (scalar) chosen so that we have descent:

$$f(x_{k+1}) < f(x_k).$$

An illustration of line-search method is given in Figure 2.1:

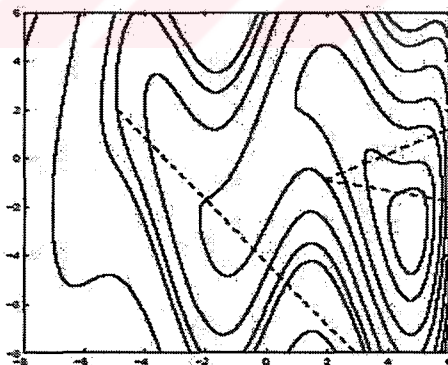


Figure 2.1: Three possible iterations in a line-search algorithm [19].

The previous inequality with respect to the objective function implies a decrease in the function value at every step towards the minimum. To have a decrease



in the function value,  $p_k$  needs to be a *descent direction* at  $x_k$ . That is,  $p_k$  must satisfy

$$p_k^T \nabla f(x_k) < 0,$$

where  $p_k^T \nabla f(x_k)$  is just the directional derivative of  $f(x_k + \alpha p_k)$  at  $\alpha = 0$ . At each iteration,  $p_k^T \nabla f(x_k) < 0$  should be satisfied when we compute the search direction.

If  $p_k$  is a descent direction, then for small  $\alpha$ , i.e.,  $\alpha > 0$  and  $\alpha \approx 0$ , we obtain:

$$f(x_k + \alpha p_k) < f(x_k).$$

From now on and without loss of generality, we assume that  $(\alpha = \alpha_k > 0)$ .

The name *line-search* bases on the fact that every iteration in *line-search methods* searches for the new point  $x_{k+1}$  along the line  $y(\alpha) := x_k + \alpha p_k$ . The ideal choice of  $\alpha_k$  would be the solution of the following auxiliary problem, where

$$(\mathcal{LS}) \quad \begin{array}{ll} \min & f(x) := f(x_k + \alpha p_k) \\ \text{such that} & \alpha > 0. \end{array}$$

Usually, finding exactly the minimum value of this one-dimensional minimization problem is an expensive task, since it requires too many function evaluations. Instead, an approximate minimum will be acceptable if the iteration results with a sufficient decrease in the function value. However, a reduction in the function value on its own does not guarantee the convergence to the solution of  $(\mathcal{UP})$ .

One needs additional assumptions to guarantee the sufficient reduction in the function value at each iteration and, hence, to guarantee the convergence to the smallest value of  $f$ . These assumptions are as follows, where (a) and (b) are on the search direction  $p_k$ , and (c) and (d) are on the step length  $\alpha_k$  [46]:

- (a)  $p_k$  allows “sufficiently descent”,
- (b)  $p_k$  is “gradient related”,

(c)  $\alpha_k$  allows “sufficient decrease”, and

(d)  $\alpha_k$  is “not too small”.

**On (a):** Even if we require that the inequality  $p_k^T \nabla f(x_k) < 0$  should be satisfied, there is the possibility of  $p_k$  and  $\nabla f(x_k)$  becoming almost orthogonal. This would cause little progress towards a solution in the iteration procedure. We can avoid this possibility by assuming

$$-\frac{p_k^T \nabla f(x_k)}{\|p_k\|_2 \cdot \|\nabla f(x_k)\|_2} \geq \varepsilon \quad (k \in \mathbb{N}_0),$$

where  $\varepsilon > 0$  is some specified tolerance. This is an angular condition, because by our knowledge from *linear algebra* we can write it in the form

$$\cos(\theta_k) \geq \varepsilon \quad (k \in \mathbb{N}_0),$$

where  $\theta_k$  is the angle between  $p_k$  and  $-\nabla f(x_k)$  [38].

**On (b):** We say that  $p_k$  is *gradient related* if

$$\|p_k\| \geq \lambda \|\nabla f(x_k)\|_2 \quad (k \in \mathbb{N}_0),$$

where  $\lambda > 0$  is some constant. This ensures that the norm of  $p_k$  cannot go too much below the norm of  $\nabla f(x_k)$ .

It is easy to make these two conditions (a) and (b) becoming fulfilled, by slight modifications in the method used for calculating the search direction.

**On (c):** In every line-search iteration step, the step length should achieve a nontrivial reduction of  $f$ . Here, “nontrivial” can be defined in terms of a linear Taylor series approximation. Based on

$$f(x_k + \alpha p_k) \approx f(x_k) + \alpha p_k^T \nabla f(x_k),$$

we require so-called *Armijo-condition* [46]:

$$(AC) \quad f(x_k + \alpha_k p_k) \leq f(x_k) + \mu \alpha_k p_k^T \nabla f(x_k) \quad (k \in \mathbb{N}_0),$$

where  $\mu \in (0, 1]$ . If  $\mu \approx 0$ , then a small decrease in the function value is requested; hence, Armijo-condition is relatively easy to satisfy. If  $\mu \approx 1$ , then a big decrease in  $f$  is requested, so that the condition is hard to satisfy. If  $\alpha_k \approx 0$ , then  $f(x_k + \alpha p_k) \approx f(x_k)$ , which means that the linear approximation is good and a nontrivial reduction will be achieved. If  $\alpha_k \gg 0$ , then the decrease predicted by the linear approximation may strongly differ from the real decrease of  $f$ , and (c) can be violated. Therefore, (c) prevents  $\alpha_k$  from becoming “too big”.

**On (d):** To prevent  $\alpha_k$  from becoming “too small”:

- A simple line-search with backtracking can be used allowing a convergence result (not recommended for practice).
- **Wolfe condition** [46]:

$$\|p_k^T \nabla f(x_k + \alpha_k p_k)\| \leq \xi \|p_k^T \nabla f(x_k)\| \quad \forall k \in \mathbb{N}_0,$$

where  $\xi \in [0, 1)$ .

Using Wolfe condition to control  $\alpha_k$  is usually a complicated task. A discussion on this fourth condition can be found in [46].

**Theorem 2.1.** *For our minimization problem (UP), let  $x_0$  be a given initial point and define  $(x_k)_{k \in \mathbb{N}_0}$  by  $x_{k+1} = x_k + \alpha_k p_k$ , where  $\alpha_k \geq 0$  is a scalar, and  $p_k$  is an  $n$ -dimensional vector ( $k \in \mathbb{N}_0$ ). We assume:*

- The lower level set  $\mathcal{L}^{f(x_0)} := \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$  is bounded (hence, by the continuity of  $f$ , compact).*
- The gradient  $\nabla f$  is Lipschitz continuous:*

$$\exists L > 0 : \quad \|\nabla f(x) - \nabla f(y)\|_2 \leq L \cdot \|x - y\|_2 \quad \forall x, y \in \mathbb{R}^n,$$

where  $\|\cdot\|$  denotes the Euclidean norm.

(c) Condition (c) imposed on  $\alpha_k$  ( $k \in \mathbb{N}_0$ ) (sufficient decrease).

(d) Condition (b) imposed on  $\alpha_k$  ( $k \in \mathbb{N}_0$ ) (gradient relation) and (boundedness)

$$\exists M \geq 0 : \quad \|p_k\| \leq M \quad \forall k \in \mathbb{N}_0.$$

(e) Condition (c) imposed on  $\alpha_k$ , being chosen as the first element of the sequence  $(\frac{1}{2^j})_{j \in \mathbb{N}_0}$  satisfying this (Armijo condition) ( $k \in \mathbb{N}_0$ ; simple line-search).

Then, it is fulfilled:

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

**Proof.** See the book of Nash and Sofer, pp. 317 [46]. □

This theorem only implies the convergence of the gradients, it does *not* say that  $(x_k)_{k \in \mathbb{N}_0}$  tends to a local minimizer. One must impose stronger assumptions on  $(\mathcal{UP})$  in order to obtain a stronger theorem which implies convergence in the space of states, towards a local solution  $x^*$ .

Now, let us come to the second basic kind of approach.

## 2.1.2 Trust-Region Methods

### 2.1.2.1 Introduction

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*  $m_k(p)$  is computed in a neighborhood of the current iterate  $x_k$ , which we refer to as the *trust-region*. The model  $m_k(p)$  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 subproblem to obtain the *next iteration* at each step  $k$  of a trust-region method,

$$(\mathcal{QP})_{t.r.}^k \quad \begin{cases} \min_p & m_k(p) := f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \\ \text{subject to} & \|p\| \leq \Delta_k, \end{cases}$$

where  $\Delta_k > 0$  is the trust-region radius and  $\|\cdot\|$  is defined to be the Euclidean norm. These subproblems 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  $-p^T p + \Delta_k^2 \geq 0$ . In fact, usually, the model  $m_k(p)$  is a quadratic function which is truncated from a Taylor series for  $f$  around the point  $x_k$ :

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \quad (k \in \mathbb{N}_0).$$

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

Hence, our *trust-region* for the model  $m_k(p)$  is a bounded neighborhood of the current iterate  $x_k$  [48]:

$$\mathcal{B}(x_k) := \{x_k + p \mid \|p\|_2 \leq \Delta_k\}.$$

After constructing the model  $m_k(p)$  and its trust-region, then, one seeks a *trial step*  $p$  to the next iteration  $x_{k+1} = x_k + p$  which will result in a reduction for the model while the size of the step is bounded by  $\mathcal{B}(x_k)$ , i.e.,  $\|p\|_2 \leq \Delta_k$ .

Then, the objective function is evaluated at  $x_k + p$  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,  $x_k + p$  is accepted as

the next iterate and the trust-region is possibly expanded to include this new point (i.e.,  $\Delta_k$  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.,  $\Delta_k$  decreases), with the hope that the model provides a better prediction in the smaller region [20]. The illustration of the *trust-region* steps can be seen in Figure 2.2 (cf. [19]).

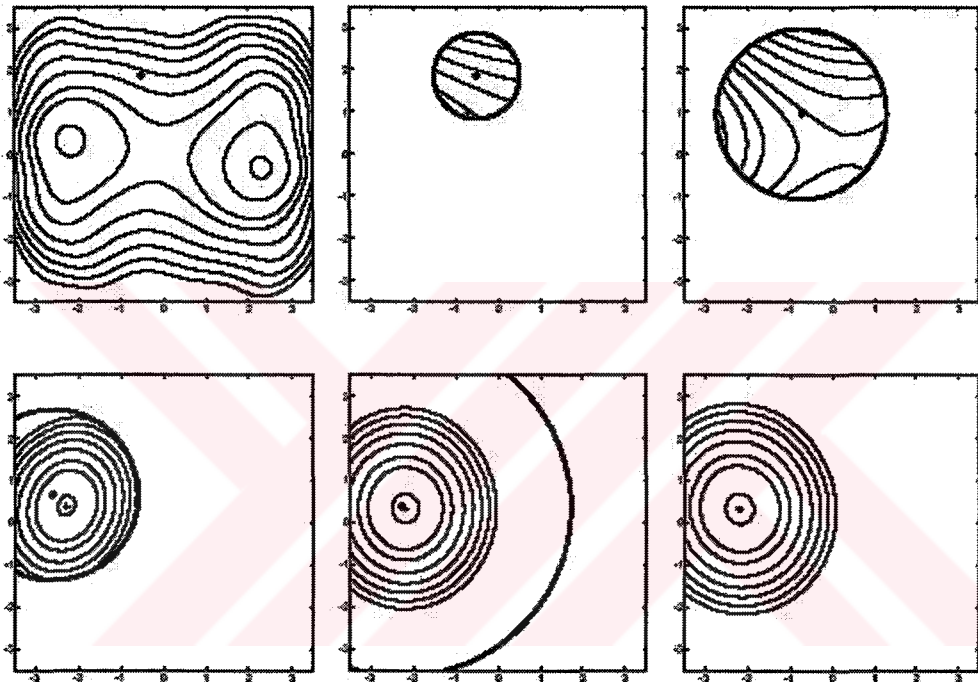


Figure 2.2: Six iterations of a trust-region algorithm [19].

### 2.1.2.2 Outline and Properties of the Trust-Region Algorithm

In a trust-region algorithm, a strategy for determining the trust-region radius  $\Delta_k$  at each iteration is needed to be developed [48]. 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  $p_k$ , we define the ratio

$$\rho_k := \frac{f(x_k) - f(x_k + p_k)}{m_k(x_k) - m_k(x_k + p_k)} = \frac{\text{actual reduction}}{\text{predicted reduction}}. \quad (2.1.1)$$

There are various definitions for  $\rho_k$  in the mathematics literature, but we shall prefer (2.1.1) in particular here. We note that the denominator of  $\rho_k$ , namely, the predicted reduction, is always nonnegative since the step  $p_k$  is computed from the subproblem  $(\mathcal{QP})_{t.r.}^k$  over a region that includes the step  $p = 0$ .

In fact,  $\rho_k$  can be called a measure of how good the model  $m_k(p)$  predicts the reduction in the  $f$ -value. If  $\rho_k$  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  $\Delta_k$ . Thus,  $p_k$  will be rejected and  $\Delta_k$  will be reduced. On the other hand, if  $\rho_k$  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.,  $\Delta_k$  should be increased. If  $\rho_k$  is *positive* but *not* close to 1, then the trust-region radius is not changed. This process is described in the following algorithm.

**Algorithm 2.2.** Choose some  $x_0$  as an initial guess,  $\bar{\Delta}$  as the overall bound on the step lengths,  $\Delta_0 \in (0, \bar{\Delta})$  as an initial trust-region radius and the constant  $\eta \in [0, \frac{1}{4})$ :

**for**  $k = 0, 1, \dots$

Solve  $(\mathcal{QP})_{t.r.}^k$  to obtain  $p_k$ .

Evaluate  $\rho_k$  from (2.1.1):

**if**  $\rho_k \leq \frac{1}{4}$

$$\Delta_{k+1} = \frac{1}{4} \|p_k\|,$$

**else**

**if**  $\rho_k > \frac{3}{4}$  and  $\|p_k\| = \Delta_k$

$$\Delta_{k+1} = \min\{2\Delta_k, \bar{\Delta}\},$$

**else**

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

**if**  $\rho_k \geq \eta$

$$x_{k+1} = x_k + p_k,$$

**else**

$$x_{k+1} = x_k,$$

**end(for)**

**Remark 2.3.** We note that the idea of *trust*-regions is similar to the one about *confidence intervals* that is dealt with in *statistical analysis* and *statistical learning*. There, the limits containing the value of a random variable with a probability of, let us say, 95% are called the 95% *confidence limits* for the parameter. In the univariate case of one variable, the interval between confidence limits is called the *confidence interval*. A pair of 95% confidence limits around an expected or estimated value of the considered parameter means an interval where it has probability of 95% that your variable's value in a sampling set is just located here. For the shape, the convention is to use a line segment centered on the measured value in one dimension, whereas confidence *ellipses* or *ellipsoids* are most frequently used in higher dimensions (see Figure 2.3). The latter sets by their shape or *eccentricity* also imply the degree of relatedness between the random variable's components [2].  $\square$

Let us state the following result which can be compared with Theorem 2.1:

**Theorem 2.4.** *For our minimization problem (UP), let  $x_0$  be a given initial point and  $(x_k)_{k \in \mathbb{N}_0}$  be defined by the trust-region algorithm as stated above. We assume:*

- (a) *The lower level set  $\mathcal{L}^f(x_0) := \{x \in \mathbb{R} \mid f(x) \leq f(x_0)\}$  is bounded (hence, by (b), compact).*
- (b)  *$f \in C^2$ , i.e.,  $f$ ,  $\nabla f$  and  $\nabla^2 f$  are continuous in  $\mathcal{L}^f(x_0)$ .*

*Then, it is fulfilled:*

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



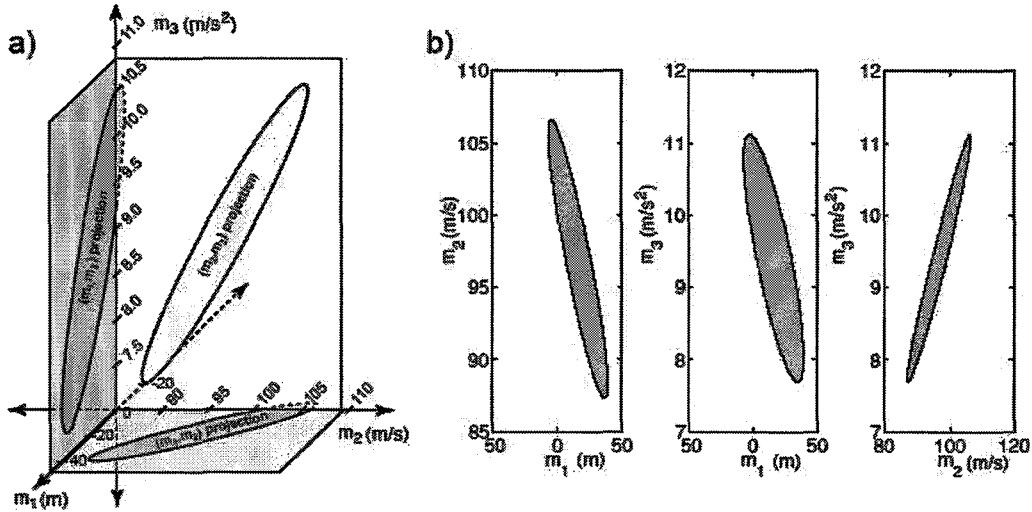


Figure 2.3: A confidence ellipsoid and its projections [2].

**Proof.** See the book of Nash and Sofer [46]. □

Like the one we stated in the previous section, this theorem implies convergence of the gradients to zero, however, it does *not* say that  $(x_k)_{k \in \mathbb{N}_0}$  tends to a local minimizer. To obtain a stronger theorem which states convergence towards a local solution  $x^*$ , additional assumptions must be imposed on  $(UP)$ .

Let us briefly summarize the approaches which the line-search and the trust-region methods use for finding the solution of  $(UP)$ . 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  $\alpha$ . Herewith, line-search finds an approximate value of  $\alpha$  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 direction  $p$  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 and the trust-region 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  $\alpha_k$  is identified. In *trust-region algorithms*, however, a region around the current iterate is chosen with a maximum radius  $\Delta_k$ , then a direction is being looked for to obtain the sufficient decrease in  $f$ .

## 2.2 Constrained Differentiable Optimization

### 2.2.1 Some Foundations

In this section, we explain the minimization problems with constraints. Generally, the constraint optimization problems are formulated as follows:

$$\begin{aligned}
 (\mathcal{CP}) \quad & \text{minimize} && f(x) \\
 & \text{subject to} && h_i(x) = 0 \quad (i \in I := \{1, 2, \dots, m\}), \\
 & && g_j(x) \geq 0 \quad (j \in J := \{1, 2, \dots, s\}).
 \end{aligned}$$

The functions  $f$ ,  $h_i$  and  $g_j$  in  $(\mathcal{CP})$  are supposed to be smooth, real-valued on  $\mathbb{R}^n$ . Here, the meaning of *smooth* is being one or two times continuously differentiable depending on our analytical necessity. As before,  $f$  is the objective function, while the constraints  $h_i(x) = 0$  and  $g_j(x) \geq 0$  are called *equality constraints* and *inequality constraints*, respectively.

The set of  $x$ 's which satisfy the constraints is the *feasible set*, introduced as follows:

$$M := \{x \in \mathbb{R}^n \mid h_i(x) = 0 \ (i \in I), \ g_j(x) \geq 0 \ (j \in J)\}.$$

Therefore, we can rewrite the constraint optimization problem  $(\mathcal{CP})$  as

$$(\mathcal{CP}) \quad \text{minimize} \quad f(x) \quad \text{subject to} \quad x \in M.$$

We need to define some important concepts which will be used in the following chapters of this thesis. If the reader is well-familiar with them, then he/she can skip them and directly go to a latter subsection or to Chapter 3.

**Definition 2.5.** Let a vector (or, point)  $x^* \in \mathbb{R}^n$  be given.

- (i) We call  $x^*$  a *local solution* of  $(\mathcal{CP})$ , if  $x^* \in M$  and there is a neighborhood  $\mathcal{N}(x^*) \subseteq \mathbb{R}^n$  of  $x^*$  such that

$$f(x^*) \leq f(x) \quad \forall x \in M \cap \mathcal{N}(x^*).$$

- (ii) We call  $x^*$  a *strict local solution* of  $(\mathcal{CP})$ , if  $x^* \in M$  and there is a neighborhood  $\mathcal{N}(x^*) \subseteq \mathbb{R}^n$  of  $x^*$  such that

$$f(x^*) < f(x) \quad \forall x \in (M \cap \mathcal{N}(x^*)) \setminus \{x^*\}.$$

- (iii) We call  $x^*$  an *isolated local solution* of  $(\mathcal{CP})$ , if  $x^* \in M$  and there is a neighborhood  $\mathcal{N}(x^*) \subseteq \mathbb{R}^n$  of  $x^*$  such that  $x^*$  is the only local solution of  $(\mathcal{CP})$  in  $M \cap \mathcal{N}(x^*)$ . □

We continue with a numerical example on constraint optimization problems to increase the understanding:

**Example 2.6.** [38, 48] Let the following minimization problem be given:

$$(\mathcal{CP}) \quad \begin{array}{ll} \text{minimize} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 - 2 = 0. \end{array}$$

Here, we see that there is one equality constraint, whereas there is no inequality constraint, i.e.,  $I = \{1\}$  and  $J = \emptyset$ . One can see that the feasible set  $M$  is the circle centered at the origin with radius  $\sqrt{2}$ . The solution  $x^*$  is obviously  $(-1, -1)^T$ , this is the global solution of  $(\mathcal{CP})$ : From any other point on the circle, it is easy to find a way to move which stays feasible (i.e., remains on the circle) while decreasing  $f$ .

By careful inspection, we see that the gradient of the equality constraint  $\nabla h(x)$  of  $(\mathcal{CP})$  is parallel to the gradient of the function  $\nabla f(x)$  at  $x^* = (-1, -1)^T$ :

$$\nabla f(x^*) = \lambda^* \nabla h(x^*),$$

where  $\lambda^* = \lambda_1^* = -\frac{1}{2}$ . □

In fact, this condition which is necessary when we are looking for a (local) minimizer, can be expressed as follows:

$$\nabla_x L(x^*, \lambda^*) = 0,$$

where  $L : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$  is called the *Lagrange function* and defined as  $L(x, \lambda) := f(x) - \lambda h(x)$ . The value  $\lambda^*$  is called the *Lagrange multiplier* at  $x^*$  with respect to the equality constraints  $h(x)$ .

One can also derive other Lagrange multipliers  $\mu$  by modifying Example 2.6 with respect to the *inequality* constraints  $g(x)$ . Then, the Lagrange function becomes  $L(x, \mu) := f(x) - \mu g(x)$  and the necessary condition for (local) minimizer becomes

$$\nabla_x L(x^*, \mu^*) = 0,$$

where  $\mu^* \geq 0$ .

The set of indices  $j$  for which the inequality constraints are equal to zero is defined as

$$J_0(x) := \{j \in J \mid g_j(x) = 0\}.$$

This set is called the *active set* and the corresponding inequality constraints are called the *active inequality constraints*. According to this, we define *Karush-Kuhn-Tucker* conditions which are our famous first-order necessary conditions for  $(\mathcal{CP})$ :

There exist vectors  $\lambda^* = (\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*)^T$  and  $\mu^* = (\mu_j^*)_{j \in J_0(x^*)}^T$ ,

$$\begin{aligned}
(\mathcal{KKT}) \quad & \nabla f(x^*) = \sum_{i=1}^m \lambda_i^* \nabla h_i(x^*) + \sum_{j \in J_0(x^*)} \mu_j^* \nabla g_j(x^*), \\
& \mu_j^* \geq 0 \quad (j \in J_0(x^*)).
\end{aligned}$$

The first condition of  $(\mathcal{KKT})$  can also be written as

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0,$$

where  $L$  is the corresponding Lagrange function (with some artificially augmented vector  $\mu^*$ ). Here, we refer to [38].

We will need these *Karush-Kuhn-Tucker* conditions later on when we explain other approaches to DFO with support vector machines in Chapter 3.

In [38, 46, 48], basic solution methods to the constrained optimization problem  $(\mathcal{CP})$  can be found in greater detail.

## 2.2.2 Linear Programming

### 2.2.2.1 Introduction

This subsection briefly introduces into linear programming which is one of the most important and commonly appearing problem and method classes in constrained optimization [37, 48]. By paying attention to  $(\mathcal{LP})$ , we also introduce a bit into the methods of interior (and exterior) points to which we will come in the context of support vector machines (cf. Section 3.2).

**Definition 2.7.** A *linear programming problem in standard form* is

$$(\mathcal{LP}) \quad \begin{cases} \text{minimize} & c^T x \\ \text{subject to} & Ax = b, \\ & x \geq 0. \end{cases}$$

Here  $A$  is an  $m \times n$  matrix,  $m \leq n$ ,  $c$  is an  $n$ -vector and  $b$  is an  $m$ -vector. Furthermore,  $c^T x$  is the *objective function*,  $Ax = b$  and  $x \geq 0$  are the *con-*

*straints* of the  $(\mathcal{LP})$ . In this kind of problems, the objective function and all the constraints are *linear* functions.  $\square$

The *feasible set*  $M$  for such a problem which is the set of all the points satisfying the constraints, is a convex polyhedron in  $n$ -dimensional space. Very often, the *simplex method* is preferred to solve the linear programming problems  $(\mathcal{LP})$  [48]. The minimum of the problem  $(\mathcal{LP})$  if existing, is known to be attained at one of the vertices of the polyhedron “simplex”  $M$ . The simplex method, however, is not in the special scope of this thesis. We are interested in the *interior-point methods* which are also used for solving linear programming problems in recent years. These interior-point methods seek the minimum attained at some vertex of the feasible set and by coming from the interior of it.

The features of an interior-point method that come into prominence can be counted as follows by comparing it with the simplex method [48]: The iterations in the algorithm of interior-point methods are computationally expensive but make a significant progress towards the solution, in contrast to the simplex method which requires a large number of inexpensive iterations. The vertices on the boundary of the feasible region are tested sequentially by the simplex method to find the minimum value of  $(\mathcal{LP})$ . However, *interior (and exterior) point methods* approach the solution of  $(\mathcal{LP})$  either from the interior (or from the exterior) of the feasible region. In fact, their iterates generally do not lie on the boundary of this region but approach the boundary of the feasible set in the limit.

### 2.2.2.2 Primal-Dual Methods

*Primal-dual methods* are known as one of the most efficient *interior-point* approaches. They solve the linear problems by formulating them as nonlinear ones [38, 48]. This reformulation of the problem into a nonlinear form forces the solution procedures to continue with nonlinear algorithms like *Newton’s method*.

**Definition 2.8.** Let the standard linear programming problem  $(\mathcal{LP})$  be given

as above. The *dual linear problem* with respect to  $(\mathcal{LP})$  is

$$(\mathcal{DP}) \quad \begin{cases} \text{maximize} & b^T y \\ \text{subject to} & A^T y + s = c, \\ & s \geq 0, \end{cases}$$

where  $y \in \mathbb{R}^m$  (*dual variable*) and  $s \in \mathbb{R}^n$  (*slack variable*). □

*Primal-dual* solutions of  $(\mathcal{LP})$  and  $(\mathcal{DP})$  fulfill the *Karush-Kuhn-Tucker conditions* which we already defined above. In our context here, the restatement of these conditions consists of the following system of *first-order necessary optimality conditions*:

$$(\mathcal{KKT}) \quad \begin{cases} A^T y + s = c, \\ Ax = b, \\ x_j s_j = 0 & (j = 1, 2, \dots, n), \\ (x, s) \geq 0. \end{cases}$$

where  $(x, s) \geq 0$  represents the nonnegativity of all  $x_j$  and  $s_j$ .

We note that the corresponding *Lagrange function* of the problem looks as follows (cf. Subsection 2.2.1):

$$L(x, y, s) := c^T x - y^T (Ax - b) - s^T x.$$

The solutions  $(x^*, y^*, s^*)$  of  $(\mathcal{KKT})$  are obtained by a modified Newton's method applied on the first three equations in  $(\mathcal{KKT})$ . This modification in Newton's method is related with the *search directions* and the *step lengths* to guarantee that the inequalities  $(x, s) \geq 0$  hold at every iteration of *primal-dual* solution procedure.

By a restatement of  $(\mathcal{KKT})$ , we derive primal-dual interior point methods

$$(\mathcal{KKT}) \quad \left\{ \begin{array}{l} \mathbb{F}(x, y, s) := \begin{pmatrix} A^T y - s - c, \\ Ax - b, \\ X S \mathbf{e} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \\ (x, s) \geq 0, \end{array} \right.$$

where  $F : \mathbb{R}^{2n+m} \rightarrow \mathbb{R}^{2n+m}$ ,  $X := \text{diag}(x_1, x_2, \dots, x_n)$ ,  $S := \text{diag}(s_1, s_2, \dots, s_n)$ , and  $\mathbf{e} := (1, 1, \dots, 1)^T$ .

We parametrize  $(\mathcal{KKT})$  by a scalar  $\tau > 0$  and obtain

$$(\mathcal{KKT})_\tau \quad \left\{ \begin{array}{l} A^T y + s = c, \\ Ax = b, \\ x_j s_j = \tau \quad (j = 1, 2, \dots, n), \\ (x, s) \geq 0. \end{array} \right.$$

By using the notation given by the function  $F$  and asking for the unknown  $(x_\tau, y_\tau, z_\tau)$  and noting the necessary positivity, we get the representation of  $(\mathcal{KKT})_\tau$  as

$$(\mathcal{KKT})_\tau \quad F(x_\tau, y_\tau, z_\tau) = \begin{pmatrix} 0 \\ 0 \\ \tau \mathbf{e} \end{pmatrix}, \quad (x, s) > 0.$$

We note that by letting  $\tau \rightarrow 0$ , the equations in  $(\mathcal{KKT})_\tau$  approximate to the ones in  $(\mathcal{KKT})$ .

### 2.2.3 Nonlinear Programming

Let us recall the general constrained optimization problem from above:

$$(\mathcal{CP}) \quad \text{minimize } f(x) \quad \text{subject to } x \in M,$$

where  $M$  is the feasible set.



From the previous chapter, we know which techniques are considered when the objective function and the constraints in  $(\mathcal{CP})$  are linear. In the nonlinear case of  $(\mathcal{CP})$ , two well known methods which are referred as *penalizing methods* come into the mind: *penalty* and *barrier*. The motivation of these methods is the desire to solve the constrained optimization problems by using techniques of *unconstrained* optimization. To convert a constrained optimization problem to an unconstrained one, an auxiliary function is inserted into the objective function where penalty terms are employed for the violations of the constraints [38, 48].

We will use the ideas behind the penalizing methods when we explain the approaches to DFO with support vector machines in Chapter 3. Especially, understanding the difference between the approaches of barrier methods and penalty methods will be necessary there.

In *penalty methods*, the idea is to impose a *penalty* for infeasibility and forcing the iterates to tend to a feasible solution while violating a constraint.

**Example 2.9.** [48] Consider our problem  $(\mathcal{CP})$  with equality constraints only, namely,  $|I| \geq 1$  and  $J = \emptyset$ . The *penalty function* which is a combination of the objective function  $f(x)$  and the equality constraints  $h_i(x)$  will be:

$$f(x) + \frac{1}{2\mu} \sum_{i \in I} h_i^2(x),$$

where  $\mu > 0$  is referred to as a *penalty parameter*. This penalty function is now handled as the objective function of an unconstrained minimization problem. The problem with this new objective function will be solved for a series of decreasing values of  $\mu$  to obtain the solution of the constrained optimization problem.  $\square$

In *barrier methods*, the idea changes to impose a *barrier* to ensure that a feasible solution never becomes infeasible when reaching the boundary given by an inequality constraint.

**Example 2.10.** [48] Consider our problem  $(\mathcal{CP})$  with inequality constraints

only, namely,  $I = \emptyset$  and  $|J| \geq 1$ . The well-known logarithmic barrier function will have the form

$$f(x) - \mu \sum_{j \in J} \log g_j(x),$$

where  $\mu > 0$  is now referred to as a *barrier parameter*. By minimizing this logarithmic barrier function for a series of decreasing values of  $\mu$ , i.e.,  $\mu \rightarrow 0^+$ , an approximate feasible solution of the original constrained problem will be obtained under certain conditions.  $\square$

**Remark 2.11.** The iterations in barrier methods converge to a solution of  $(\mathcal{CP})$  from the interior of the feasible region. Therefore, they are called *interior-point penalty function* methods [38, 48]. However, in penalty methods the iterations generate a sequence of points that converge to a solution of  $(\mathcal{CP})$  from the exterior of the feasible region. Therefore, they are called *exterior-point penalty function* methods [48].  $\square$

**Remark 2.12.** The most famous example of a barrier method is the interior-point method for linear programming which we mentioned in Subsection 2.2.2.  $\square$

# CHAPTER 3

## DERIVATIVE FREE OPTIMIZATION

### 3.1 What is DFO?

DFO [14] has been developed for solving constrained optimization problems of the following form:

$$\begin{aligned} & \min && f(x) \\ (\mathcal{P}) \quad & \text{such that} && a_i \leq c_i(x) \leq b_i \quad (i = 1, 2, \dots, m), \\ & && x \in \mathbb{F} \subseteq \mathbb{R}^n, \end{aligned}$$

where the objective function  $f(x)$  and the constraints  $c_i(x)$  are expensive to compute at a given vector  $x$  and the derivatives of  $f$  at  $x$  are *not* available. Here,  $x \in \mathbb{F}$  represents *easy constraints*, where  $\mathbb{F}$  stands, e.g., for an open or closed elementary geometrical body such as a cube or a ball, and  $c_i(x)$  ( $i = 1, 2, \dots, m$ ) are *difficult constraints*. We note that also *inequalities* may be allowed as difficult constraints in  $(\mathcal{P})$ . The idea is to approximate the objective function by a *model* which is assumed to describe the objective function well in a *trust-region* without explicitly modeling its derivatives. This model is computationally less expensive to evaluate and easier to optimize than the objective function itself.

The model is obtained by interpolating the objective function using a *quadratic interpolation* polynomial. Quadratic interpolation is preferred to approximate the objective function since it can be used successfully within a trust-region method. In the following sections, we will present modern approaches given by support vector machines, too.

### 3.1.1 Trust-Region Framework

In Chapter 2, we explained the *trust-region methods* as one of the solution procedure class for treating (*UP*). 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.

### 3.1.2 Quadratic Interpolation

Consider the problem of interpolating a given or suitably chosen function  $f(x)$  from  $\mathbb{R}^n$  into  $\mathbb{R}$  by a quadratic polynomial  $Q(x)$  at a chosen set of points  $Y = \{y^1, y^2, \dots, y^p\} \subseteq \mathbb{R}^n$ . The quadratic polynomial  $Q(x)$  is an interpolation of the function  $f(x)$  with respect to the set  $Y$  if

$$Q(y^j) = f(y^j) \quad (j = 1, 2, \dots, p), \quad (3.1.1)$$

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 Chapter 2 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  $\phi_i(\cdot)$  ( $i = 1, 2, \dots, q$ ). Then, any quadratic polynomial can be written in terms of these basis functions, i.e.,  $Q(x) = \sum_{i=1}^q \alpha_i \phi_i(x)$ , where the coefficient vector  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_q)^T$  is to be determined. We need

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

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 (3.1.2). When this is the case, the system of linear equations

$$\sum_{i=1}^q \alpha_i \phi_i(y^j) = f(y^j) \quad (j = 1, 2, \dots, p) \quad (3.1.2)$$

can be solved to derive interpolation parameters.

The parameter or coefficient matrix of this system is of type  $p \times q$  and looks as follows:

$$\Phi(Y) := \begin{pmatrix} \phi_1(y^1) & \phi_2(y^1) & \cdots & \phi_q(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \cdots & \phi_q(y^2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(y^p) & \phi_2(y^p) & \cdots & \phi_q(y^p) \end{pmatrix}. \quad (3.1.3)$$

For a given set of points and a set of function values, an interpolation polynomial exists and is unique if and only if  $\Phi(Y)$  is square, i.e.,  $p = q$ , and nonsingular. Theoretically, this means that the system (3.1.2) can be solved, but in practice the solvability of this system depends on whether the matrix  $\Phi(Y)$  is ill-conditioned or not [55].

From the above arguments, we conclude that if we manage to determine the quadratic polynomial uniquely, then we have  $p = q = \frac{1}{2}(n+1)(n+2)$ . However, we need to be aware of the fact that not any  $\frac{1}{2}(n+1)(n+2)$  points in  $\mathbb{R}^n$  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 [53].

**Definition 3.1.** [55] 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.  $\square$

**Remark 3.2.** 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.  $\square$

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

**Example 3.3.** [55] Suppose  $n = 2$  and  $Y$  is a set of six points on a unit circle. Then,  $Y \subset \mathbb{R}^2$  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  $a_0 + a_1x_1 + a_2x_2 + a_{1,1}x_1^2 + a_{1,2}x_1x_2 + a_{1,1,1}x_1^3$ . Therefore,  $Y$  is poised in an appropriate subspace of the space of cubic polynomials.  $\square$

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

**Example 3.4.** [25] Consider the two quadrics<sup>1</sup>  $q_1(x, y) = 2x + x^2 - y^2$  and  $q_2(x, y) = x^2 + y^2$ , whose intersection curve  $I$  projects in the  $(x, y)$ -plane to the

---

<sup>1</sup>A *quadric* is a surface defined by an implicit equation of degree 2 in  $\mathbb{R}^3$

conics  $C(x, y) = 0$  with  $C(x, y) = x - y^2$  (see Figure 3.1). Namely,

$$\begin{aligned} 2x + x^2 - y^2 &= x^2 + y^2, \\ \Leftrightarrow 2x &= 2y^2, \\ \Leftrightarrow x &= y^2. \end{aligned}$$

The conics can be seen in Figure 3.1:

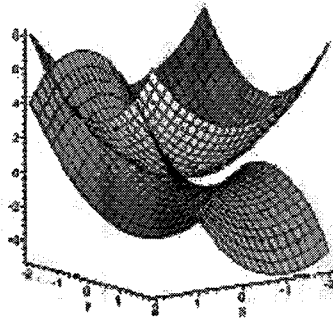


Figure 3.1: Two quadrics whose intersection curve  $I$  projects onto the parabola  $C : x = y^2$  [25].

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<sup>2</sup> of  $q_1$  and  $q_2$  goes through  $I$ .  $\square$

**Definition 3.5.** 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.  $\square$

For the sake of completeness, a brief presentation of  $\Lambda$ -poisedness can be found in the last section of this chapter.

As we mentioned before, a set of points is poised if  $\Phi(Y)$  is nonsingular with respect to the space of quadratic polynomials [55]. If we look for an understanding of this by the fact that an interpolation polynomial exists and is unique if and only if  $\Phi(Y)$  is square and nonsingular, then we conclude:

---

<sup>2</sup>A *pencil of quadrics* generated by two quadrics  $q_1$  and  $q_2$  is the set of quadrics:  $\{\lambda q_1 - q_2 \mid \lambda \in \mathbb{R} \cup \{\infty\}\}$ ; i.e., the ‘infinite’ quadric  $q = “\infty”$  is included.

$Y$  is poised if the determinant of  $\Phi(Y)$  is nonvanishing, i.e., if

$$\delta(Y) := \det \begin{pmatrix} \phi_1(y^1) & \phi_2(y^1) & \cdots & \phi_q(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \cdots & \phi_q(y^2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(y^p) & \phi_2(y^p) & \cdots & \phi_q(y^p) \end{pmatrix} = \det(\Phi(Y)) \quad (3.1.4)$$

$\neq 0.$

The measure of poisedness in a DFO algorithm can be explained by a methodology based on *Newton fundamental polynomials* [14]. 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 [65] instead of the Newton fundamental polynomials.

Let us focus on the Newton fundamental points [14]: The points  $y$  in our interpolation set  $Y = \{y^1, y^2, \dots, y^p\}$  which is a subset of  $\mathbb{R}^n$  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 3.6.** A single *Newton fundamental polynomial* of degree  $l$  corresponds to each point  $(y^i)^{[l]} \in Y^{[l]}$  satisfying the following conditions:

$$N_i^l (y^j)^{[m]} = \delta_{ij} \delta_{lm} \quad \text{for all } (y^j)^{[m]} \in Y^{[m]} \quad \text{with } m \in \{0, 1, 2, \dots, l\}. \quad \square$$

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

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



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 (see Figure 3.2).

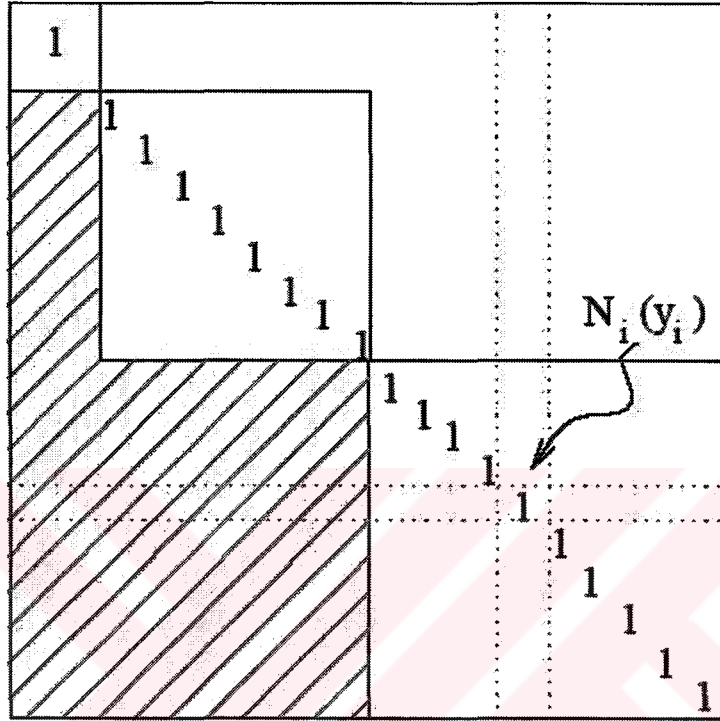


Figure 3.2: Newton fundamental polynomials [14].

Hence,  $Y^{[0]}$  has a single element,  $Y^{[1]}$  has  $n$  elements and  $Y^{[2]}$  has  $\frac{n(n+1)}{2}$  elements. The basis  $\{N_i(\cdot)\}$  of NFP is also partitioned into three blocks  $\{N_i^0(\cdot)\}$ ,  $\{N_i^1(\cdot)\}$  and  $\{N_i^2(\cdot)\}$  with the appropriate number of elements in each block. The unique element of  $\{N_i^0(\cdot)\}$  is a polynomial of degree zero, each of the  $n$  elements of  $\{N_i^1(\cdot)\}$  is a polynomial of degree one and, finally, each of the  $\frac{n(n+1)}{2}$  elements of  $\{N_i^2(\cdot)\}$  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  $\{N_i^l(\cdot)\}$ . A Newton fundamental polynomial  $N_i(\cdot)$  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 3.7.** Consider the quadratic interpolation on a plane. We require six interpolation points using three blocks:

$$Y^{[0]} = \{(0, 0)\}, Y^{[1]} = \{(1, 0), (0, 1)\}, \text{ and } Y^{[2]} = \{(2, 0), (1, 1), (0, 2)\}$$

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:

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

□

### 3.1.3 Algorithm

Now, we are ready to present an outline of the DFO algorithm which can be found in [16] and [55].

**Algorithm 3.8.** The steps of DFO are as follows:

#### Step 0: *Initialization*

Let be given a starting point  $x_s$  and the value  $f(x_s)$ . Choose an initial trust-region radius  $\Delta_0 > 0$ . Choose at least  $n$  additional points not further than  $\Delta_0$  away from  $x_s$  to create an initial well-poised interpolation set  $Y$  and initial basis of Newton fundamental polynomials. Determine  $x_0 \in Y$  which has the best objective function value; i.e.,  $x_0$  solves the *enumeration problem*

$$(\mathcal{EP})_0 \quad \min f(x) \text{ subject to } x \in Y.$$

Set  $k = 0$ , choose parameters  $\eta_0, \eta_1$ , where  $0 < \eta_0 < \eta_1 < 1$  and  $0 < \gamma_0 \leq \gamma_1 < 1 \leq \gamma_2$ .

#### Step 1: *Build the model*

Using the interpolation set  $Y$  and basis of NFP, build a quadratic interpolation polynomial  $Q_k(x)$ .

**Step 2: *Minimize the model within the trust-region***

Set

$$\mathcal{B}_k = \{x \in \mathbb{R}^n \mid \|x - x_k\| \leq \Delta_k\}.$$

Compute the point  $\hat{x}_k$  solving the quadratic optimization problem

$$(\mathcal{QP})_k \quad \min Q_k(x) \quad \text{subject to} \quad x \in \mathcal{B}_k.$$

Compute  $f(\hat{x}_k)$  and the ratio

$$\rho_k := \frac{f(x_k) - f(\hat{x}_k)}{Q_k(x_k) - Q_k(\hat{x}_k)}.$$

**Step 3: *Update of the interpolation set***

- If  $\rho_k \geq \eta_0$ , include  $\hat{x}_k$  in  $Y$ , dropping one of the existing interpolation points if necessary.
- If  $\rho_k < \eta_0$ , include  $\hat{x}_k$  in  $Y$ , if it improves the quality of the model.
- If  $\rho_k < \eta_0$  and there are less than  $n + 1$  points in the intersection of  $Y$  and  $\mathcal{B}_k$ , generate a new interpolation point in  $\mathcal{B}_k$ , while preserving/improving well-posedness.
- Update the basis of the Newton fundamental polynomials.

**Step 4: *Update of the trust-region radius***

- If  $\rho_k \geq \eta_1$ , increase the trust-region radius:

$$\Delta_{k+1} \in [\Delta_k, \gamma_2 \Delta_k].$$

- If  $\rho_k < \eta_0$  and the cardinality of  $Y \cap \mathcal{B}_k$  was less than  $n + 1$  when  $\hat{x}_k$  was computed, reduce the trust-region radius:

$$\Delta_{k+1} \in [\gamma_0 \Delta_k, \gamma_1 \Delta_k].$$

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

**Step 5: Update of the current iterate**

Determine  $\bar{x}_k$  with the best objective function value by solving the discrete problem of enumeration

$$(\mathcal{EP})_k \quad \min f(x) \quad \text{subject to} \quad x \in Y \setminus \{x_k\}.$$

If improvement is sufficient (in the sense of prediction), i.e.,

$$\frac{f(x_k) - f(\bar{x}_k)}{Q_k(x_k) - Q_k(\hat{x}_k)} \geq \eta_0,$$

then we put

$$\bar{\rho}_k := \frac{f(x_k) - f(\bar{x}_k)}{Q_k(x_k) - Q_k(\hat{x}_k)}.$$

Set  $x_{k+1} := \bar{x}_k$ . Otherwise, set  $x_{k+1} := x_k$ . Increase  $k$  by one,  $k \leftarrow k + 1$ , and go to Step 1.  $\square$

**Remark 3.9.** [55] The interpolation set is enlarged with the points which are generated by minimizing the model at each iteration. These points are expected to contribute to the minimization of the objective function value. Another aspect in generating interpolation points during the iteration is to improve the interpolation set and the model. But, whatever the aim in generating the interpolation points, it is important to gather as much information as possible from these points about the function value to get rid of the function evaluations which are usually very expensive and time consuming.  $\square$

**Remark 3.10.** [55] It is sufficient that the interpolation set  $Y$  contains just two points at the beginning of the algorithm. Although the cardinality of  $Y$  is

not required to exceed  $n + 1$ , it is advantageous to increase the size of  $Y$  as long as it remains well-poised and contains local information. On the other hand, maintaining full quadratic interpolation may be too expensive. For example, if  $n = 20$ , it requires  $p = \frac{(n + 1)(n + 2)}{2} = 231$  interpolation points to build a full quadratic model. It may be unacceptable to evaluate the objective function 231 times before obtaining any progress in the algorithm.  $\square$

**Remark 3.11.** In step 3, we see that there is no obligation in choosing a technique to update the interpolation set. There are various ways of improving an interpolation set and various criteria for adding or deleting interpolation points. In [55], the preferred technique is based on the properties of Newton fundamental polynomials concerning threshold pivoting.  $\square$

## 3.2 Derivative Free Optimization via Support Vector Machines

### 3.2.1 Introduction into SVMs

Conn, Scheinberg and Toint [19] suggest a *derivative free optimization method via support vector machines* (SVM). This section serves for a short introduction into this approach. The goal of their study is to solve any optimization problem of the form

$$\begin{aligned}
 (\mathcal{P}) \quad & \min && f(x) \\
 & \text{such that} && x \in \mathbb{M} \subseteq \mathbb{R}^n,
 \end{aligned}$$

where  $\mathbb{M}$  is the feasible set. For solving  $(\mathcal{P})$  they use a derivative free optimization method where the model of the objective function, explained in the previous chapter as  $Q(x)$ , is derived by means of *support vector machines* instead of *quadratic interpolation*.

The aims behind the idea of integrating into derivative free optimization (DFO), a support vector machine which is in fact a *supervised learning* algorithm (see

[36] for details), are the following [19]:

- to improve the *convergence* properties of DFO,
- to decrease the *tolerance* to noise in the model  $Q(x)$ ,
- to guarantee the *good initial reduction* in Step 2 of Algorithm 3.8,
- to take into account the appearance of *outliers* and to rule them out, and
- to exploit the *geometry* of the interpolation points.

In order to understand the basic ideas of SVMs better, we explain them for a standard application in *classification* theory firstly.

### 3.2.2 SVMs in Classification

Consider the training vectors  $p_i \in \mathbb{R}^n$  ( $i = 1, 2, \dots, m$ ) from the training data set  $P$ . Given two classes of these vectors, for all  $i = 1, 2, \dots, m$  we put:

$$y_i := \begin{cases} 1, & \text{if } p_i \text{ in class 1,} \\ -1, & \text{if } p_i \text{ in class 2.} \end{cases} \quad (3.2.5)$$

Let us assume that we are able to divide the data set into two given disjoint classes by a *separator* without any misclassification. A *separator* is a linear classifier that separates the data. In higher dimensional spaces, a separator classifier is called a *hyperplane classifier*. For example, in the 2-dimensional spaces a hyperplane is simply a line, and in 3-dimensional spaces a hyperplane is a 2-dimensional affine subspace [3]. We will also mention what the approach is when we allow training errors to separate the data set into two classes [45].

In *support vector machines* method, a hyperplane  $w^T p + b = 0$  is found for partitioning the data set as shown in (3.2.6) (see Figure 3.3) such that

$$\begin{aligned} (w^T p_i) + b &> 0 & \text{if } y_i = 1, \\ (w^T p_i) + b &< 0 & \text{if } y_i = -1, \end{aligned} \quad (3.2.6)$$

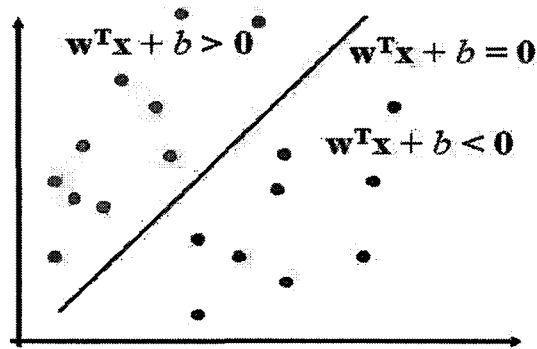


Figure 3.3: SVM finds a hyperplane for separating the data set into two classes [45].

where  $y_i$  is defined by (3.2.5).

This hyperplane maintains a *maximum margin* from any point in the data set. Statistical learning theory suggests that choosing the maximum margin hyperplane for separation of the data into two classes enables one to make a maximal generalization to classify the unseen data points [57].

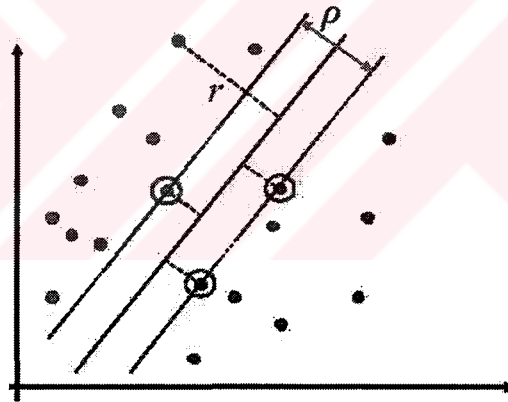


Figure 3.4: Support vectors and the margin  $\rho$  [45].

As delineated in Figure 3.4, let us define  $r$ , the distance from any data point to the separator:  $r := \frac{w^T p + b}{\|w\|}$ . Any data point closest to the separator is called a *support vector*. Furthermore, we define  $\rho$  which is the *margin*, i.e., the distance between  $w^T p + b = +1$  and  $w^T p + b = -1$ .

Assuming all the data are at least distance 1 from the hyperplane, we rewrite

(3.2.6) from above as follows:

$$\begin{aligned} (w^T p_i) + b &\geq 1, & \text{if } y_i = 1, \\ (w^T p_i) + b &\leq -1, & \text{if } y_i = -1. \end{aligned} \tag{3.2.7}$$

For *support vectors* which are defined as the closest points to the separator, the inequalities become equalities, i.e.,  $(w^T p_i) + b = 1$  and  $(w^T p_i) + b = -1$ .

Therefore, the distances of support vectors to the separator satisfying  $(w^T p_i) + b = 1$  and  $(w^T p_i) + b = -1$  are  $r := \frac{1}{\|w\|}$  and  $r := \frac{-1}{\|w\|}$ , respectively. By acknowledging the distance 2 between  $-1$  and  $1$ , the support vectors on different sides of the separator are far from each other by the distance of

$$\rho = \frac{2}{\|w\|}, \tag{3.2.8}$$

which is equal to the margin. A SVM finds the maximum of this margin by solving the following optimization problem:

$$\begin{aligned} \max \quad & \frac{2}{\|w\|} \\ \text{subject to} \quad & \text{the constraints (3.2.7)}. \end{aligned}$$

### 3.2.3 SVMs in DFO

After an equivalent reformulation of this maximization problem [44], we obtain a quadratic problem which will be solved by the support vector machine algorithm and can be stated as follows:

$$(MIP) \quad \begin{cases} \min_{w,b,y} & \frac{1}{2} w^T w \\ \text{such that} & y_i((w^T p_i) + b) \geq 1 \quad (i = 1, 2, \dots, m). \end{cases}$$

Here, optimizing the objective function just comes from an equivalent reformulation of (3.2.8). This is a *mixed-integer problem*, since the vector  $w$  and the scalar  $b$  are real, while the vector  $y = (y_1, y_2, \dots, y_m)^T$  consists of the integers  $y_i \in \{-1, +1\}$ .



As we mentioned, it is usual to be faced with non-separable data in real-world problems for which there exists no hyperplane to separate the data set into two different classes without any misclassification. The theory of support vector machines suggests a way of mapping the inseparable data into a higher-dimensional space and defines a separating hyperplane there, to overcome the problem of inseparability. This higher-dimensional space is called the *feature space*. Any consistent training set can be made separable with an appropriately chosen feature space of sufficient dimensionality. We remark here that it is a computationally expensive task to translate the training set into a higher-dimensional space. Moreover, this artificial way of separation may result with trivial solutions that overfit data. To give a short indication about this increase of dimensions, think of polynomial regression model, e.g., in two variables  $x_1$  and  $x_2$ . We substitute  $\tilde{x}_1 := x_1$ ,  $\tilde{x}_2 := x_2$ ,  $\tilde{x}_3 := x_1^2$ ,  $\tilde{x}_4 := x_1 \cdot x_2$  and  $\tilde{x}_5 := x_2^2$ , arising in feature space of dimension 5. If our given dimension is still 2, but the degree of the regression polynomial is  $q$ , then the feature space dimension is becoming  $\frac{q(q+1)}{2}$ . For further details, we refer to the theory of statistic and learning (linear discriminant analysis and logistic regression [36]).

In the following, we explain the ideas of Conn, Scheinberg and Toint by still referring to the linear case of a SVM. For a better understanding, recall that we are back to *minimization* of  $f$  again which was not the case with the SVM before. The model  $Q(p)$  in DFO used by them is:

$$Q(p) := w^T p + \beta,$$

illustrated in Figure 3.5. This model means a *linear approximation* of the unknown function  $f$ , where we only know the perturbed (noisy) values of  $f$ .

In Figure 3.5, the shown SVM model is constructed by referring to finitely many data points (measurements). This figure also reflects the realization of the function affected by noise in the linear case.

Now, the adjusted form of the ( $\mathcal{MIP}$ ) defined above becomes considered in our DFO algorithm as follows:

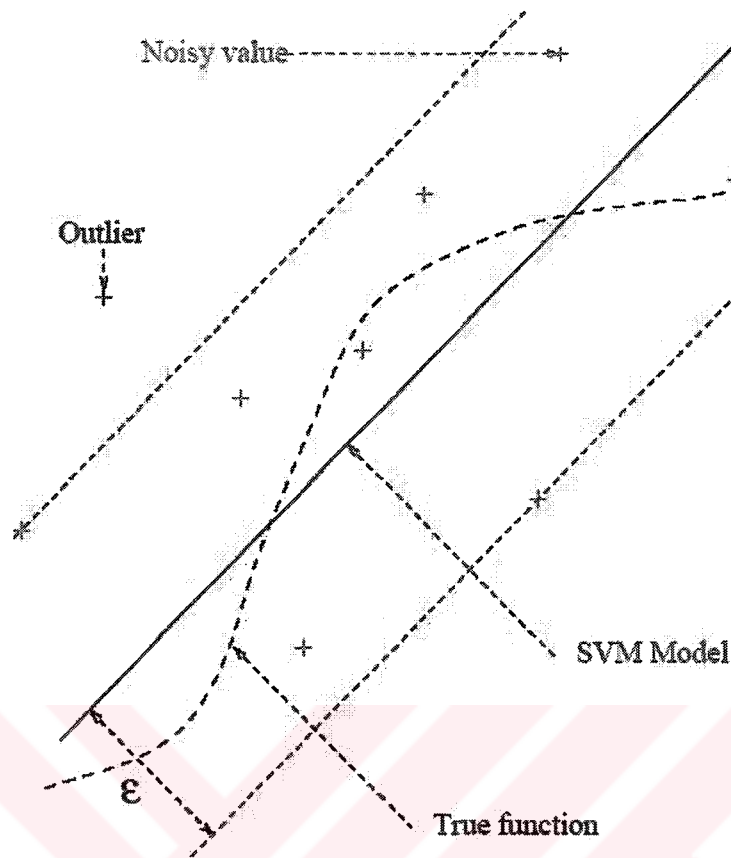


Figure 3.5: Linear support vector machines model [19].

$$(\mathcal{P})_{SVM} \begin{cases} \min & \frac{1}{2}w^T w \\ \text{such that} & |w^T p_i + \beta - f(p_i)| \leq \epsilon \quad (i = 1, 2, \dots, m). \end{cases}$$

Here,  $w$  can be imagined to be an *approximation* to the first derivative of  $f(x)$ , i.e., to the gradient of  $f(x)$ , and  $\beta$  is the *intercept* we remember from inverse problems and statistical learning [2, 67]. The aim behind the formulation of this optimization problem  $(\mathcal{P})_{SVM}$  is to minimize the gradient of  $f$  in norm, i.e., to push it or its approximation to zero in norm:  $\|w\| \rightarrow 0$ . The constraints of  $(\mathcal{P})_{SVM}$  reflect that some error in our model is allowed, i.e., some difference between the model values and the measurement (data) values, the latter ones being considered as perturbed evaluations of  $f$ . Hence, the problem can be stated as finding  $w$  and  $\beta$  under error bounds of a given tolerance  $\epsilon$ .

Now, we rephrase and, additionally, *relax* quadratic optimization problem  $(\mathcal{P})_{SVM}$ :

$$(\mathcal{RP})_{SVM} \quad \left\{ \begin{array}{l} \min_{\xi, \xi', w, \beta} \quad \frac{1}{2} w^T w + c \sum_{i=1}^m (\xi_i + \xi'_i) \\ \text{such that} \quad w^T p_i + \beta + \xi_i \geq f(p_i) - \epsilon \quad (i = 1, 2, \dots, m), \\ \quad \quad \quad w^T p_i + \beta - \xi'_i \leq f(p_i) + \epsilon \quad (i = 1, 2, \dots, m), \\ \quad \quad \quad \xi, \xi' \geq 0. \end{array} \right.$$

Here,  $c$  is a parameter to bring a balance in approximately fitting any outliers (see Figure 3.5);  $\xi_i$  and  $\xi'_i$  are the perturbation parameters which imply an *exterior-point* approach. By  $\xi = (\xi_1, \xi_2, \dots, \xi_m)^T$  and  $\xi' = (\xi'_1, \xi'_2, \dots, \xi'_m)^T$ , one can *tune* the amount of violation of constraints from  $(\mathcal{P})_{SVM}$ , both being requested to be nonnegative coordinate-wise.

**Remark 3.12.** In the theory of *inverse problems*, we know about the *tradeoff* between the minimization of the error (mostly in terms of sums of squares), and the minimization of the norm of the unknown parameter vector. This second goal can be interpreted as a diminishment of the system's complexity and, herewith, a diminishment of numerical instability. In statistical learning [36, 66], this wish for a “small” parameter vector is treated by testing of zero hypotheses. Here, in our approach by SVMs, the authors give priority to this second goal and, as we explained, this is associated with the minimization of  $f(x)$  also. Furthermore, concerning the first goal, a tolerance is given. For the minimization of  $f$ , the parameter  $\beta$  is not important; for a reason like this,  $\beta$  was not included into the parameter vector  $w$  which has to be minimized in norm. There are further approaches and methods developed to deal with this tradeoff, e.g., by so-called *Tikhonov regularization* [2].  $\square$

The solution of  $(\mathcal{RP})_{SVM}$  involves constructing a *dual problem*, introduced in Chapter 2, where a Lagrange multiplier vector  $(\alpha^T, \gamma^T)^T$  is associated with all the constraints in the problem.

As we explained in Section 2.2 on constrained optimization, we obtain a *Karush-Kuhn-Tucker* condition with respect to the variable  $w$  of the following form:

$$(KKT) \quad \begin{cases} \frac{\partial L}{\partial w}(w, \alpha, \gamma) = w - P^T \alpha + P^T \gamma = 0, \\ \alpha, \gamma \geq 0, \end{cases}$$

where  $L$  is the *Lagrange function* of  $(\mathcal{RP})_{SVM}$ ,  $P^T = (p_1, p_2, \dots, p_n)^T$  and  $0 = 0_n$  is the origin in the input and sample space  $\mathbb{R}^n$ .

This *Karush-Kuhn-Tucker* condition (*KKT*) immediately gives

$$w = (P^T \mid -P^T) \begin{pmatrix} \alpha \\ \gamma \end{pmatrix}.$$

Substituting  $w$  in this way yields that the following representation of  $(\mathcal{RP})_{SVM}$  can in vector notation be written as:

$$\begin{aligned} \min_{\xi, \xi', \alpha, \gamma, \beta} \quad & \frac{1}{2}(\alpha^T, \gamma^T) Q \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} + c \sum_{i=1}^m (\xi_i + \xi'_i) \\ \text{such that} \quad & Q_1 \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} + \beta e + \xi \geq f(P) - \epsilon e, \\ & Q_1 \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} + \beta e - \xi' \leq f(P) + \epsilon e, \\ & \xi, \xi' \geq 0, \quad 0 \leq \alpha, \gamma \leq c, \end{aligned}$$

where  $e := (1, 1, \dots, 1)^T \in \mathbb{R}^n$  and

$$Q = \begin{pmatrix} Q_1 \\ -Q_1 \end{pmatrix} := \begin{pmatrix} PP^T & -PP^T \\ -PP^T & PP^T \end{pmatrix}.$$

Here, the important feature consists in the construction of the model  $Q$ . Although it is explained in the first section of this chapter that the model is usually obtained from *quadratic interpolation*, now we can also use *approximation* by support vector machines to construct our model for DFO.

### 3.2.4 $\Lambda$ - Poisedness

Let us finish this chapter with the definition of an issue used by Conn, Scheinberg and Vicente [18] in the algorithmic framework of their studies for building and maintaining sample sets with good geometry. They have suggested the

notion of a  $\Lambda$ -poised sample set, where  $\Lambda$  is a nonnegative constant.

**Definition 3.13.** Given  $\Lambda \geq 0$ ,  $P$  is  $\Lambda$ -poised if for any  $x \in \mathcal{B}(\Delta)$ , there are numbers  $\lambda_1, \lambda_2, \dots, \lambda_m \in \mathbb{R}$  such that

$$x - p_1 = \sum_{i=2}^m \lambda_j (p_i - p_1), \quad |\lambda_j| \leq \Lambda \quad (j = 1, 2, \dots, m). \quad (3.2.9)$$

Here,  $\mathcal{B}(\Delta) = \{x \in \mathbb{R}^n \mid \|x\|_2 \leq \Delta\}$  is a closed ball of radius  $\Delta$  centered at the origin and  $P = \{p_1, p_2, \dots, p_m\}$ .  $\square$

We may assume that  $p_1 = 0$  and, herewith,  $p_1$  is the center of  $\mathcal{B}(\Delta)$ . This assumption can be made without loss of generality, since it can always be satisfied by a translation performing  $p \mapsto 0$ . For example, the smallest ball around the origin containing  $P$  may just be  $\mathcal{B}(\Delta)$ , i.e.,  $P \subset \mathcal{B}(\Delta)$  tightly.

Definition 3.13 deals with the governing of the relation between the sets  $P$  and  $\mathcal{B}(\Delta)$ . We see that  $\Lambda$ -poisedness is a condition that can be interpreted as a well-distribution condition of  $P$  with respect to  $\mathcal{B}(\Delta)$ . Thus, we conclude that the points  $p_i$  should not lie too close and not much on a line etc.. The nonnegative constant  $\Lambda$  is “controlling” about how far the vectors are stretched, i.e., expanded or contracted.

Considering this well-distribution issue, we observe that in Definition 3.13 the equation (3.2.9) is a linear combination, controlled by a condition generalizing a convex combination:  $|\lambda_j| \leq \Lambda$  instead of both  $0 \leq \lambda_j \leq 1$  and  $\sum_{i=2}^m \lambda_i = 1$ .

Herewith, the set  $\mathcal{B}(\Delta)$  is generated (spanned) by the sample set  $P$ . Hence,  $P$  must be sufficiently “independent” in order that such a spanning of  $\mathcal{B}(\Delta)$  is possible.

We note that in the context of the studies of Conn, Scheinberg and Vicente [18],  $\mathcal{B}(\Delta)$  can be interpreted as the *trust-region*.

## CHAPTER 4

# OPTIMIZATION OF STIRRER CONFIGURATIONS

In this chapter, we introduce and treat the optimization problem of configuring a stirrer which is a common device for mixing different substances and, then, we explain the implementation of the numerical tool which is employed for finding the optimized stirrer configuration. This research serves for various practical applications in chemistry [72], petrol or food engineering, in pharmacy, medicine and waste recycling [21]. Optimization of stirring configurations has been the scientific content of a joint project supported by the Volkswagen Foundation between Institute of Applied Mathematics of METU together with the Department of Mechanical Engineering of Darmstadt University of Technology.

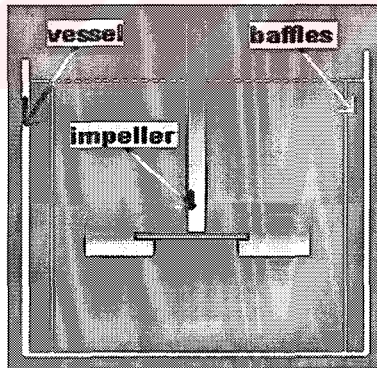


Figure 4.1: A typical stirrer tank [72].

A typical *stirred vessel* consists of three parts: vessel, baffles and impeller as shown in Figure 4.1 [72]. There are many types of stirrers (and vessels) according to the mixing task they get involved with. In order to choose a suitable

stirrer among the existing ones or to design a new one for a certain process, we need to understand the flow behaviors in the stirrers during the mixing process. We note that the flow model can be described by Navier-Stokes equations [54]. A detailed study on the numerical investigation of the hydrodynamics in the stirrer tanks and the explanations of the employed numerical methods is presented in [72]. Figure 4.2 shows an illustration of the stirrer tank which is considered in this chapter. The design variables of this stirrer which will be the parameters of our optimization problem are also shown.

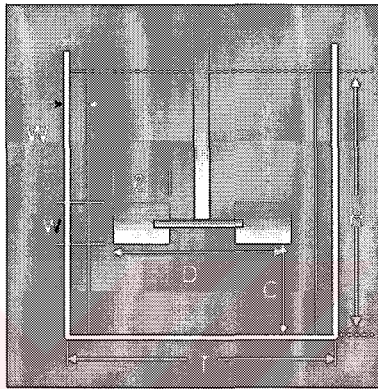


Figure 4.2: The configuration of a stirrer tank [54].

In a standard stirrer tank, the values of these geometrical parameters in terms of the tank diameter are given in Table 4.1:

Parameter	Value
tank (vessel) diameter	$T = 0.15m$
impeller diameter	$D = T/3 = 0.05m$
bottom clearance	$C = H/2 = 0.75m$
height of the liquid	$H = T = 0.15m$
length of the baffles	$W = 3D/10 = 0.015m$
length of the blade	$l = D/4 = 0.0125m$
height of the blade	$w = D/5 = 0.01m$
disc thickness	$x = D/5 = 0.00175m$
diameter of the disc	$D_D = 3D/4 = 0.0375m$

Table 4.1: Geometrical parameters of stirrer configuration [54].

We note that the flow field and mixing process are very complicated even in a



simple stirred vessel. The rotation of the impeller blades makes the fluid in the vessel to interact with the stationary baffles adjacent to the stirrer walls. This generates a complex, three-dimensional turbulent flow. The other parameters of the stirrer tank like impeller clearance from the tank bottom, diameter of the impeller disc and baffle length also affect the generated flow. The nature of the optimization process depends on these design parameters [62].

## 4.1 Optimization Problem of Stirrer Configurations

The problem can be thought as a nonlinear optimization problem with nonlinear constraints as follows [54]:

$$\begin{aligned}
 (\text{SC}) \quad & \text{minimize} && f(\alpha) \\
 & \text{subject to} && \alpha_k^l \leq \alpha_k \leq \alpha_k^r, && (k = 1, 2, \dots, N), \\
 & && h_i(\alpha) = 0 && (i \in I := \{1, 2, \dots, m\}), \\
 & && g_j(\alpha) \geq 0 && (j \in J := \{1, 2, \dots, s\}),
 \end{aligned}$$

where  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_N)^T$ . By using our definition of  $(\mathcal{P})$  from Chapter 3, we recall that  $\alpha \in \mathbb{F}$  corresponds to easy constraint and see now that can be written as  $\mathbb{F} := \prod_{k=1}^N [\alpha_k^l, \alpha_k^r]$ . The functions  $h_i(\alpha)$ ,  $g_j(\alpha)$  are the difficult constraints. The objective function  $f$  may correspond to the Newton number, power number, mixing time, stresses, etc., which are derived from the flow field. The parameters  $\alpha_k$  are the design variables determining the stirrer geometry which are presented in Figure 4.2, or operating parameters like the rotational speed. The equality constraint and the inequality constraint functions  $h_i$  and  $g_j$ , respectively, can be restrictions of the stirrer geometry, of operational parameters, or of other quantities depending on the flow field.



## 4.2 Numerical Toolbox

This tool involves the following components [54]:

- (a) an *efficient parallel multigrid flow solver*,
- (b) a specially designed *parametrized grid generator* for stirrer geometries,
- (c) a *derivative-free optimization method*.

For the configuration of a stirrer, this kind of numerical optimization tools is accepted to be useful as far as it can deal with the geometries of the stirrer.

The flow solver employs a finite volume method for *non-orthogonal, boundary-fitted* block structured grid. In this study, the solver is *Flow Analysis by Solving Transport Equations Simulating Turbulence (FASTEST)* [54].

The grid generator, another component of the optimization tool, is good for providing a parametrized generation of block-structured grids for stirrer geometries. For finding the optimum geometry, abundant modifications of the grids need to be done; this task is easy in parametrized grids [54].

Our derivative free method is *DFO* which is based on a *trust-region framework* with *quadratic interpolation* as already explained in Section 3.1.2.

### 4.2.1 FASTEST 3D

When we consider the complete FASTEST3D program, there are three computer subprograms we encounter [54]:

- (1) a *pre-processor* for generating the numerical grid,
- (2) a *flow predictor*,
- (3) a *post-processor* for the graphical visualization of the result.

The details of the process during these three steps of a typical FASTEST3D program can be stated as follows: The process begins with the generation of the grid according to the shape and the size of the tank and the impeller. The grid generation is the subject of the next section. The code of FASTEST3D requires some parameter inputs such as angular velocity, fluid properties, clicking step size and time step number. The solution to the mixing problem is produced by running the code which yields the quantities characterizing the flow field like velocity components, turbulence, pressure quantities and power consumption.

One of the advantages of FASTEST3D is its *flexibility*. This means that it can be modified according to *different* fluid flow problems. Although it lacks of being a user friendly program, any user with programming skills and practical experience in running the similar programs will not have difficulties to modify the code according to the needs.

After this brief review of FASTEST3D, obviously the reason for choosing it is its flexibility which makes the discretization of even very complex geometries possible. Considering the parallelization procedures of the general flow and of the flow solver, the essential issues are block structured grid partitioning, block connecting, data dependence handling at block interfaces, data structures in the communication and the implementation of the communication in the code [59]. The grid movement of the stirrer grid as against the vessel is handled by a clicking mesh approach. We note that our solver was already proved to be successful in computing complex problems of stirrer technology on parallel computers with high numerical and parallel efficiency [54].

#### 4.2.2 Grid Generation Tool

The success of FASTEST3D in handling the complex geometries associated with stirrer configurations which we mentioned above is based on the *grid generation* process. The grid generation tool overcomes the difficulties related with the geometry since it involves the algebraic method based on *transfinite interpolation* [27] for the generation of a multi-block boundary fitted grid. Moreover, a

built-in library of impellers which are commonly used in chemical and process industries is included. This allows any combination of impeller, baffle, and vessel geometries [54]. The input parameters of the tool are radii of disk (impeller), vessel or numbers and dimensions of blades, baffles, etc.. Therefore, the grid generation is parametrized with respect to the characteristic geometry quantities for the different stirrer types which means that the design variation can easily occur. We note that this concept tells us the possibility of using the geometrical input parameters to the grid generation tool as the design parameters for the optimization purpose.

Without numerical efficiency and robustness in the flow solver, the optimization procedure will demand a large number of individual flow simulations which may lead to “nasty” grids. However, our flow solver FASTEST3D employs a finite-volume solver which specially takes this aspect into account [54].

### 4.2.3 DFO

We know the theoretical details of DFO from Chapter 3. In the context of the numerical tool, DFO is employed for finding the optimum of  $(SC)$ . The main benefit of using a derivative free method is that the local gradient of  $f$  does not have to be provided by the flow solver. As a matter of fact, the gradient of  $f$  with respect to the design variables is not directly available for the complex discrete Navier-Stokes system [50].

## 4.3 Illustration of the Numerical Toolbox

Now, since we described the components of our numerical tool, we can continue with the implementation of the tool as a program code which is schematized in Figure 4.3.

The process can be summarized as follows:

1. **Optimizer**: The optimizer is started and computes a new set of design

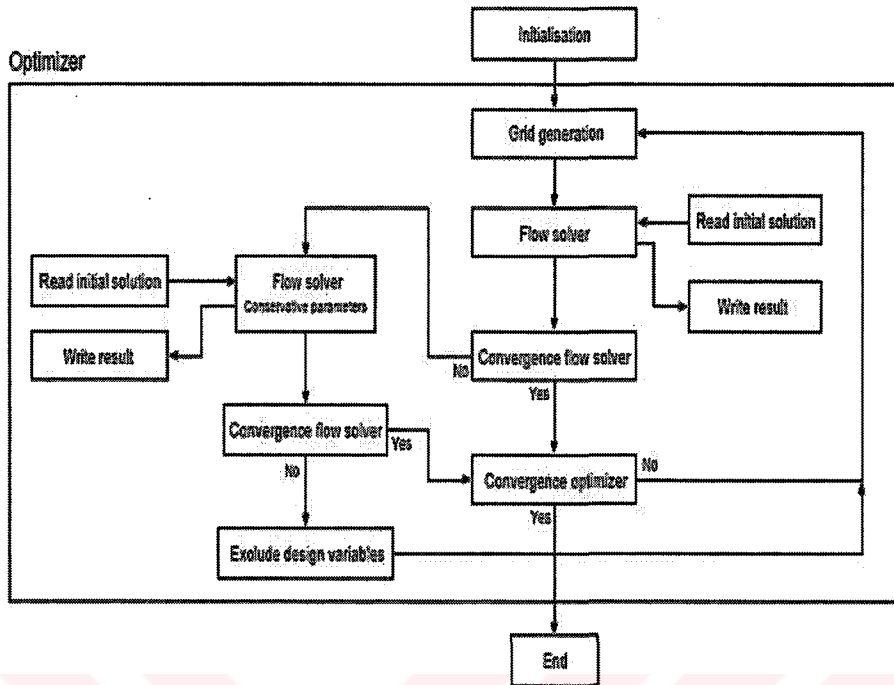


Figure 4.3: Flow chart of control script [54].

variables. Afterwards it turns into a waiting state.

2. **Grid variation:** Getting the signal that the new design variables are available, the grid generation tool becomes active and creates the new geometry and corresponding numerical grid.
3. **Flow simulation:** Getting the signal that the new grid is available, the flow solver computes the flow field and the corresponding objective function for the new geometry. As a starting value, the solution from the previous simulation is used.
4. **Test of flow solver convergence:** If the flow solver has sufficiently much converged, the optimizer gets a signal to continue. It may happen that the flow solver does not converge. Then, another run with more conservative numerical parameters, i.e., reduced relaxation factors, is started. If this also fails, a corresponding set of design variables is excluded.

5. *Test of optimizer convergence*: The optimizer decides by a given criterion, whether the current value of the objective function is accepted as the (approximative) optimum. If yes, then the procedure is finished, if not, then the procedure is continued with Step 1.

## 4.4 Implementation of the Numerical Tool

### 4.4.1 Optimization of the Power Number

Power number is a dimensionless number which relates the resistance force to the inertia force. It is expressed as

$$N_p = \frac{P}{\rho N^3 D_i^5}.$$

Here,  $P$  is the power in *watt*,  $\rho$  is the density in  $kg/m^3$ ,  $N$  is the rotational speed of the impeller in *rev/sec* and  $D$  is the impeller diameter. A plot of the power curve, i.e.,  $N_p$  versus  $Re$  is shown in Figure 4.4.

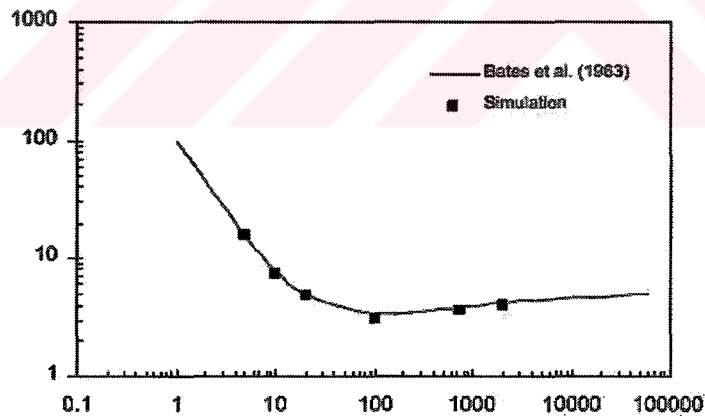


Figure 4.4: Power curve for standard tank configuration [54].

Here,  $Re$  denotes the Reynolds number which can be defined  $Re = \frac{ND^2\rho}{\mu}$  where  $\mu$  is the dynamic fluid viscosity.

The optimum value of the power number is computed with respect to the design variables baffle length and bottom clearance, where the working fluid is glucose solution and Reynolds number is 100. We note that other design variables of the stirrer are hold constant during the optimization process. The rotational speed of the impeller is  $N = 189.54rpm^1$ , density of the glucose solution is  $\rho = 1330kg/m^3$ . These values are used by FASTEST 3D to produce corresponding power number value. The power number value produced by FASTEST 3D are fed into DFO as initial values with the subroutine *FUN*, then the optimum value is computed by DFO. The initial values of the impeller parameters, baffle length ( $W$ ) and bottom clearance ( $C$ ) are also inputted by the subroutine *FUN*. The subroutine *FUN* can be found in Appendix B. The values which are computed at every step of DFO until finding the optimum value is given in Table 4.2.

	clearance (mm)	baffle length (mm)	power number
initial condition	75	15	3.18431998
after 1st loop	75.7615504	15.5398182	3.22962
after 2nd loop	75.0	14.0	3.17557649
after 3rd loop	74.0	13.0	3.16774136
after 4th loop	73.2500126	14.0	3.17688619
after 5th loop	74.2566223	13.5	3.17183382
after 6th loop	74.25	12.75	3.16495611
after 7th loop	74.5	12.5	3.16287491
after 8th loop	74.1267361	12.0	3.15914713
after 9th loop	73.1804206	11.0	3.15205714
after 10th loop	72.106706	9.0	3.13914519
after 11th loop	70.9642869	5.0	3.10280395
after 12th loop	73.0235831	5.0	3.10927274
after 13th loop	71.7418922	5.0	3.1067214
after 14th loop	68.9642269	5.0	3.10168401

Table 4.2: Optimization of power number [54].

We remark that one time step is approximately completed in 20 seconds of computing time on an eight processor Red-stone cluster machine. Hence, a steady state flow in the sense of a frozen rotor computation is reached in about 8 hours computing time [54].

<sup>1</sup>*rpm* is the abbreviation for rotation per minute.

#### 4.4.2 Optimization of the Newton Number

The numerical optimization tool we described can be used to minimize the Newton number for  $Re = 100$ , where the involved design variables are the *disk thickness*  $x$ , the *bottom clearance*  $C$ , and the *baffle length*  $W$ . According to these variables the constraints can be prescribed as  $0.001 \leq x \leq 0.005$ ,  $0.002 \leq C \leq 0.075$  and  $0.005 \leq W \leq 0.03$ . Let us keep the other parameters of the standard configuration constant.

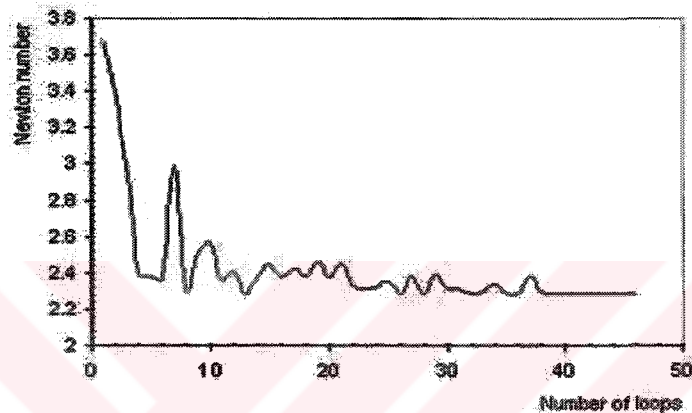


Figure 4.5: Newton number versus number of the loops [54].

Figure 4.5 shows the Newton number versus the number of cycles of the optimization algorithm. From this figure we observe that the Newton number first sharply drops and then slowly approaches the minimum in a slightly oscillating manner.

The Newton number value produced by FASTEST 3D are inputted into DFO as initial values with the subroutine *FUN* (see Appendix B), as well as the initial values of disk thickness ( $x$ ), bottom clearance ( $C$ ) and baffle length ( $W$ ), then the optimum value is computed by DFO.

The change in the values during the optimization process and the optimized Newton number can be seen in Table 4.3.

Figure 4.6 shows the velocity fields in a plane midway between two baffles for the

	clearance (m)	baffle length (m)	disc thickness (m)	Newton number
initial condition	0.10633	0.015	0.00175	3.19967866
after 1st loop	0.129503979	0.0159044909	0.00350440904	2.67351747
after 2nd loop	0.11504933	0.0171563964	0.00500000999	2.73856258
after 3rd loop	0.080999991	0.0167459387	0.00500000992	2.70852542
after 4th loop	0.13676001	0.0170457874	0.00500000999	2.46210527
after 5th loop	0.0810001754	0.030000009	0.000999991356	3.41583061
after 6th loop	0.13676001	0.00499999009	0.00500000999	2.44725537
after 7th loop	0.13676001	0.00927787172	0.00500000998	2.45263529
after 8th loop	0.136759929	0.00500000839	0.000999996444	2.87033367
after 9th loop	0.13675556	0.0128115706	0.00100015391	2.87464595
after 10th loop	0.134806931	0.00597657184	0.00499995707	2.46530557
after 11th loop	0.135783619	0.00500188821	0.0040236691	2.55282521
after 12th loop	0.136759084	0.00527586658	0.00451204941	2.49486613
after 13th loop	0.136560001	0.0051999984	0.00500000791	2.44932532
after 14th loop	0.13676001	0.00519997366	0.00500000999	2.44754863

Table 4.3: Optimization of Newton number for Re=100.

optimized and the standard geometry cases. This figure reveals the differences in the optimized and the standard geometry by means of the flow patterns. We remark that the whole optimization process completes in 5 days on the eight processor Redstone cluster machine.

Finally, Figure 4.7 reflects how the three design variables change during the optimization process. We infer that the optimum value of disk thickness approaches its upper bound, however, the optimum value of the bottom clearance and the baffle length seems to be in between their given bounds.

Here, power consumption is considered but not the total energy consumption. Energy consumption calculation would require some sort of mixing time determination which is beyond the scope of this study. Since only power is considered, there is a tendency in the baffle length to decrease as the solution approaches to the optimum point. It should be noted that baffles introduce turbulence that dissipates mechanical energy. Therefore reducing the baffle length results with a decrease in power consumption.



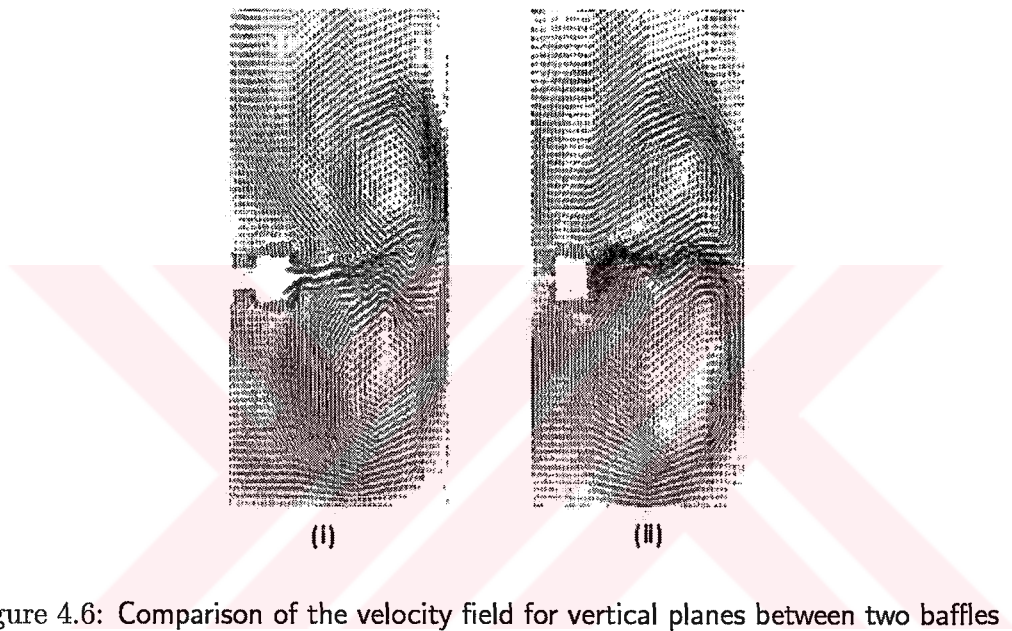


Figure 4.6: Comparison of the velocity field for vertical planes between two baffles at Reynolds number 1000 [54]:  
(i) optimized parameter obtained from DFO,  
(ii) standard parameters.

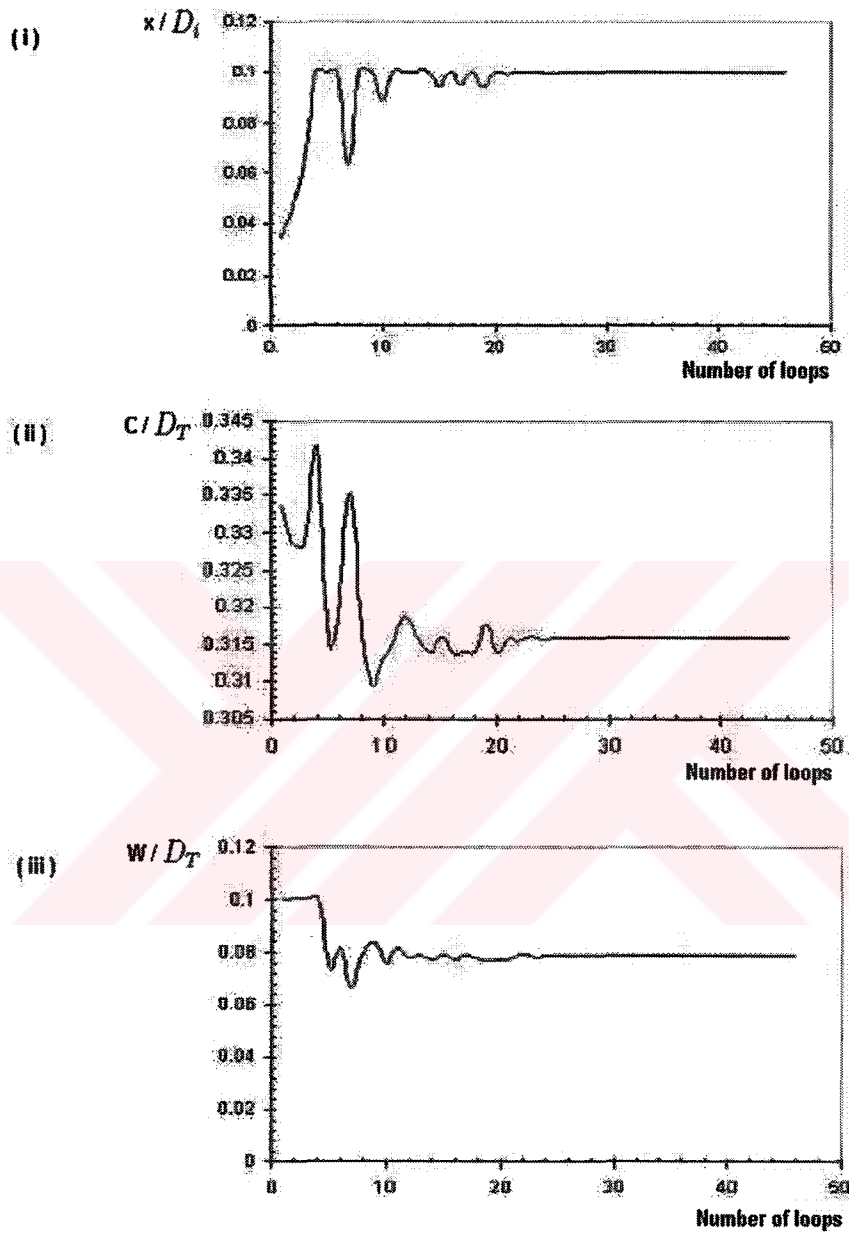


Figure 4.7: Dimensionless impeller parameter versus number of the loops [54]: (i) dimensionless disk thickness , (ii) clearance , (iii) baffle length.

## CHAPTER 5

# DERIVATIVE FREE OPTIMIZATION BY NON-SMOOTH ANALYSIS AND VARIOUS APPLICATIONS

In this chapter, we will examine a *derivative free* algorithm from the paper of Bagirov and Yearwood [6], which is developed for solving the *minimum sum of squares clustering problem*. This problem is a *non-smooth* and *non-convex* optimization problem; in Section 5.2 it is modelled and discussed. The algorithm will be examined in Sections 5.2-5.3, it is based on non-smooth analysis and optimization methods [6]. Herewith, they take into account a further important field of modern continuous optimization [13] which for their application in data mining turns out to be very appropriate and helpful.

A further aim of Bagirov and Yearwood in their study is to discuss the application of this algorithm to large-scale data sets as being reported in Section 5.4. To explain the reasons for the ideas behind their derivative free algorithm, let us firstly take a look at *clustering* in Section 5.1.

### 5.1 Overview of Clustering

In general, the *unsupervised* classification of the patterns is called *clustering*. The clustering process can be thought as dividing patterns into clusters according to their similarities [1, 6].

The aim of cluster analysis is to partition a given finite set  $X$  which has finitely

many points of  $d$ -dimensional space  $\mathbb{R}^d$ , i.e.,

$$X = \{x^1, x^2, \dots, x^n\}, \text{ where } x^i \in \mathbb{R}^d \text{ (} i = 1, 2, \dots, n\text{)}.$$

As a result of this partitioning and referring to a firstly given (later on in Section 5.2 optimized) number  $q \in \mathbb{N}$ , we come to  $q$  overlapping or disjoint subsets  $C_i$  ( $i = 1, 2, \dots, q$ ) satisfying

$$X = \bigcup_{i=1}^q C_i.$$

The sets  $C_i$  ( $i = 1, 2, \dots, q$ ) are called *clusters*. The clustering problem can be either a **hard clustering** problem or a **fuzzy clustering** one. A clustering problem is called a *hard* clustering problem if a data point belongs to only one cluster. In *fuzzy* clustering, there may exist two or more clusters including the same data point, i.e., the clusters may overlap. In this chapter, the **hard** clustering problem is considered rather than the fuzzy clustering, i.e.,

$$C_i \cap C_k = \emptyset \text{ if } i \neq k \text{ (} i, k = 1, 2, \dots, q\text{)}.$$

We note that there are *no* constraints on the clusters  $C_i$  ( $i = 1, 2, \dots, q$ ) and every point  $x \in X$  is contained in exactly one set  $C_i$ .

Considering the fact that each cluster  $C_i$  can be identified by its center denoted by  $a_i$ , the clustering problem can be reduced to the following optimization problem (see [10, 11, 60]), which is known as **minimum sum of squares clustering**:

$$(CLO) \quad \begin{cases} \text{minimize} & \varphi(C, a) = \frac{1}{n} \sum_{i=1}^q \sum_{x \in C_i} \|a^i - x\|^2, \\ \text{subject to} & C \in \bar{C}, \\ & C = \{C_1, C_2, \dots, C_q\}, a = (a^1, a^2, \dots, a^q)^T \in \mathbb{R}^{d \times q}. \end{cases}$$

Here,  $\|\cdot\|$  denotes the Euclidean norm,  $C$  is a set of clusters,  $\bar{C}$  is a set of

all possible  $q$ -partitions of the set  $X$ ,  $a^i$  is the center of the cluster  $C_i$  ( $i = 1, 2, \dots, q$ ). Let  $|C_i|$  denote a cardinality of the set of clusters. Then, the *centers* or *centroids*  $a^i$  can be represented by (cf. [6])

$$a^i = \frac{1}{|C_i|} \sum_{x \in C_i} x.$$

In general, instead of  $(\mathcal{CLO})$ , its rewritten form  $(\mathcal{RCLO})$  is used when direct application of mathematical programming techniques is considered:

$$(\mathcal{RCLO}) \left\{ \begin{array}{l} \text{minimize} \quad \psi(a, w) = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^q w_{ij} \|a^j - x^i\|^2, \\ \text{subject to} \quad w_{ij} \in \{0, 1\} \quad (i = 1, 2, \dots, n; j = 1, 2, \dots, q) \quad \text{and} \\ \sum_{j=1}^q w_{ij} = 1 \quad (i = 1, 2, \dots, n), \quad \text{where} \\ a = (a^1, a^2, \dots, a^q)^T \quad \text{and} \quad w = (w_{ij})_{i=1,2,\dots,n; j=1,2,\dots,q}. \end{array} \right.$$

We note that  $(\mathcal{RCLO})$  is a *mixed-integer* optimization problem. The centers can be rewritten as

$$a^j = \frac{\sum_{i=1}^n w_{ij} x^i}{\sum_{i=1}^n w_{ij}} \quad (j = 1, 2, \dots, q).$$

Here,  $w_{ij}$  is the association weight of the pattern  $x^i$  with cluster  $j$ , given by

$$w_{ij} = \begin{cases} 1, & \text{if pattern } i \text{ is allocated to cluster } j \quad (i = 1, 2, \dots, n; j = 1, 2, \dots, q), \\ 0, & \text{otherwise.} \end{cases} \quad (5.1.1)$$

We note that  $w$  is an  $n \times q$  matrix, while  $a$  is a  $d \times q$  matrix.

Furthermore,  $(\mathcal{RCLO})$  is a *global* optimization problem with possibly many local minima [6]. There are a number of algorithms in the literature that can be used for solving these kinds of problems like *dynamic programming* [60], *branch and bound* [23], *k-means* [1] algorithms. An overview of these algorithms can be

found in [32]. In general, solving the clustering problems with these algorithms is a task which takes significantly much time. This fact makes it necessary to develop clustering algorithms which will compute the local minimizers by making use of *optimization* techniques in the solution procedures. In [6], a clustering algorithm is suggested which is based on *non-smooth optimization* techniques.

## 5.2 Clustering via Non-Smooth Optimization

### 5.2.1 Algorithmic Framework

Before stating the algorithm, let us give a modeling and analysis of the non-smooth optimization approach to minimum sum of squares clustering.

The problems  $(\mathcal{CLO})$  and  $(\mathcal{RCLO})$  which we stated in the previous section, can be reformulated as a clustering problem in terms of unconstrained *non-smooth* and *non-convex* optimization as follows (see [7, 8, 10, 11]):

$$(\mathcal{CLP}) \quad \text{minimize } f(a) \quad (a = (a^1, a^2, \dots, a^q)^T \in \mathbb{R}^{d \times q}),$$

where

$$f(a) := \frac{1}{n} \sum_{i=1}^n \min_{j=1,2,\dots,q} \|a^i - x^i\|^2. \quad (5.2.2)$$

The intuitive clear equivalence between the problems  $(\mathcal{CLO})$ ,  $(\mathcal{RCLO})$  and  $(\mathcal{CLP})$  is also explicitly proved in [10]. The number of variables in  $(\mathcal{RCLO})$  is  $(n + d) \times q$ , whereas it is  $d \times q$  in  $(\mathcal{CLP})$ . This shows that the number of variables in  $(\mathcal{CLP})$  does *not* depend on the number of instances  $n$ . In fact, the number of instances  $n$  is greater than the number of attributes  $d$  in many real-world data bases.

As an advantage, we note that the non-smooth optimization problem  $(\mathcal{CLP})$  contains only the *continuous* real variables but not the integer coefficients  $w_{ij}$ , whereas the problem  $(\mathcal{RCLO})$  reveals both integer and continuous variables.

If the number of clusters is greater than one, i.e.,  $q > 1$ , then the objective function (5.2.2) in problem  $(\mathcal{CLP})$  is non-convex and non-smooth. We call a global optimization problem *large-scale* if the number  $q$  of clusters and the number  $d$  of attributes are large. In clustering analysis, it is important to choose a *meaningful number*  $q$  of clusters. We usually do not know how many clusters represent the considered set  $X$  a priori. This difficulty can be overcome by calculating clusters step-by-step in the optimization algorithm as discussed in the next section.

We also emphasize that the non-smooth form of the objective function in the problem  $(\mathcal{CLP})$  makes it possible to accelerate the calculation of the objective function significantly by reducing the number of records in a data set.

**Algorithm (Framework) 5.1.** This is an algorithm developed for solving cluster analysis problem  $(\mathcal{CCO})$  where, in addition, the minimization is also with respect to  $q \in \mathbb{N}$ .

**Step 1: Initialization**

Select a tolerance  $\epsilon > 0$ . Select an initial point  $a^0 = (a_1^0, a_2^0, \dots, a_d^0)^T \in \mathbb{R}^d$  and solve the minimization problem  $(\mathcal{CLP})$  with  $q = 1$ . Let  $a^{1*} \in \mathbb{R}^d$  be a solution to this problem and  $f^{1*}$  be the corresponding objective function value. Set  $k = 1$ .

**Step 2: Computation of the next cluster center**

Select a point  $y^0 \in \mathbb{R}^d$  and solve the following unconstrained (non-smooth) minimization problem:

$$(\mathcal{MP}) \quad \text{minimize } \bar{f}^k(y) \quad (y \in \mathbb{R}^d),$$

$$\text{where } \bar{f}^k(y) := \sum_{i=1}^n \min \{ \|a^{1*} - x^i\|^2, \|a^{2*} - x^i\|^2, \dots, \|a^{k*} - x^i\|^2, \|y - x^i\|^2 \}.$$

**Step 3: Refinement of all cluster centers**

Let  $\bar{y}^{k+1,*}$  be a solution to problem  $(\mathcal{MP})$ . Take  $a^{k+1,0} = (a^{1*}, a^{2*}, \dots, a^{k*}, \bar{y}^{k+1,*})^T$  as a new starting point and solve the following unconstrained (non-smooth) minimization problem:

$$(\mathcal{MP}2) \quad \text{minimize } f^{k+1}(a) \quad (a = (a^1, a^2, \dots, a^{k+1})^T \in \mathbb{R}^{d \times (k+1)}),$$

$$\text{where } f^{k+1}(a) := \sum_{i=1}^n \min_{j=1, \dots, k+1} \|a^j - x^i\|^2.$$

#### Step 4: *Stopping criterion*

Let  $a^{k+1,*}$  be a solution to the problem  $(\mathcal{MP}2)$  and  $f^{k+1,*}$  be the corresponding value of the objective function. If

$$\frac{f^{k*} - f^{k+1,*}}{f^{1*}} < \epsilon,$$

then stop, otherwise set  $k \leftarrow k + 1$  and go to Step 2.  $\square$

The important item which we should discuss is about the choice of the tolerance  $\epsilon > 0$ . Large values of  $\epsilon$  can result in the appearance of large clusters, whereas small values can produce small and artificial clusters. In order to explain this, we give an example.

**Example 5.2.** [6] Let us consider one artificial data set on  $\mathbb{R}^2$ . There are three isolated clusters in this data set given by the following formulas, respectively:

$$X^1 = \left\{ x^k \in \mathbb{R}^2 \mid x^k = (x_1^k, x_2^k), x_1^k = \frac{1}{2} |\sin(k)|, x_2^k = 2 + |\cos(k)|, k = 1, 2, \dots, 50 \right\},$$

$$X^2 = \left\{ x^k \in \mathbb{R}^2 \mid x^k = (x_1^k, x_2^k), x_1^k = \frac{1}{2} (1 + \sin(k)), x_2^k = |\cos(k)|, k = 1, 2, \dots, 50 \right\},$$

$$X^3 = \left\{ x^k \in \mathbb{R}^2 \mid x^k = (x_1^k, x_2^k), x_1^k = 2 + |\cos(k)|, x_2^k = \frac{3}{2} + \sin(k), k = 1, 2, \dots, 50 \right\}.$$

If  $\epsilon = 10^{-1}$ , then Algorithm (Framework) 5.1 with the appropriate subalgorithm for solving the subproblems exactly calculates these three clusters; if  $\epsilon = 10^{-2}$ , then this algorithm divides the third cluster into three clusters and leaves two other clusters unchanged. If  $\epsilon > 0$  is smaller, then we get a further division of these clusters. So, if  $\epsilon$  is small enough, we obtain some artificial clusters. The results of numerical experiments show that the best values for  $\epsilon$  are  $\epsilon \in [10^{-1}, 10^{-2}]$ .  $\square$



## 5.2.2 An Algorithm for Solving Optimization Problems in the Algorithmic Framework

Let us also explain an algorithm for solving non-smooth optimization problems ( $\mathcal{MP}$ ) and ( $\mathcal{MP2}$ ) in the clustering Algorithm 5.1. We state some definitions from non-smooth analysis for more easily following of the issue.

### 5.2.2.1 Some Elements of Non-smooth Analysis

**Definition 5.3.** Let  $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}$  be a given function. This function is said to be a *locally Lipschitz continuous* on  $\mathbb{R}^p$  if for any bounded subset  $S \in \mathbb{R}^p$  there exists a constant  $L > 0$  such that

$$|\Phi(y) - \Phi(u)| \leq L\|y - u\| \quad \forall y, u \in S. \quad \square$$

The function  $\Phi$  is differentiable almost everywhere [13] and one can define for it a *Clarke subdifferential* by

$$\partial\Phi(y) := \text{co} \left\{ v \in \mathbb{R}^p \mid \exists(y^k \in D(\Phi), y^k \rightarrow y (k \rightarrow +\infty)) : v = \lim_{k \rightarrow +\infty} \nabla\Phi(y^k) \right\}.$$

Here,  $D(\Phi) \subseteq \mathbb{R}^p$  denotes the domain set where  $\Phi$  is differentiable and “co” denotes the convex hull of a set. This set is nonempty [13].

Since the objective functions  $\bar{f}^k$  and  $f^k$  in problems ( $\mathcal{MP}$ ) and ( $\mathcal{MP2}$ ), respectively, are represented as a sum of minima of norms, they are locally Lipschitz continuous.

### Definition 5.4. (Directional Derivative)

A given function  $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}$  is called *differentiable* at the point  $y \in \mathbb{R}^p$  *with respect* to the direction  $g \in \mathbb{R}^p$  if the limit

$$\Phi'(y, g) := \lim_{\alpha \rightarrow +0} \frac{\Phi(y + \alpha g) - \Phi(y)}{\alpha}$$

exists. In case of this directional differentiability of  $\Phi$  at  $y$ , the number  $\Phi'(y, g)$

is said to be the *derivative* of the function  $\Phi$  *with respect to the direction*  $g \in \mathbb{R}^p$  at the point  $y$ . □

**Definition 5.5. (Clarke Upper Derivative)**

The *Clarke upper derivative*  $\Phi^0(y, g)$  of a given function  $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}$  at the point  $y$  *with respect to the direction*  $g \in \mathbb{R}^p$  is defined as follows:

$$\Phi^0(y, g) = \limsup_{\alpha \rightarrow +0, u \rightarrow y} \frac{\Phi(u + \alpha g) - \Phi(u)}{\alpha}. \quad \square$$

It should be noted that the Clarke upper derivative always exists for locally Lipschitz functions.

**Definition 5.6. (Clarke Regularity)**

A given function  $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}$  is said to be *Clarke regular* at the point  $y \in \mathbb{R}^p$  if

$$\Phi'(y, g) = \Phi^0(y, g) \quad \text{for all } g \in \mathbb{R}^p. \quad \square$$

**5.2.2.2 Consequence for the Clustering Problem**

For Clarke regular functions there exists a full calculus (see, e.g., [10, 11]). However, in general, for *non-regular* functions such a calculus does *not* exist.

Functions represented as a minimum of norms such as arising in our classification problems, are *not* Clarke regular. Since the sum of non-regular functions is not regular, our functions  $\bar{f}^k$  and  $f^k$  in general are *not* regular. Therefore, subgradients of these functions cannot be calculated using subgradients of their terms. We can conclude that the calculation of the subgradients of the functions  $\bar{f}^k$  and  $f^k$  is a very difficult task and, hence, the application of methods of non-smooth optimization requiring a subgradient evaluation at each iteration, can *not* be effective. Therefore, we use the so-called ***discrete gradient method*** to solve the problems  $(\mathcal{MP})$  and  $(\mathcal{MP}2)$  in which the subgradients of the objective function are *replaced* by its discrete gradients. A detailed description of this method can be found in [4].

The discrete gradient method uses values of the objective function only. It should be noted that the calculation of the objective functions in the problems  $(\mathcal{MP})$  and  $(\mathcal{MP}2)$  can be expensive when the number of instances in the data set is large. We will show that the use of the discrete gradient method allows one to significantly reduce the number of objective function evaluations.

## 5.3 Discrete Gradient Method

In this subsection, we will briefly introduce the discrete gradient method. We start with the definition of the discrete gradient. Here, we follow the authors [6] along a refined way of approximating or substituting a given or not given gradient (directional derivative) by difference quotients.

### 5.3.1 Definition of Discrete Gradient

Let  $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}$  be a locally Lipschitz continuous function defined on  $\mathbb{R}^p$ . Let

$$S_1 := \{g \in \mathbb{R}^p \mid \|g\| = 1\},$$

$$G := \{e \in \mathbb{R}^p \mid e = (e_1, e_2, \dots, e_p)^T, |e_j| = 1, j = 1, 2, \dots, p\},$$

$$I(g, \alpha) := \{i \in \{1, 2, \dots, p\} \mid |g_i| \geq \alpha\}, \quad \text{and}$$

$$P := \{z : (0, \infty) \rightarrow \mathbb{R} \mid z(\lambda) \in \mathbb{R}, z(\lambda) > 0 (\lambda > 0), \lambda^{-1}z(\lambda) \rightarrow 0, (\lambda \rightarrow 0)\},$$

where  $\alpha \in (0, p^{-1/2}]$  is a fixed number.

Here,  $S_1$  is the unit sphere,  $G$  is the set of vertices of the unit hypercube in  $\mathbb{R}^p$ , and  $P$  is the set of so-called *univariate positive infinitesimal* functions.

We define operators  $H_i^j : \mathbb{R}^p \rightarrow \mathbb{R}^p$  ( $i = 1, 2, \dots, p$ ;  $j = 0, 1, \dots, p$ ) by the

formula

$$H_i^j g := \begin{cases} (g_1, g_2, \dots, g_j, 0, \dots, 0), & \text{if } j < i, \\ (g_1, g_2, \dots, g_{i-1}, 0, g_{i+1}, \dots, g_j, 0, \dots, 0), & \text{if } j \geq i. \end{cases} \quad (5.3.3)$$

We can see that

$$H_i^j g - H_i^{j-1} g = \begin{cases} (0, \dots, 0, g_j, 0, \dots, 0), & \text{if } j = 1, 2, \dots, p, \quad j \neq i, \\ 0 & \text{if } j = i. \end{cases} \quad (5.3.4)$$

For any  $e := (e_1, e_2, \dots, e_p)^T \in G$  we put:  $e(\beta) := (\beta e_1, \beta^2 e_2, \dots, \beta^p e_p)^T$ , where  $\beta \in (0, 1]$ . For  $y \in \mathbb{R}^p$ , we consider vectors

$$y_i^j := y_i^j(g, e, z, \lambda, \beta) = y + \lambda g - z(\lambda) H_i^j e(\beta) \quad (i = 1, 2, \dots, p; j = 1, 2, \dots, p).$$

From (5.3.4), it follows that

$$y_i^{j-1} - y_i^j = \begin{cases} (0, \dots, 0, z(\lambda), e_j(\beta), 0, \dots, 0), & \text{if } j = 1, 2, \dots, p, \quad j \neq i, \\ 0 & \text{if } j = i. \end{cases} \quad (5.3.5)$$

It is clear that  $H_i^0 g = 0$  and  $y_i^0(g, e, z, \lambda, \beta) = y + \lambda g$  for all  $i \in I(g, \alpha)$ .

**Definition 5.7.** The *discrete gradient* of a given function  $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}$  at some point  $x \in \mathbb{R}^p$  and any  $i \in I(g, \alpha)$  is the vector

$$\Gamma^i(x, g, e, z, \lambda, \beta) = (\Gamma_1^i, \Gamma_2^i, \dots, \Gamma_p^i)^T \in \mathbb{R}^p \quad (g \in S_1)$$

with the following coordinates of difference quotient type for  $j = 1, 2, \dots, p$ :

$$\Gamma_j^i := (z(\lambda) e_j(\beta))^{-1} (\Phi(y_i^{j-1}(g, e, z, \lambda, \beta)) - \Phi(y_i^j(g, e, z, \lambda, \beta))) \quad (j \neq i),$$

$$\Gamma_i^i = (\lambda g_i)^{-1} \left( \Phi(y_i^p(g, e, z, \lambda, \beta)) - \Phi(y) - \sum_{j=1, j \neq i}^p \Gamma_j^i (\lambda g_j - z(\lambda) e_j(\beta)) \right).$$

□

Now, let us state the discrete gradient method with the sequences  $(\lambda_k)_{k \in \mathbb{N}_0}$ ,  $(z_k)_{k \in \mathbb{N}_0}$  and  $(\beta_k)_{k \in \mathbb{N}_0}$ , where  $\delta_k > 0$ ,  $z_k \in P$ ,  $\lambda_k > 0$ ,  $\beta_k \in (0, 1]$  ( $k \in \mathbb{N}_0$ ),  $\delta_k \rightarrow 0^+$ ,  $z_k \rightarrow 0^+$ ,  $\lambda_k \rightarrow 0^+$ ,  $\beta_k \rightarrow 0^+$  ( $k \rightarrow +\infty$ ), and let numbers  $c_1 \in (0, 1)$ ,  $c_2 \in (0, c_1]$  be given.

### 5.3.2 The Algorithm

**Algorithm 5.8.** The discrete gradient method is working as follows:

**Step 1:** Choose any starting point  $y^0 \in \mathbb{R}^p$  and set  $k = 0$ .

**Step 2:** Set  $s = 0$  and  $y_s^k = y^k$ .

**Step 3:** Apply the algorithm stated in [6] for the calculation of the descent direction at  $y = y_s^k$ ,  $\delta = \delta_k$ ,  $z = z_k$ ,  $\lambda = \lambda_k$ ,  $\beta = \beta_k$ ,  $c = c_1$ . This algorithm terminates after a finite number of iterations  $l > 0$ . As a result, we get the set  $\bar{D}_l(y_s^k)$  which is the topological closure of  $D_l(y_s^k)$ , and an element  $v_s^k$  such that

$$\|v_s^k\| = \min\{\|v\| \mid v \in \bar{D}_l(y_s^k)\}.$$

Furthermore, either  $\|v_s^k\| \leq \delta_k$  or for the search direction  $g_s^k := -\|v_s^k\|^{-1}v_s^k$  it holds

$$\Phi(y_s^k + \lambda_k g_s^k) - \Phi(y_s^k) \leq -c_1 \lambda_k \|v_s^k\|.$$

**Step 4:** If  $\|v_s^k\| \leq \delta_k$ , then set  $y^{k+1} := y_s^k$ ,  $k \leftarrow k + 1$  and go to Step 2. Otherwise, go to Step 5.

**Step 5:** Construct the following iteration  $y_{s+1}^k := y_s^k + \sigma_s g_s^k$ , where  $\sigma_s$  is defined as follows:

$$\sigma_s := \arg \max\{\sigma \geq 0 \mid \Phi(y_s^k + \sigma g_s^k) - \Phi(y_s^k) \leq -c_2 \sigma \|v_s^k\|\}.$$

**Step 6:** Set  $s \leftarrow s + 1$  and go to Step 3. □

## 5.4 Numerical Results

The numerical experiments presented in this section are carried out on a Pentium-4, 1.7 GHz, PC to verify the effectiveness of the clustering algorithm [6]. The algorithms which are compared on three standard problems are *k-means* (K-M) [1], *j-means* (J-M+) [33], *variable neighborhood search* (VNS) [34], *global k-means* (G1) [43], *fast global k-means* (G2) [35], *tabu search* (TS) [29], *genetic algorithm* (GA) [58] and *simulated annealing* (SA) [39].

Additionally, the results obtained using K-M, TS, GA and SA and presented in [61], as well as, the results obtained using K-M, J-M+, VNS, G1, G2 and presented in [33, 35] for comparison. We note that TS, GA and SA have been applied to problem (*RCLO*) which is equivalent to the *non-smooth* optimization formulation of a clustering problem (*CLP*).

In the tables below, the best known value for the global minimum is reported for some selected problems. These values are given as  $nf(x^*)$  where  $n$  is the number of instances and  $x^*$  is a local minimizer. For each algorithm we define the *relative error*  $E$  by

$$E := \frac{(\bar{f} - f_{\text{opt}})}{f_{\text{opt}}} \cdot 100.$$

Here,  $\bar{f}$  and  $f_{\text{opt}}$  denote the solution found by some considered algorithm and the best known solution, respectively. In the forthcoming tables, please find the corresponding values for  $E$ .

The average results for 10 restarts from [33, 35] for K-M, J-M+ and VNS are used. However, for the clustering Algorithm (Framework) 5.1, one result is presented since the starting points are updated by the algorithm itself.

The three *clustering test problems* used for comparing the Algorithm (Framework) 5.1 with K-M, TS, GA and SA are as follows [60]:

1. The first set is known as German towns, which uses the Cartesian coordinates of 59 towns and has 59 records with 2 attributes.
2. The second set contains 89 postal zones in Bavaria (Germany) and their

names. It contains 89 records with 3 attributes.

3. The third set consists of the above Bavarian postal zones but with four attributes; these are the number of self-employed people, of civil servants, clerks and manual workers. The number of instances is again 89.

Results of the numerical experiments are presented in Tables 5.1-5.3.

$q$	best known value $nf(x^*)$	K-M	TS	GA	SA	Alg. 5.1
		$E$	$E$	$E$	$E$	$E$
2	$0.121426 \cdot 10^6$	0.00	0.00	0.00	0.00	0.00
3	$0.77009 \cdot 10^5$	1.45	0.00	0.00	0.29	0.29
4	$0.49601 \cdot 10^5$	0.55	0.00	0.00	0.00	0.00
5	$0.39453 \cdot 10^5$	2.75	0.00	0.15	0.15	0.15

Table 5.1: Results for German towns data base (relative errors) [6].

From Table 5.1, we see that Algorithm (Framework) 5.1 achieves better results than  $k$ -means for all number of clusters and has similar results with simulated annealing. Although the results of tabu search and Algorithm (Framework) 5.1 are close, tabu search is better for three and five clusters. The genetic algorithm is slightly better than Algorithm 5.1 for three clusters. The solutions obtained by the Algorithm 5.1 are very similar and close to the solutions of global optimization techniques. It can be derived that this algorithm can be used to calculate the *deep* local minima of the objective function in a clustering problem [6]. By a *deep local minimizer* we understand a local solution with its objective value being relatively close to the value of the global solution and, herewith, locally the best clearly.

$q$	best known value $nf(x^*)$	K-M	TS	GA	SA	Alg. 5.1
		$E$	$E$	$E$	$E$	$E$
2	$0.60255 \cdot 10^{12}$	7.75	0.00	0.00	0.00	0.00
3	$0.29451 \cdot 10^{12}$	23.48	23.48	23.48	23.48	0.00
4	$0.10447 \cdot 10^{12}$	166.88	18.14	0.00	0.39	0.00
5	$0.59762 \cdot 10^{12}$	335.32	33.35	0.00	40.32	0.00

Table 5.2: Results for the first Bavarian postal zones data set (relative errors) [6].

From Table 5.2 we can see that Algorithm 5.1 again gives better results than the  $k$ -means algorithm. Algorithm 5.1 achieves better results than TS and SA for all values of  $q$ , except  $q = 2$  and a better result than GA for  $q = 3$  [6].

$q$	best known value $nf(x^*)$	K-M	TS	GA	SA	Alg. 5.1
		$E$	$E$	$E$	$E$	$E$
2	$0.199080 \cdot 10^{11}$	144.25	0.00	144.25	144.25	144.25
3	$0.173987 \cdot 10^{11}$	106.79	0.00	0.00	77.77	0.00
4	$0.755908 \cdot 10^{10}$	303.67	0.00	0.00	9.13	0.00
5	$0.540379 \cdot 10^{10}$	446.13	15.76	15.76	18.72	0.00

Table 5.3: Results for the second Bavarian postal zones data set (relative errors) [6].

The results presented in Table 5.3 show that for this data set, Algorithm (Framework) 5.1 achieves better results than the  $k$ -means algorithm and SA. The result obtained by our Algorithm 5.1 for  $q = 5$  is better than those obtained by TS and GA. However, TS is best for  $q = 2$  [6]. From Tables 5.1-5.3, we infer that at least for these three data sets, Algorithm (Framework) 5.1 works better than the  $k$ -means algorithm and achieves close, similar and sometimes better results than tabu search, genetic and simulated annealing algorithms. These results confirm that the proposed algorithm, i.e., *Algorithm 5.1*, may be used for finding a global minimum of the objective function of a clustering problem since it finds deep local minima successfully as seen in Tables 5.1-5.3 [6].

For the comparison of the proposed algorithm with K-M, J-M+, VNS, G1 and G2 we have used the following *data sets*:

1. the first Bavarian postal zones data set;
2. Fishers iris data set: it contains 150 instances and 4 attributes [16];
3. 2-dimensional data on 1060 points in the plane from a traveling salesman problem of [34] (TSPLIB;  $n = 1060$ );
4. 2-dimensional data on 3038 points in the plane from a traveling salesman problem of [34] (TSPLIB;  $n = 3038$ ).



$q$	$f_{opt}$	K-M		J-M+		VNS		VNS-1	G1	G2	Algorithm 5.1	
		$E_{mean}$	$E_{best}$	$E_{mean}$	$E_{best}$	$E_{mean}$	$E_{best}$	$E$	$E$	$E$	$E$	time
2	$0.60255 \cdot 10^{12}$	7.75	7.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
3	$0.29451 \cdot 10^{12}$	23.40	20.02	0.00	0.00	7.04	0.00	0.00	0.00	0.00	0.00	0.12
4	$0.10447 \cdot 10^{12}$	156.17	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.30
5	$0.59762 \cdot 10^{11}$	315.28	23.58	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54
6	$0.35909 \cdot 10^{11}$	531.44	27.79	27.70	27.65	11.06	0.00	0.00	0.00	0.00	0.00	0.71
7	$0.21983 \cdot 10^{11}$	832.60	69.39	44.00	0.00	7.07	0.00	0.00	0.00	0.00	1.48	1.06
8	$0.13385 \cdot 10^{11}$	1239.64	0.00	0.24	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.61
9	$0.84237 \cdot 10^{10}$	1697.17	35.81	28.59	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.52
10	$0.64465 \cdot 10^{10}$	1638.30	57.81	0.16	0.00	0.00	0.00	0.00	0.00	0.00	11.32	3.61
14	$0.21155 \cdot 10^{10}$	1922.39	67.10	11.48	0.00	0.00	0.00	0.00	1.48	0.11	0.00	7.86
18	$0.98069 \cdot 10^9$	2703.85	244.13	5.62	0.00	1.11	0.00	0.00	0.00	0.00	0.00	13.65
22	$0.54214 \cdot 10^9$	4735.31	228.89	18.55	4.71	5.56	0.00	0.00	0.00	0.00	0.74	20.26
26	$0.28223 \cdot 10^9$	8835.90	105.62	6.19	0.00	0.00	0.00	0.00	6.44	9.14	1.96	28.01
30	$0.17138 \cdot 10^9$	14032.88	171.98	8.17	0.00	0.53	0.00	0.00	0.00	3.80	0.01	34.48

Table 5.4: Results for the first Bavarian postal zones data set (relative errors, running time) [6].

The results presented in Table 5.4 demonstrate that for the first Bavarian postal zones data set Algorithm 5.1 performs better than K-M and J-M+, and slightly better than VNS. The results obtained by Algorithm 5.1 and G2 are almost similar. G1 performs better than Algorithm 5.1 for  $q = 7, 10, 22$ . The deviation of the results obtained by Algorithm 5.1 from the global minimum is zero or very small for all  $q$  except  $q = 10$  [6].

$q$	$f_{opt}$	K-M		J-M+		VNS		VNS-1	G1	G2	Algorithm 5.1	
		$E_{mean}$	$E_{best}$	$E_{mean}$	$E_{best}$	$E_{mean}$	$E_{best}$	$E$	$E$	$E$	$E$	time
2	152.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
3	78.851	13.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.19
4	57.228	11.26	0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.56
5	46.446	13.83	0.00	3.97	0.00	1.46	0.00	0.00	0.00	0.06	0.00	0.83
6	39.040	16.21	0.00	4.26	0.00	0.01	0.00	0.00	0.00	0.07	0.00	1.09
7	34.298	17.43	0.00	2.86	0.00	0.00	0.00	0.00	0.02	1.55	1.34	2.47
8	29.989	20.81	0.00	2.76	0.00	0.02	0.00	0.00	0.01	0.25	0.00	3.25
9	27.786	18.80	0.00	1.62	0.00	0.00	0.00	0.00	0.01	0.82	1.37	4.72
10	25.834	17.43	0.16	2.84	0.00	0.42	0.00	0.00	0.51	1.00	0.00	5.91

Table 5.5: Results for the Fishers iris data set (relative errors, running time) [6].

The results for the Fishers iris data set presented in Table 5.5 show that for this data set, Algorithm 5.1 gives better results than K-M and J-M+, and slightly better results than G2. The results by VNS and Algorithm 5.1 are quite similar. G1 gives better or similar results for all values of  $q$  except  $q = 10$ . We can see that the deviation of the results from the global minimum obtained by Algorithm 5.1 is zero or small for all  $q$  [6].

$q$	$f_{opt}$	K-M	J-M+	VNS	Algorithm 5.1	
		$E$	$E$	$E$	$E$	time
10	$0.17548 \cdot 10^{10}$	0.03	0.19	0.04	0.13	5.49
20	$0.79179 \cdot 10^9$	3.96	0.04	0.83	0.72	81.11
30	$0.48125 \cdot 10^9$	10.51	1.82	0.42	0.56	189.40
50	$0.25551 \cdot 10^9$	16.58	3.84	1.70	0.98	534.84

Table 5.6: Results for the TSPLIB ( $n=1060$ ) data set (relative errors, running time) [6].

From Table 5.6, we learn that the results by J-M+, VNS and Algorithm (Framework) 5.1 for the TSPLIB ( $n = 1060$ ) data set are comparable and these algorithms perform better than K-M. Again, Algorithm 5.1 produces results which are very close to best known ones. For the TSPLIB ( $n = 3068$ ) data set the results by Algorithm 5.1 are better than those for K-M and J-M+ and comparable with those for VNS and G2. We can conclude that *Algorithm 5.1 reaches the best known solution in many cases and gives results which are close to the best ones in all other cases* [6].

The results presented in Tables 5.1-5.7 show us that by using Algorithm (Framework) 5.1 one can calculate the global minimizer or best known solution of the minimum sum of squares clustering problem in 37 cases out of 55 (67%). Moreover, we can calculate a solution which is close to the best one in all other cases. Although Algorithm 5.1 allows us to find at least a deep local minimizer in many minimum sum of squares clustering problem, there is no guarantee that we can always find one with this local method.

Algorithm (Framework) 5.1 requires much more *computational time* than K-M and J-M+. However, it requires significantly less computational time than G1

$q$	$f_{opt}$	K-M		J-M+		VNS		VNS-1	G1	G2	Algorithm 5.1	
		$E_{mean}$	$E_{best}$	$E_{mean}$	$E_{best}$	$E_{mean}$	$E_{best}$	$E$	$E$	$E$	$E$	time
2	$0.31688 \cdot 10^{10}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
3	$0.21763 \cdot 10^{10}$	1.55	0.00	1.29	0.00	1.37	0.00	0.00	0.00	0.00	0.00	0.26
4	$0.14790 \cdot 10^{10}$	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.46
5	$0.11982 \cdot 10^{10}$	0.12	0.00	0.11	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.76
6	$0.96918 \cdot 10^9$	1.22	0.00	1.98	0.00	0.01	0.00	0.00	0.00	0.02	0.00	1.25
7	$0.83966 \cdot 10^9$	1.65	0.00	1.48	0.00	0.73	0.00	0.00	0.00	0.84	0.00	1.79
8	$0.73475 \cdot 10^9$	1.90	0.00	1.48	0.00	0.62	0.00	0.00	0.00	1.28	1.71	2.57
9	$0.64477 \cdot 10^9$	1.47	0.00	0.99	0.01	0.11	0.00	0.00	0.00	1.41	0.00	4.10
10	$0.56025 \cdot 10^9$	2.44	0.00	1.81	0.00	0.06	0.00	0.00	0.00	0.00	0.00	5.85
20	$0.26681 \cdot 10^9$	3.16	0.02	2.60	0.05	0.09	0.01	0.00	0.01	0.42	0.45	61.65
30	$0.17557 \cdot 10^9$	4.04	0.60	2.89	0.42	0.91	0.01	0.00	0.00	1.48	0.82	255.17
40	$0.12548 \cdot 10^9$	6.21	0.67	3.49	0.00	0.93	0.00	0.00	0.42	1.42	0.70	586.13
50	$0.98400 \cdot 10^8$	6.79	0.79	3.51	0.74	0.33	0.00	0.00	0.07	1.18	1.04	1062.10

Table 5.7: Results for the TSBLIB ( $n=3038$ ) zones data set (relative errors, running time) [6].

algorithm. The ability of Algorithm (Framework) 5.1 to solve minimum sum of squares clustering problems in large data sets is also limited.

The numerical experiments presented in this chapter show that Algorithm (Framework) 5.1 is able to solve clustering problems in data sets with  $n \times d \leq 5 \cdot 10^5$  and  $d \leq 1000$  for a reasonable time *without* using any complexity reduction procedures!

**Remark 5.9.** There are many further fields where clustering theory and, here-with, the explanations of this chapter will more and more become fruitful. Among of them, we mention computational biology and bioinformatics (see, e.g., [28]) and financial mathematics (see, e.g., [42]). In the first one of these areas, clustering is used in *gene expression* data analysis and molecular mechanisms of heavy metal response in *P. chrysosporium* [49]. In the second one, customers of *loan banks* are clustered and prototypes of contracts prepared by loan banks in this way [40, 41]. Here, a new approach consists in modeling the payments by *hidden Markov models* (HMM) and classifying them by clusters [41]. □

## 5.5 Concluding Remark

In this chapter, we learned about how valuable an enrichment non-smooth methods are for derivative free optimization and its practical utilization. We saw that in the emerging area of data mining with its various clustering problems, non-smooth optimization means a precious alternative and promises worthwhile future applications in science, technology, social and public live.  $\square$



# CHAPTER 6

## CONCLUSION

Mathematical programming in the absence of derivatives or, what is even more worse, in the presence of finitely many values of  $f$  only, is one of the greatest and most challenging problems in modern applied mathematics. This thesis tried to give a valuable overview, a structuring of given approaches, new contributions and views. These new views come, e.g., from mathematical data evaluation theory, i.e., statistical learning, inverse problems and data mining. A special contribution consists in the author's participation in the search on stirrer configuration.

Two implementations of derivative free methods have been used in this thesis. The first one we made for our derivative free method, applied to a stirrer configuration problem from chemical engineering, manufacturing or ecology. We used DFO, a FORTRAN package developed for solving general nonlinear optimization problems. We optimized the power number and the Newton number for standard stirrer configuration when the Reynolds number is 100.

The second implementation we did is for the derivative free clustering algorithm in combination with the non-smooth optimization methods. We have presented the results of this derivative free algorithm and compared them with various algorithms from the literature. We have confirmed and recommended that this clustering algorithm can be used for finding a global minimum of the objective function of a clustering problem. It is able to very well approximate the best known solution in many cases and to give results which are close to the best ones in all other cases.

We are looking forward for worthwhile and important applications of derivative

free optimization, serving for overcoming various future challenges of modern civilization and life.



# APPENDIX A

## SOME NOTATION

Let an open set  $U \subseteq \mathbb{R}^n$  and a number  $k \in \mathbb{N}_0 \cup \{\infty\}$  be given [68]. For a  $k$ -times continuously differentiable, i.e.,  $C^k$ -function  $f : U \rightarrow \mathbb{R}$  we denote the partial derivative of  $r^{\text{th}}$  order ( $r \leq k$ ) at some point  $x \in U$  by

$$\frac{\partial^r f}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \dots \partial x_n^{\alpha_n}}(x) \quad (\alpha_1, \alpha_2, \dots, \alpha_n \in \mathbb{N}); \quad \sum_{j=1}^n \alpha_j = r.$$

The sequence of differentiation can be changed arbitrarily - also in notation. (Sometimes the whole differentiation operator stands in front of  $f$ .) In the case  $r = 0$ , this number is  $f(x)$  itself. The row vector of first partial derivatives  $f_x(x) := \frac{\partial f}{\partial x}(x)$ ,

$$\left( \frac{\partial f}{\partial x_j}(x) \right)_{j \in \{1, 2, \dots, n\}}$$

is denoted by  $\nabla f(x) := Df(x)$ , while  $D^T f(x)$  stands for the transposed column vector ( $k \geq 1$ ).

Let  $\nabla^2 f(x) := D^2 f(x)$  represent the matrix of second partial derivatives, called the *Hessian* ( $k \geq 2$ ):

$$\left( \frac{\partial^2 f}{\partial x_i \partial x_j}(x) \right)_{i \in \{1, 2, \dots, n\}; j \in \{1, 2, \dots, n\}}$$

In case  $n = 1$ , we use symbols  $\frac{df}{dx}(x)$ ,  $\frac{d^2 f}{dx^2}(x)$  and replace  $x$  by  $t$  sometimes.

# APPENDIX B

## A SUBROUTINE OF DFO FOR STIRRER CONFIGURATION

The following is the subroutine which we use in the optimization process for finding the Newton number and power number. The values of power number and Newton number are inputted into DFO as initial values with the subroutine *FUN*. Then, these values are optimized by DFO.

```
SUBROUTINE FUN(N, X, F, IFERR)
DOUBLE PRECISION X(N), F
LOGICAL IFERR, ex
INTEGER N, status
open (20, File='a.dat')
do i = 1, N
    write(20, *) X(i)
end do
close(20)
100 inquire (File='b.dat', Exist=ex)
if (ex) then
    open(95, File='b.dat')
    read(95, *) F
    close(95)
    status=unlink ('b.dat')
    IFERR = .false.
else
    pause 'testing...'
    goto 100
endif
```



## REFERENCES

- [1] D. Akume and G.-W. Weber, Cluster algorithms: Theory and Methods, *Journal of Computational Technologies* 7, 1 (2002) 15-27.
- [2] R.C. Aster, B. Borchers and C. Thurber, *Parameter Estimation and Inverse Problems*, Academic Press, 2004.
- [3] M. Awad and L. Khan, Applications and Limitations of Support Vector Machines, ed. John Wang to appear in *Encyclopedia of Data Warehousing and Mining* by Information Science Publishing, 2004.
- [4] A.M. Bagirov, Minimization methods for one class of nonsmooth functions and calculation of semi-equilibrium prices, in: *Progress in Optimization: Contribution from Australasia*, A. Eberhard et. al. (eds.), Kluwer Academic Publishers (1999) 147-175.
- [5] A.M. Bagirov, A method for minimization of quasidifferentiable functions, *Optimization Methods and Software* 17, 1 (2002) 31-60.
- [6] A.M. Bagirov and J. Yearwood, A new nonsmooth optimization algorithm for minimum sum-of-squares clustering problem, Centre for Informatics and Applied Optimization, School of Information Technology and Mathematical Sciences, University of Ballarat, Victoria 3353, Australia.
- [7] A.M. Bagirov, A.M. Rubinov and J. Yearwood, Using global optimization to improve classification for medical diagnosis and prognosis, *Topics in Health Information Management* 22 (2001) 65-74.
- [8] A.M. Bagirov, A.M. Rubinov and J. Yearwood, Global optimization approach to classification, *Optimization and Engineering* 22 (2001) 65-74.

- [9] R. Bates, P. Fondy and R. Corpstein, An examination of some geometrical parameters of impeller power, *Ind. Eng. Chem. Proc. Des. Dev.* 2 (1963) 310-314.
- [10] H.H. Bock, *Automatische Klassifikation*, Vandenhoeck and Ruprecht, Göttingen, 1974.
- [11] H.H. Bock, Clustering and neural networks, in: *Advances in Data Science and Classification*, A. Rizzi, M. Vichi and H.H. Bock (eds.), Springer-Verlag, Berlin, 1998, 265-277.
- [12] F. Cazals and M. Pouget, Estimating differential quantities using polynomial fitting of osculating jets, Eurographics Symposium on Geometry Processing, 2003, [http://graphics.stanford.edu/courses/cs468-03-fall/Papers/cazals\\_jets.pdf](http://graphics.stanford.edu/courses/cs468-03-fall/Papers/cazals_jets.pdf).
- [13] F.H. Clarke, *Optimization and Nonsmooth Analysis*, Wiley, New York, 1983.
- [14] A.R. Conn, K Scheinberg and Ph.L. Toint, On the convergence of derivative-free methods for unconstrained optimization, in: *Approximation Theory and Optimization: Tributes to M.J.D. Powell*, A. Iserles and M. Buhmann (eds.), Cambridge University Press, Cambridge, UK, 1997, 83-108.
- [15] A.R. Conn and Ph.L. Toint, An algorithm using quadratic interpolation for unconstrained derivative free optimization, in: *Nonlinear Optimization and Applications*, G. Di Pillo and F. Giannessi (eds.), New York, Plenum Publishing (1996) 27-47.
- [16] A.R. Conn, K. Scheinberg and Ph.L. Toint, Recent progress in unconstrained nonlinear optimization without derivatives, *Mathematical Programming* 79 (1997) 397-414.
- [17] A.R. Conn, K Scheinberg and Ph.L. Toint, Derivative free optimization algorithm for constrained problems, preprint, IBM T.J. WATSON Research Center, 1999.

- [18] A.R. Conn, K. Scheinberg and L.N. Vicente, Geometry of sample sets in derivative free optimization part I: polynomial interpolation, preprint, Departamento de Matemática, Universidade de Coimbra, April 2003, revised: September 2004.
- [19] A.R. Conn, K. Scheinberg and P.L. Toint, A derivative free optimization method via support vector machines, 1999.
- [20] A.R. Conn, N.I.M. Gould and P.L. Toint, *Trust-Region Methods*, MPS-SIAM Series on Optimization, SIAM, Philadelphia, 2000.
- [21] D. Çekmecilioglu, Feedstock optimization of in-vessel food waste composting systems for inactivation of pathogenetic microorganisms, Seminar on Computational Biology and Medicine, Institute of Applied Mathematics, 2005, [http://www.fde.metu.edu.tr/personal\\_sites/deniz/](http://www.fde.metu.edu.tr/personal_sites/deniz/).
- [22] J.E. Dennis and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
- [23] G. Diehr, Evaluation of a branch and bound algorithm for clustering, SIAM J. Scientific and Statistical Computing 6 (1985) 268-284.
- [24] M. Dunham, *Data Mining Introductory and Advanced Topics*, ISBN: 0130888923, Prentice Hall, 2003.
- [25] L. Dupont, D. Lazard, S. Lazard and S. Petitjean, Near-optimal parameterization of the intersection of quadrics, presentation, San Diego, 2003, <http://www.loria.fr/~dupont/files/sandiego2003.pdf>.
- [26] M.J. Greenberg, *Euclidean and Non-Euclidean Geometries* (3rd ed.), W.H. Freeman, New York, 1993.
- [27] L.E. Eriksson, Practical three-dimensional mesh generation using transfinite interpolation, SIAM J. Sci. Stat. Comput. 6, 3 1985, 712-741.
- [28] W.J. Ewens and G.R. Grant, *Statistical Methods in Bioinformatics: An Introduction*, Springer-Verlag, New York, 2001.

- [29] F.W. Glover and M. Laguna, *Tabu Search*, Springer, 1997.
- [30] A. Griewank, Computational differentiation and optimization, in: *Mathematical Programming: State of Art*, J.R. Birge and K.G. Murty (eds.), The University of Michigan, Ann Arbor, MI (1994) 102-131.
- [31] A. Griewank and G. Corliss, *Automatic Differentiation Algorithms*, SIAM, Philadelphia, PA, 1991.
- [32] P. Hansen and B. Jaumard, Cluster analysis and mathematical programming, *Mathematical Programming*, 79 (1-3) (1997) 191-215.
- [33] P. Hansen and N. Mladenovic, J-means: a new heuristic for minimum sum-of-squares clustering, *Pattern Recognition*, 4 (2001) 405-413.
- [34] P. Hansen and N. Mladenovic, Variable neighborhood decomposition search, *Journal of Heuristics* 7 (2001) 335-350.
- [35] P. Hansen, E. Ngai, B.K. Cheung and N. Mladenovic, Analysis of global k-means, an incremental heuristic for minimum sum-of-squares clustering, discussion papers, Group for Research in Decision Analysis 2002.
- [36] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer, New York, 2001.
- [37] M. Heath, *Scientific Computing: An Introductory Survey*, McGraw-Hill, Boston, 2002.
- [38] B. Karasözen and G. W. Weber, *Numerical Optimization*, lecture notes, Institute of Applied Mathematics, METU, Ankara, Spring 2003.
- [39] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, *Science* 220 (1983).
- [40] B. Knab, R. Schrader, I. Weber, K. Weinbrecht and B. Wichern, Ein Mesoskopisches Simulationsmodell zur Kollektivfortschreibung, preprint, ZAIK, University of Cologne (1997).

- [41] B. Knab, A. Schliep, B. Steckemetz and B. Wichern, Model-based clustering with Hidden Markov Models and its application to financial time-series data, preprint, ZAIK, University of Cologne (2002).
- [42] D. Lamberton and B. Lapeyre, *Introduction to Stochastic Calculus Applied to Finance*, Chapman&Hall, 1996.
- [43] A. Likas, M. Vlassis and J. Verbeek, The global k-means clustering algorithm, *Pattern Recognition*, 36 (2003) 451-461.
- [44] C. Lin, Support vector machines for data classification, talk at Centrum voor Wiskunde en Informatica, 2004, <http://www.csie.ntu.edu.tw/~cjlin/talks/cwi.pdf>.
- [45] R.J. Mooney, Support vector machines, Machine Learning Group, Department of Computer Sciences, University of Texas at Austin, <http://www.cs.utexas.edu/users/mooney/cs391L/svm.ppt>.
- [46] S.C. Nash and A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill, 1996.
- [47] J.A. Nelder and R. Mead, A simplex method for function minimization, *Computer*, j.7 (1965) 308-313.
- [48] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer Series in Operations Research, 1999.
- [49] S. Özcan, V. Yıldırım, L. Kaya, D. Becher, M. Hecker and G. Özcengiz, Phaeo-rochaete Chrysosporium Proteomo and a large-scale study of heavy metal response, preprint, Department of Biology, METU (2005).
- [50] S. Pope, *Turbulent Flows*, University Press, Cambridge, 2000.
- [51] M.J.D. Powell, A direct search optimization method that models the objective and constraint functions by linear interpolation, in: *Advances in Optimization and Numerical Analysis*, Proceedings of the 6th Workshop on Optimization and

Numerical Analysis, Oaxaca, Mexico, Vol. 275, Kluwer Academic Publishers, Dordrecht (1994) 51-67.

- [52] M.J.D. Powell, A direct search optimization method that models the objective by the quadratic interpolation, presentation at the 5th Stockholm Optimization Days, 1994.
- [53] T. Sauer and Y. Xu, On multivariate Lagrange interpolation, *Mathematics of Computation* 64 (1995) 1147-1170.
- [54] M. Schäfer, B. Karasözen, Y. Uludağ, K.Yapıcı and Ö. Uğur, Numerical method for optimizing stirrer configurations, preprint, Institute of Applied Mathematics, METU, 2004.
- [55] K. Scheinberg, Derivative free optimization method, preprint, IBM T.J. WATSON Research Center, 2000.
- [56] Manual for FORTRAN software package DFO v2.0, preprint, IBM T.J. WATSON Research Center, 2003.
- [57] SDSC, A national laboratory for computational science and engineering, Overview of the SVM algorithm, <http://svm.sdsc.edu/svm-overview.html>.
- [58] B.A. Shapiro, J-C. Wu, D. Bengali and M. Potts, The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation, *Bioinformatics* 17 (2001) 137-148.
- [59] J. Shi, Parallelization of the Multigrid Solver for Flows in Complex Geometries: FASTEST2D-LBR, abstract, 2000 <http://www.fz-juelich.de/zam/gaststudenten/abstracts2000.html#SECTION05000>.
- [60] H. Spath, *Cluster Analysis Algorithms*, Ellis Horwood Limited, Chichester, 1980.

- [61] K.S. Al-Sultan and M.M. Khan, Computational experience on four algorithms for the hard clustering problem, *Pattern Recognition Letters* 17 (1996) 295-308.
- [62] G.B. Tatterson, *Scaleup and Design of Industrial Mixing Process*, McGraw-Hill, New York, 1994.
- [63] V. Torczon, On the convergence of the multidimensional search algorithm, *SIAM J. Optimization* 1 (1991) 123-145.
- [64] V. Torczon and J.E. Dennis, Direct search methods on parallel machines, *SIAM J. Optimization* 1 (1991) 448-474.
- [65] X. Wanf, Derivative-free optimization algorithms, report, Department of Computing and Software, McMaster University, 2003.
- [66] G. W. Weber, *Statistical Learning and Simulation*, lecture notes, Institute of Applied Mathematics, METU, Ankara, Fall 2003-2004.
- [67] G. W. Weber, *Inverse Problems*, lecture notes, Institute of Applied Mathematics, METU, Ankara, Spring 2004.
- [68] G.W. Weber, *Generalized Semi-Infinite Optimization and Related Topics*, in: *Research and Exposition in Mathematics*, K.H. Hofmann and R. Wille (eds.), Heldermann Verlag, Germany.
- [69] D. Winfield, *Function and Functional Optimization by Interpolation in Data Tables*, Ph.D. Thesis, Harvard University, Cambridge, MA, 1969.
- [70] D. Winfield, Function minimization by interpolation in a data table, *Journal of the Institute of Mathematics and its Applications* 12 (1973) 339-347.
- [71] M.H. Wright, Direct search methods: Once scorned, new respectable, in: *Proceeding of the 1995 Dundee Biennial Conference in Numerical Analysis*, D.F. Griffiths and G.A. Watson (eds.), Addison-Wesley, Reading, MA, and Longman, Harlow, UK, 1996.

[72] K. Yapıcı, *Numerical Method for Optimizing Stirrer Configurations*, M.Sc. Thesis, Graduate School of Natural and Applied Sciences, METU, 2004.

