Bundle Methods for Large-Scale Nonsmooth Optimization

Licentiate thesis

Marjo Haarala
hamasi@mit.jyu.fi

University of Jyväskylä
Department of Mathematical Information Technology
Scientific Computing

Abstract

Haarala Marjo, *hamasi@mit.jyu.fi*
Bundle Methods for Large-Scale Nonsmooth Optimization
Licentiate thesis, 79 pages and 4 Appendices
University of Jyväskylä, 2002

Many practical optimization applications involve nonsmooth (that is, not necessarily differentiable) functions of many variables. On the one hand, the direct application of smooth gradient-based methods may lead to a failure due to the nonsmooth nature of the problem. On the other hand, none of the current nonsmooth optimization methods is very efficient in large-scale settings. In this thesis, we introduce a new limited memory variable metric bundle method for nonsmooth large-scale optimization. We also give some encouraging results from numerical experiments.

Keywords: Nonsmooth optimization, large-scale optimization, bundle methods, variable metric methods, limited memory methods.

Preface

I would like to take this opportunity to thank my supervisors Dr. Marko Mäkelä and Dr. Kaisa Miettinen for excellent guidance and continuous support throughout my research. I am very grateful to Marko Mäkelä for introducing me into this field of nonsmooth optimization. I am also grateful to Dr. Ladislav Lukšan and Dr. Jan Vlček for permission to use and change their variable metric bundle software to make the method suitable for large-scale optimization. Finally, I would like to thank MSc. Tommi Ronkainen for enlightening discussions and advice concerning the computational implementations.

This work was financially supported by COMAS Graduate School of the University of Jyväskylä.

Jyväskylä, 22 November 2002

Marjo Haarala

List of Symbols

$\mathbb{R}^n$	n-dimensional Euclidean space
$\mathbb{N}$	set of natural numbers
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	(column) vectors
$\mathbf{x}^T$	transposed vector
$\mathbf{x}^T\mathbf{y}$	inner product of $\mathbf{x}$ and $\mathbf{y}$
$\\|\mathbf{x}\\|$	norm of $\mathbf{x}$ in $\mathbb{R}^n$, $\\|\mathbf{x}\\| = (\mathbf{x}^T\mathbf{x})^{\frac{1}{2}}$
x_i	component i of vector $\mathbf{x}$
$(\mathbf{x}_k)$	sequence
A, B, G	matrices
A^{-1}	inverse of matrix A
$\nabla f(\mathbf{x})$	gradient of f at $\mathbf{x}$
H	Hessian matrix ($H = \nabla^2 f(\mathbf{x})$)
B	approximation of the Hessian matrix
D	approximation of the inverse of the Hessian matrix
(a, b)	open interval
$[a, b]$	closed interval
$[a, b)$, $(a, b]$	half-open intervals
$B(\mathbf{x}, r)$	open ball with radius r and central point $\mathbf{x}$
$\operatorname{conv} S$	convex hull of set S
$\mathbf{x}_k \downarrow 0$	$\mathbf{x}_k \to 0^+$
$C^m(\mathbb{R}^n)$	the space of functions $f : \mathbb{R}^n \to \mathbb{R}$ with continuous partial derivatives up to order m
$f'(\mathbf{x}; \mathbf{d})$	directional derivative of f at $\mathbf{x}$ in the direction $\mathbf{d}$
$f^\circ(\mathbf{x}; \mathbf{d})$	generalized directional derivative of f at $\mathbf{x}$ in the direction $\mathbf{d}$
$\partial f(\mathbf{x})$	subdifferential of function f at $\mathbf{x}$
$\partial_c f(\mathbf{x})$	subdifferential of the convex function f at $\mathbf{x}$
$\partial_\varepsilon^G f(\mathbf{x})$	Goldstein ε-subdifferential of function f at $\mathbf{x}$
$\boldsymbol{\xi} \in \partial f(\mathbf{x})$	subgradient of function f at $\mathbf{x}$
$\arg\min f(\mathbf{x})$	point where function f attains its minimum value
$\mathcal{I}$, $\mathcal{J}$, $\mathcal{M}$	sets of indices
$\operatorname{diag}[\theta_1, \ldots, \theta_n]$	diagonal matrix with diagonal elements $\theta_1, \ldots, \theta_n$

Contents

1 Introduction

The classical theory of optimization presumes certain differentiability and strong regularity assumptions (see, e.g., Fletcher (1987)). However, these assumptions are too demanding for many practical applications, since the functions involved are often nonsmooth (that is, not necessarily differentiable). The source of nonsmoothness may be the objective function itself, its possible interior function or both. For example, piecewise linear tax models in economics (see Lemaréchal (1989)) and the phase changes of material in the continuous casting of steel (see, e.g., Miettinen et al. (1998)) typically contain various discontinuities and irregularities in the original phenomenon. In optimal control problems (see, e.g., Mäkelä and Neittaanmäki (1992)) the nonsmoothness is usually caused by some extra technological constraints. In addition, there exist so-called stiff problems that are analytically smooth but numerically nonsmooth. This means that the gradient varies too rabidly and, thus, the problems behave like nonsmooth problems.

In this thesis, we consider the unconstrained minimization of nonsmooth but locally Lipschitz continuous objective functions. This means that the objective function is not required to have continuous derivatives. The direct application of smooth gradient-based methods to nonsmooth problems may lead to a failure in convergence, in optimality conditions or in gradient approximation (see, e.g., Lemaréchal (1989)). On the other hand, the direct methods, for example, Powel's method (see, e.g., Fletcher (1987)) employing no derivative information, are quite unreliable and become inefficient when the size of the problem increases. Thus, nonsmooth optimization deals with a broader class of problems than smooth optimization in the sense that nonsmooth optimization techniques can be successfully applied to smooth problems but not vice versa (see, e.g., Clarke (1983)).

Optimization methods are typically iterative. The basic idea of iterative minimization methods is by starting from a given initial point $\mathbf{x}_1 \in \mathbb{R}^n$ to generate a sequence $(\mathbf{x}_k) \subset \mathbb{R}^n$ converging to a (local) minimum point $\mathbf{x}^*$ of the objective function f, that is $\mathbf{x}_k \to \mathbf{x}^*$ whenever $k \to \infty$. The next iteration point $\mathbf{x}_{k+1}$ is defined by the formula $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$, where $\mathbf{d}_k$ is a descent direction such that there exist $\varepsilon > 0$ such that $f(\mathbf{x}_k + t\mathbf{d}_k) < f(\mathbf{x})$ for all $t \in (0, \varepsilon]$ and t_k is the step size such that $t_k \approx \arg\min_{t>0} f(\mathbf{x} + t\mathbf{d}_k)$.

For smooth objective functions a descent direction may be generated by exploiting the fact that the direction opposite to the gradient is locally the steepest descent direction. Then, the step size can be determined by using line search, which usually employs some efficient univariate smooth optimization method or some polynomial interpolation. With smooth functions the necessary condition for local optimality is that the gradient must be zero at each local solution and by continuity it becomes small when we are close to an optimal point. This fact provides a useful stopping criterion for smooth iterative methods.

In nonsmooth problems, we have to use so-called generalized gradients (or subgradients) instead of gradients. This allows us to generalize the effective smooth gradient-based methods for nonsmooth problems. The methods for solving nonsmooth optimization problems can be divided into two main classes: subgradient methods and bundle methods. Both of these methods are based on the assumptions that the functions involved are locally Lipschitz continuous and only the value of the function and its arbitrary subgradient at each point are available. The basic idea behind the subgradient methods is to generalize smooth methods by replacing the gradient by an arbitrary subgradient. Due to this simple structure, they are widely used methods in nonsmooth optimization. However, there exist some serious drawbacks in these methods. Firstly, a nondescent search direction may occur, because the direction opposite to an arbitrary subgradient need not be a descent one. Thus, the standard line search operation can not be applied for step size selection. Secondly, due to the fact that an arbitrary subgradient does not necessarily become small in the neighborhood of an optimal point, there exists no an implementable stopping criterion. For more information of the subgradient methods we refer to Shor (1985).

At the moment, bundle methods are regarded as the most effective and reliable methods for nonsmooth optimization (see, e.g., Mäkelä and Neittaanmäki (1992)). They are based on the subdifferential theory developed by Rockafellar (1970) and Clarke (1983), where the classical differential theory is generalized for convex and locally Lipschitz continuous functions, respectively.

The basic idea of bundle methods is to approximate the subdifferential (that is, the set of subgradients) of the objective function by gathering the subgradients from previous iterations into a bundle. The history of bundle methods was initiated with the ε-steepest descent method introduced in Lemaréchal (1976). The idea of this method is to combine the cutting plane method (see Kelley (1960)) with the conjugate subgradient method (see Lemaréchal (1975)). The main difficulty of this method is the selection of the approximation tolerance that controls the radius of the ball where the the cutting plane model is thought to be a good approximation to the objective function. To avoid this difficulty the idea of the generalized cutting plane method was introduced in Lemaréchal (1978) and the ideas of this method were further developed in the book of Kiwiel (1985). The basic idea of the generalized cutting plane method is to form a convex piecewise linear approximation to the objective function by using linearizations generated by the subgradients. In his book, Kiwiel also presented two strategies to bound the number of stored subgradients, namely the subgradient selection and the subgradient aggregation. In spite of the different backgrounds of these methods both Lemaréchal's ε-steepest descent and Kiwiel's generalized cutting plane methods generate the search direction by solving quadratic direction finding problems that are closely related. Although the generalized cutting plane method avoids the difficulties of the ε-steepest descent method, there exits a great disadvantage also in this method: It has been found to be very sensitive to the scaling of the objective function (that is, multiplication of f by a positive constant) (see Mäkelä (1998)).

The next improvement of bundle methods was the development of the proximal bundle method by Kiwiel (1990) and the bundle trust region method by Schramm and Zowe (1992). The proximal bundle method is based on the proximal point algorithm of Rockafellar (1976) and the work of Auslender (1987) while the idea of the bundle trust region method is to combine the bundle idea with the classical trust region method (see, e.g., Fletcher (1987)). There exist strong similarities between these methods and, in fact, they use approximately the same algorithm that differs only in technical details.

Later, various different bundle methods have been proposed. Among them are tilted bundle methods (see Kiwiel (1991)), where the cutting plane model is replaced by a so-called tilted cutting plane model and the bundle-Newton method (see Lukšan and Vlček (1998)) using second-order information to construct a quadratic model of the objective function. For thorough overview of various bundle methods we refer to Mäkelä (2002).

In their present form, bundle methods are efficient for small- and medium-scale problems. However, their computational demand expands in large-scale problems with more than 1000 variables. This is explained by the fact that to be computationally efficient bundle methods need relatively large bundles. In other words, the size of the bundle has to be approximately the same as the number of the variables and, thus, the quadratic direction finding problem become very time-consuming. In variable metric bundle methods (see Lukšan and Vlček (1999a), Vlček and Lukšan (1999)) this problem is avoided by using the variable metric BFGS and SR1 updates to determine the search direction. The aggregation procedure is done using only three subgradients and, thus, the size of the bundle does not grow with the dimension of the problem. However, the variable metric bundle methods uses dense approximations of the Hessian matrix to calculate the search direction and, thus, also these methods become inefficient when the dimension of the problem increases.

We can say that at the moment, the only possibility to optimize nonsmooth large-scale problems is to use some subgradient methods but, as said before, these methods suffer from some serious disadvantages. This means that there is an evident need of reliable and efficient solvers for nonsmooth large-scale optimization problems.

In this thesis, we introduce a new limited memory variable metric bundle method for large-scale nonsmooth unconstrained minimization. The method to be presented is a hybrid of the variable metric bundle method (see Lukšan and Vlček (1999a), Vlček and Lukšan (1999)) and the limited memory variable metric method (see, e.g., Byrd et al. (1994)), where the latter has been developed for smooth large-scale optimization. The new method does not have to solve any time-consuming quadratic direction finding problems appearing in the standard bundle methods and it uses just few vectors to represent the variable metric approximation of the Hessian matrix and, thus, avoids storing and manipulating large matrices as is the case in variable metric bundle methods. These improvements make the limited memory variable metric bundle methods suitable for large-scale optimization, since the number of operations used for the calculations of the search direction and the aggregate values is only linearly dependent on the number of variables while, for example, with the original variable metric bundle method this dependence is quadratic.

Before introducing the new method we present some results from nonsmooth analysis and give a short description of some basic methods for (small-scale) nonsmooth optimization. To make the basis of the new method more steady, we also give a description of smooth limited memory variable metric methods.

In order to get some impression about how the different optimization methods (including our new method) operate in practice, we have tested them with large-scale minimization problems. Thus, in addition to the descriptions of the methods we are able to give some details of the performance of these methods applied to large-scale optimization. The numerical results to be presented demonstrate the usability of the new method with both smooth and nonsmooth large-scale minimization problems.

This thesis is organized as follows: In Chapter 2, we first recall some notations and basic results from smooth analysis. Then we generalize differential concepts for convex and locally Lipschitz continuous functions, respectively, and present some basic results. At the end of the chapter, we generalize the classical optimality conditions to nonsmooth case.

In Chapter 3, we give a short description of some basic methods for (small-scale) nonsmooth optimization. Firstly, we consider standard variable metric methods, which were originally developed for smooth unconstrained optimization. By replacing the gradient by an arbitrary subgradient these methods can be used also for nonsmooth optimization. Secondly, we give a survey of standard bundle methods. Thirdly, we present variable metric bundle methods that are hybrids of variable metric and bundle methods. At the end of the chapter, we describe the main ideas of the bundle-Newton methods using second order information of the objective function in the form of an approximative Hessian matrix.

In Chapter 4, we review some smooth methods for large-scale optimization. First, we give the basic idea of the limited memory BFGS method as it is presented in Nocedal (1980). Then we give compact matrix representations of variable metric updates and apply these representations to limited memory variable metric methods.

In Chapter 5, we introduce the new limited memory variable metric bundle method for large-scale nonsmooth unconstrained optimization and in Chapter 6 we analyze some numerical experiments concerning the methods presented in Chapters 3, 4 and 5.

Finally, in Chapter 7, we conclude by giving a short summary of the performance of the methods described in Chapters 3 and 4 as well as the promising results obtained with the new method introduced in Chapter 5.

2 Theoretical Background

In this chapter, we first give some notations and basic results from smooth analysis. Then we generalize differential concepts for convex, not necessarily differentiable functions. We define subgradients and subdifferentials and present some basic results. After that we generalize the convex differential theory to locally Lipschitz continuous functions and define so-called ε-subdifferentials that approximate the ordinary subdifferentials. In the third part of this chapter, we generalize the classical optimality conditions: We give the necessary conditions for a locally Lipschitz continuous function to attain its minimum in an unconstrained case. Moreover, we define some notions of linearizations for locally Lipschitz continuous functions and present their basic properties.

The proofs of this chapter are omitted since they can be found in Mäkelä and Neittaanmäki (1992).

2.1 Notations and Definitions

All the vectors $\mathbf{x}$ are considered as column vectors and, respectively, all the transposed vectors $\mathbf{x}^T$ are considered as row vectors. We denote by $\mathbf{x}^T\mathbf{y}$ the usual *inner product* and by $\|\mathbf{x}\|$ the *norm* in the n-dimensional real Euclidean space $\mathbb{R}^n$, that is,

$$\|\mathbf{x}\| = (\mathbf{x}^T\mathbf{x})^{\frac{1}{2}} = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}},$$

where $\mathbf{x} \in \mathbb{R}^n$ and $x_i \in \mathbb{R}$ is the i:th component of the vector $\mathbf{x}$.

An open ball with center $\mathbf{x} \in \mathbb{R}^n$ and radius $\lambda > 0$ is denoted by $B(\mathbf{x}; \lambda)$, that is,

$$B(\mathbf{x}; \lambda) = \{\, \mathbf{y} \in \mathbb{R}^n \mid \|\mathbf{y} - \mathbf{x}\| < \lambda \,\}.$$

A set $S \subset \mathbb{R}^n$ is said to be *convex* if

$$\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in S$$

whenever $\mathbf{x}$ and $\mathbf{y}$ are in S and $\lambda \in [0, 1]$. Geometrically this means that the *closed line-segment*

$$[\mathbf{x}, \mathbf{y}] = \{\, \mathbf{z} \in \mathbb{R}^n \mid \mathbf{z} = \lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \;\text{ for }\; \lambda \in [0, 1] \,\}$$

is entirely contained in S whenever its endpoints $\mathbf{x}$ and $\mathbf{y}$ are in S. If S_1 and S_2 are convex sets in $\mathbb{R}^n$ and $\lambda_1, \lambda_2 \in \mathbb{R}$, then the set $\lambda_1 S_1 + \lambda_2 S_2$ is also convex. If $S_i \subset \mathbb{R}^n$ are convex sets for $i = 1, \ldots, m$, then their intersection $\cap_{i=1}^{m} S_i$ is also convex.

A linear combination $\sum_{i=1}^{k} \lambda_i \mathbf{x}_i$ is called a *convex combination* of elements $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^n$ if each $\lambda_i \geq 0$ and $\sum_{i=1}^{k} \lambda_i = 1$.

The intersection of all the convex sets containing a given subset $S \subset \mathbb{R}^n$ is called the *convex hull* of set S and it is denoted by $\operatorname{conv} S$. For any $S \subset \mathbb{R}^n$, $\operatorname{conv} S$ consists of all the convex combinations of the elements of S, that is,

$$\operatorname{conv} S = \{\, \mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} = \sum_{i=1}^{k} \lambda_i \mathbf{x}_i,\; \sum_{i=1}^{k} \lambda_i = 1,\; \mathbf{x}_i \in S,\; \lambda_i \geq 0 \,\}.$$

The convex hull of set S is the smallest convex set containing S and S is convex if and only if $S = \operatorname{conv} S$. Furthermore, the convex hull of a compact set is compact.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex* if

$$f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}), \tag{2.1}$$

whenever $\mathbf{x}$ and $\mathbf{y}$ are in $\mathbb{R}^n$ and $\lambda \in [0,1]$. If strict inequality holds in (2.1) for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{x} \neq \mathbf{y}$ and $\lambda \in (0,1)$, the function f is said to be *strictly convex*.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *locally Lipschitz continuous* with constant $L > 0$ at $\mathbf{x} \in \mathbb{R}^n$ if there exists a positive number ε such that

$$|f(\mathbf{y}) - f(\mathbf{z})| \leq L \|\mathbf{y} - \mathbf{z}\|$$

for all $\mathbf{y}, \mathbf{z} \in B(\mathbf{x}; \varepsilon)$. In what follows, we use a shorter term locally Lipschitz. A convex function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz at $\mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^n$.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *positively homogeneous* if

$$f(\lambda \mathbf{x}) = \lambda f(\mathbf{x})$$

for all $\lambda \geq 0$ and *subadditive* if

$$f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$$

for all $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$. A function is said to be *sublinear* if it is both positively homogeneous and subadditive. A sublinear function is always convex.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *upper semicontinuous* at $\mathbf{x} \in \mathbb{R}^n$ if for every sequence $(\mathbf{x}_k)$ converging to $\mathbf{x}$ the following holds

$$\limsup_{k \to \infty} f(\mathbf{x}_k) \leq f(\mathbf{x})$$

and *lower semicontinuous* if

$$f(\mathbf{x}) \leq \liminf_{k \to \infty} f(\mathbf{x}_k).$$

A both upper and lower semicontinuous function is continuous.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *differentiable* at $\mathbf{x} \in \mathbb{R}^n$ if there exists a vector $\nabla f(\mathbf{x}) \in \mathbb{R}^n$ and a function $\varepsilon : \mathbb{R}^n \to \mathbb{R}$ such that for all $\mathbf{d} \in \mathbb{R}^n$

$$f(\mathbf{x} + \mathbf{d}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{d} + \|\mathbf{d}\| \varepsilon(\mathbf{d}),$$

where the vector $\nabla f(\mathbf{x})$ is the *gradient vector* of f at $\mathbf{x}$ and $\varepsilon(\mathbf{d}) \to 0$ whenever $\|\mathbf{d}\| \to 0$. The gradient vector $\nabla f(\mathbf{x})$ has the following formula

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right)^T,$$

where components $\frac{\partial f(\mathbf{x})}{\partial x_i}$, $i = 1, \ldots, n$ are called *partial derivatives* of the function f. If the function is differentiable and all the partial derivatives are continuous, then the function is said to be *continuously differentiable* or *smooth* ($f \in C^1(\mathbb{R}^n)$).

The limit

$$f'(\mathbf{x}; \mathbf{d}) = \lim_{t \downarrow 0} \frac{f(\mathbf{x} + t\mathbf{d}) - f(\mathbf{x})}{t}$$

(if it exists) is called the *directional derivative* of f at $\mathbf{x} \in \mathbb{R}^n$ in the direction $\mathbf{d} \in \mathbb{R}^n$. The function $\mathbf{d} \mapsto f'(\mathbf{x}; \mathbf{d})$ is positively homogeneous and subadditive, in other words, it is sublinear. If a function f is differentiable at $\mathbf{x}$, then the directional derivative exists in every direction $\mathbf{d} \in \mathbb{R}^n$ and

$$f'(\mathbf{x}; \mathbf{d}) = \nabla f(\mathbf{x})^T \mathbf{d}.$$

If, in addition, f is convex, then for all $\mathbf{y} \in \mathbb{R}^n$

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}).$$

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *twice differentiable* at $\mathbf{x} \in \mathbb{R}^n$ if there exists a vector $\nabla f(\mathbf{x}) \in \mathbb{R}^n$ and a symmetric matrix $\nabla^2 f(\mathbf{x}) \in \mathbb{R}^{n \times n}$ and a function $\varepsilon : \mathbb{R}^n \to \mathbb{R}$ such that for all $\mathbf{d} \in \mathbb{R}^n$

$$f(\mathbf{x} + \mathbf{d}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \nabla^2 f(\mathbf{x}) \mathbf{d} + \|\mathbf{d}\| \varepsilon(\mathbf{d}),$$

where $\varepsilon(\mathbf{d}) \to 0$ whenever $\|\mathbf{d}\| \to 0$. The matrix $\nabla^2 f(\mathbf{x})$ is called the *Hessian* of the function f at $\mathbf{x}$ and it is defined to consist of second partial derivatives of f, that is,

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}.$$

If the function is twice differentiable and all the second partial derivatives are continuous, then the function is said to be *twice continuously differentiable* ($f \in C^2(\mathbb{R}^n)$).

A matrix $A \in \mathbb{R}^{n \times n}$ is called *positive definite* if $A = A^T$, that is, A is *symmetric* and

$$\mathbf{x}^T A \mathbf{x} > 0$$

for all nonzero $\mathbf{x} \in R^n$.

Let A and B be $n \times n$ matrices. Suppose that

$$AB = BA = I,$$

where I is the *identity matrix.* Then B is called an *inverse* of A and is denoted by A^{-1}. If A has an inverse, then A is said to be *invertible* or *nonsingular.* Otherwise, A is said to be *singular.*

A matrix $A \in \mathbb{R}^{m \times n}$ is said to be *bounded*, if its eigenvalues lie in the compact interval that does not contain zero.

From now on we use some special notations for special matrices: the Hessian matrix of the objective function is denoted by H, the approximation of the Hessian is denoted by B and the approximation of the inverse of the Hessian is denoted by D.

2.2 Nonsmooth Analysis

We first define the *subgradient* and the *subdifferential* of a convex function (see Rockafellar (1970)). Then we generalize these results to nonconvex locally Lipschitz continuous functions.

Definition 2.2.1. *The* subdifferential *of a convex function* $f : \mathbb{R}^n \to \mathbb{R}$ *at* $\mathbf{x} \in \mathbb{R}^n$ *is the set* $\partial_c f(\mathbf{x})$ *of vectors* $\boldsymbol{\xi} \in \mathbb{R}^n$ *such that*

$$\partial_c f(\mathbf{x}) = \{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid f(\mathbf{y}) \geq f(\mathbf{x}) + \boldsymbol{\xi}^T(\mathbf{y} - \mathbf{x}) \text{ for all } \mathbf{y} \in \mathbb{R}^n \,\}.$$

Each vector $\boldsymbol{\xi} \in \partial_c f(\mathbf{x})$ *is called a* subgradient *of* f *at* $\mathbf{x}$.

Theorem 2.2.2. *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a convex function. Then the directional derivative exists in any direction* $\mathbf{d} \in \mathbb{R}^n$ *and it satisfies*

$$f'(\mathbf{x}; \mathbf{d}) = \inf_{t>0} \frac{f(\mathbf{x} + t\mathbf{d}) - f(\mathbf{x})}{t}.$$

Next we present the relationship between subdifferentials and the directional derivatives. It turns out that it is enough to know either of the concepts; one can be computed from the other.

Theorem 2.2.3. *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a convex function. Then for all* $\mathbf{x} \in \mathbb{R}^n$

(i) $f'(\mathbf{x}; \mathbf{d}) = \max\{\, \boldsymbol{\xi}^T \mathbf{d} \mid \boldsymbol{\xi} \in \partial_c f(\mathbf{x}) \,\}$ *for all* $\mathbf{d} \in \mathbb{R}^n$,

(ii) $\partial_c f(\mathbf{x}) = \{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid f'(\mathbf{x}, \mathbf{d}) \geq \boldsymbol{\xi}^T \mathbf{d}$ *for all* $\mathbf{d} \in \mathbb{R}^n \,\}$,

(iii) *$\partial_c f(\mathbf{x})$ is a nonempty, convex and compact set such that $\partial_c f(\mathbf{x}) \subset B(0; L)$, where $L > 0$ is the Lipschitz constant of f at $\mathbf{x}$.*

Since for locally Lipschitz continuous functions there need not exist any classical directional derivatives, we first define a *generalized directional derivative* (see Clarke (1983)). Then we generalize the subdifferential to nonconvex locally Lipschitz continuous functions.

Definition 2.2.4. *(Clarke). Let a function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at a point $\mathbf{x} \in \mathbb{R}^n$. The* generalized directional derivative *of f at $\mathbf{x}$ in the direction $\mathbf{d} \in \mathbb{R}^n$ is defined by*

$$f^\circ(\mathbf{x}; \mathbf{d}) = \limsup_{\substack{\mathbf{y} \to \mathbf{x} \\ t \downarrow 0}} \frac{f(\mathbf{y} + t\mathbf{d}) - f(\mathbf{y})}{t}.$$

Definition 2.2.5. *(Clarke). Let a function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at a point $\mathbf{x} \in \mathbb{R}^n$. The subdifferential of f at $\mathbf{x}$ is the set of vectors $\boldsymbol{\xi} \in \mathbb{R}^n$ such that*

$$\partial f(\mathbf{x}) = \{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid f^\circ(\mathbf{x}; \mathbf{d}) \geq \boldsymbol{\xi}^T \mathbf{d} \text{ for all } \mathbf{d} \in \mathbb{R}^n \,\}.$$

Each vector $\boldsymbol{\xi} \in \partial f(\mathbf{x})$ is called a subgradient of f at $\mathbf{x}$.

The subdifferential has the following basic properties.

Theorem 2.2.6. *Let a function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$ with constant L. Then*

(i) *$f^\circ(\mathbf{x}; \mathbf{d}) = \max\{\, \boldsymbol{\xi}^T \mathbf{d} \mid \boldsymbol{\xi} \in \partial f(\mathbf{x}) \,\}$ for all $\mathbf{d} \in \mathbb{R}^n$,*

(ii) *$\partial f(\mathbf{x})$ is a nonempty, convex and compact set such that $\partial f(\mathbf{x}) \subset B(0; L)$.*

The next theorem shows that the subdifferential for locally Lipschitz continuous functions is a generalization of the subdifferential for convex functions.

Theorem 2.2.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then*

(i) *$f'(\mathbf{x}; \mathbf{d}) = f^\circ(\mathbf{x}; \mathbf{d})$ for all $\mathbf{d} \in \mathbb{R}^n$ and*

(ii) *$\partial_c f(\mathbf{x}) = \partial f(\mathbf{x})$.*

The next two theorems show that the subdifferential really is a generalization of the classical derivative.

Theorem 2.2.8. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be both locally Lipschitz and differentiable at $\mathbf{x} \in \mathbb{R}^n$. Then*

$$\nabla f(\mathbf{x}) \in \partial f(\mathbf{x}).$$

Theorem 2.2.9. *If $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable at $\mathbf{x} \in \mathbb{R}^n$, then*

$$\partial f(\mathbf{x}) = \{\, \nabla f(\mathbf{x}) \,\}.$$

Theorem 2.2.10. *(Rademacher). Let $S \subset \mathbb{R}^n$ be an open set. A function $f : S \to \mathbb{R}$ that is locally Lipschitz on S is differentiable almost everywhere on S.*

By Rademacher's Theorem we know that a locally Lipschitz continuous function is differentiable almost everywhere and, thus, the gradient exists almost everywhere. Now, the subdifferential can be reconstructed as a convex hull of all possible limits of gradients at points $(\mathbf{x}_i)$ converging to $\mathbf{x}$.

The set of points in which a given function f fails to be differentiable is denoted by Ω_f.

Theorem 2.2.11. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$. Then*

$$\begin{aligned}\partial f(\mathbf{x}) = \operatorname{conv}\{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid & \textit{ there exists } (\mathbf{x}_i) \subset \mathbb{R}^n \setminus \Omega_f \textit{ such that} \\ & \mathbf{x}_i \to \mathbf{x} \textit{ and } \nabla f(\mathbf{x}_i) \to \boldsymbol{\xi} \,\}.\end{aligned}$$

In nonsmooth optimization, so-called bundle methods are based on the theory of *ε-subdifferentials.* Next we define the *Goldstein ε-subdifferentials* for nonconvex functions.

Definition 2.2.12. *Let a function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$ and let $\varepsilon \geq 0$. Then the* Goldstein ε-subdifferential *of f is the set*

$$\partial_\varepsilon^G f(\mathbf{x}) = \operatorname{conv}\{\, \partial f(\mathbf{y}) \mid \mathbf{y} \in B(\mathbf{x}; \varepsilon) \,\}.$$

Each element $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\mathbf{x})$ is called an ε-subgradient *of the function f at $\mathbf{x}$.*

The following theorem summarizes some basic properties of the Goldstein ε-subdifferential.

Theorem 2.2.13. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz function at $\mathbf{x} \in \mathbb{R}^n$ with constant L. Then*

(i) $\partial_0^G f(\mathbf{x}) = \partial f(\mathbf{x})$,

(ii) *if* $\varepsilon_1 \leq \varepsilon_2$, *then* $\partial^G_{\varepsilon_1} f(\mathbf{x}) \subset \partial^G_{\varepsilon_2} f(\mathbf{x})$, *and*

(iii) $\partial^G_{\varepsilon} f(\mathbf{x})$ *is a nonempty, convex and compact set such that* $\|\boldsymbol{\xi}\| \leq L$ *for all* $\boldsymbol{\xi} \in \partial^G_{\varepsilon} f(\mathbf{x})$.

Corollary 2.2.14. *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a locally Lipschitz function at* $\mathbf{x} \in \mathbb{R}^n$. *Then*

$$\partial^G_{\varepsilon} f(\mathbf{x}) = \operatorname{conv}\{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid \textit{there exists } (\mathbf{y}_i) \subset \mathbb{R}^n \setminus \Omega_f \textit{ such that } \mathbf{y}_i \to \mathbf{y},\ \nabla f(\mathbf{y}_i) \to \boldsymbol{\xi} \textit{ and } \mathbf{y} \in B(\mathbf{x};\varepsilon)\,\}.$$

2.3 Optimality Conditions

In this section we present some results connecting the theory of nonsmooth analysis and optimization. We first define global and local minima of functions. After that, we generalize the classical first order optimality conditions for unconstrained optimization. At the end of this chapter we define some notions of linearizations for locally Lipschitz continuous functions and present their basic properties.

We consider a nonsmooth unconstrained optimization problem of the form

$$\begin{cases} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathbb{R}^n, \end{cases} \tag{2.2}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous at $\mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^n$.

Definition 2.3.1. *A point* $\mathbf{x} \in \mathbb{R}^n$ *is a* global minimum *of* f, *if it satisfies*

$$f(\mathbf{x}) \leq f(\mathbf{y}) \quad \textit{for all } \mathbf{y} \in \mathbb{R}^n.$$

Definition 2.3.2. *A point* $\mathbf{x} \in \mathbb{R}^n$ *is a* local minimum *of* f, *if there exists* $\varepsilon > 0$ *such that*

$$f(\mathbf{x}) \leq f(\mathbf{y}) \quad \textit{for all } \mathbf{y} \in B(\mathbf{x};\varepsilon).$$

Theorem 2.3.3. *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a locally Lipschitz function at* $\mathbf{x} \in \mathbb{R}^n$. *If* f *attains its minimum at* $\mathbf{x}$, *then*

$$\mathbf{0} \in \partial f(\mathbf{x}).$$

Definition 2.3.4. *A point* $\mathbf{x} \in \mathbb{R}^n$ *satisfying* $\mathbf{0} \in \partial f(\mathbf{x})$ *is called a* critical *or a* stationary point *of* f.

We give now the necessary conditions for a locally Lipschitz continuous function to attain its local minimum in an unconstrained case. For a convex objective function these conditions are also sufficient and the minimum is global.

Theorem 2.3.5. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$. If f attains its local minimum at $\mathbf{x}$, then*

(i) $\mathbf{0} \in \partial f(\mathbf{x})$,

(ii) $f^\circ(\mathbf{x}; \mathbf{d}) \geq 0$ *for all* $\mathbf{d} \in \mathbb{R}^n$.

Theorem 2.3.6. *If $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then the following conditions are equivalent:*

(i) *Function f attains its global minimum at $\mathbf{x}$.*

(ii) $\mathbf{0} \in \partial_c f(\mathbf{x})$.

(iii) $f'(\mathbf{x}; \mathbf{d}) \geq 0$ *for all* $\mathbf{d} \in \mathbb{R}^n$.

Now we present an optimality condition with the help of ε-subdifferentials.

Corollary 2.3.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$. If f attains its local minimum at $\mathbf{x}$, then*

$$\mathbf{0} \in \partial_\varepsilon^G f(\mathbf{x}).$$

Let us next define some notions of *linearization* for locally Lipschitz continuous functions and present their basic properties. With these linearizations we can construct a piecewise linear local approximation to the unconstrained optimization problem (2.2). This approximation is used in nonsmooth optimization methods in the next chapter.

Definition 2.3.8. *Let the function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$ and let $\boldsymbol{\xi} \in \partial f(\mathbf{x})$ be an arbitrary subgradient. Then the ξ-*linearization *of f at $\mathbf{x}$ is the function $\bar{f}_\xi : \mathbb{R}^n \to \mathbb{R}$ defined by*

$$\bar{f}_\xi(\mathbf{y}) = f(\mathbf{x}) + \boldsymbol{\xi}^T(\mathbf{y} - \mathbf{x})$$

for all $\mathbf{y} \in \mathbb{R}^n$ and the linearization *of f at $\mathbf{x}$ is the function $\hat{f}_\mathbf{x} : \mathbb{R}^n \to \mathbb{R}$ such that*

$$\hat{f}_\mathbf{x}(\mathbf{y}) = \max\{\, \bar{f}_\xi(\mathbf{y}) \mid \boldsymbol{\xi} \in \partial f(\mathbf{x}) \,\} \tag{2.3}$$

for all $\mathbf{y} \in \mathbb{R}^n$.

Theorem 2.3.9. *Let the function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$. Then the linearization $\hat{f}_{\mathbf{x}}$ is convex and*

(i) $\hat{f}_{\mathbf{x}}(\mathbf{x}) = f(\mathbf{x})$,

(ii) $\hat{f}_{\mathbf{x}}(\mathbf{y}) = f(\mathbf{x}) + f^\circ(\mathbf{x}; \mathbf{y} - \mathbf{x})$ *for all* $\mathbf{y} \in \mathbb{R}^n$ *and*

(iii) $\partial_c \hat{f}_{\mathbf{x}}(\mathbf{x}) = \partial f(\mathbf{x})$.

Theorem 2.3.10. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then*

(i) $f(\mathbf{y}) = \max\{\, \hat{f}_{\mathbf{x}}(\mathbf{y}) \mid \mathbf{x} \in \mathbb{R}^n \,\}$ *for all* $\mathbf{y} \in \mathbb{R}^n$ *and*

(ii) $\hat{f}_{\mathbf{x}}(\mathbf{y}) \le f(\mathbf{y})$ *for all* $\mathbf{y} \in \mathbb{R}^n$.

The fundamental difficulty in iterative optimization methods is to find a direction such that the function values will decrease when moving in this direction. Next we define *a descent direction* for a function $f : \mathbb{R}^n \to \mathbb{R}$.

Definition 2.3.11. *The direction $\mathbf{d} \in \mathbb{R}^n$ is said to be a* descent direction *for $f : \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x} \in \mathbb{R}^n$, if there exists $\varepsilon > 0$ such that for all $t \in (0, \varepsilon]$*

$$f(\mathbf{x} + t\mathbf{d}) < f(\mathbf{x}).$$

The next theorem shows how to find descent directions for a locally Lipschitz continuous function.

Theorem 2.3.12. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$. The direction $\mathbf{d} \in \mathbb{R}^n$ is a descent direction for f if any of the following holds:*

(i) $f^\circ(\mathbf{x}; \mathbf{d}) < 0$,

(ii) $\boldsymbol{\xi}^T \mathbf{d} < 0$ *for all* $\boldsymbol{\xi} \in \partial f(\mathbf{x})$,

(iii) $\boldsymbol{\xi}^T \mathbf{d} < 0$ *for all* $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\mathbf{x})$,

(iv) $\mathbf{d}$ *is a descent direction for the linearization* $\hat{f}_{\mathbf{x}}$ *at* $\mathbf{x}$.

The most promising nonsmooth optimization methods are based on the following theorem. It tells how to find a descent direction for the linearization function. By Theorem 2.3.12 this direction is a descent direction also for the original function.

Theorem 2.3.13. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at $\mathbf{x} \in \mathbb{R}^n$ and let $\boldsymbol{\xi}^* \in \partial f(\mathbf{x})$ exist such that $\boldsymbol{\xi}^* = \arg\min\{\, \|\boldsymbol{\xi}\| \mid \boldsymbol{\xi} \in \partial f(\mathbf{x})\,\}$. Consider the problem*

$$\begin{cases} \text{minimize} & \hat{f}_{\mathbf{x}}(\mathbf{x}+\mathbf{d}) + \frac{1}{2}\|\mathbf{d}\|^2 \\ \text{subject to} & \mathbf{d} \in \mathbb{R}^n. \end{cases} \tag{2.4}$$

Then

(i) *The problem (2.4) has a unique solution $\mathbf{d}^* \in \mathbb{R}^n$ such that $\mathbf{d}^* = -\boldsymbol{\xi}^*$,*

(ii) $f^\circ(\mathbf{x};\mathbf{d}^*) = -\|\mathbf{d}^*\|^2$,

(iii) $\hat{f}_{\mathbf{x}}(\mathbf{x}+\lambda\mathbf{d}^*) = \hat{f}_{\mathbf{x}}(\mathbf{x}) - \lambda\|\boldsymbol{\xi}^*\|^2$ *for all* $\lambda \in [0,1]$,

(iv) $0 \notin \partial f(\mathbf{x}) \iff \mathbf{d}^* \neq 0$,

(v) $0 \in \partial f(\mathbf{x}) \iff \hat{f}_{\mathbf{x}}$ *attains its global minimum at* $\mathbf{x}$.

Finally, we say a few words about convergence. In iterative optimization methods we try to generate a sequence $(\mathbf{x}_k)$ converging to a minimum point $\mathbf{x}^*$ of the objective function f, that is $\mathbf{x}_k \to \mathbf{x}^*$ whenever $k \to \infty$. If an iterative method converges to a (local) minimum $\mathbf{x}^*$ from any arbitrary starting point $\mathbf{x}_1$, it is said to be *globally convergent.* If it converges to a (local) minimum in some neighborhood of $\mathbf{x}^*$, it is said to be *locally convergent.*

Definition 2.3.14. *(Rate of the convergence) An iterative method is said to be* linearly convergent *to an optimal solution $\mathbf{x}^*$, if there exist $\alpha \in [0,1)$ and $M \geq 0$ such that for all $k \geq M$*

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \alpha\|\mathbf{x}_k - \mathbf{x}^*\|.$$

An iterative method is said to be superlinearly convergent *to an optimal solution $\mathbf{x}^*$, if there exist $M \geq 0$ and for some sequence (α_k) converging to zero, there is*

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \alpha_k\|\mathbf{x}_k - \mathbf{x}^*\|$$

for all $k \geq M$.

3 Basic Methods of Nonsmooth Optimization

In this chapter, we give a short description of some basic methods for nonsmooth optimization. Firstly, we consider standard variable metric methods, which have originally been developed for smooth unconstrained optimization. By replacing the gradient by an arbitrary subgradient these methods can be used also for nonsmooth optimization. Secondly, we give a survey of standard bundle methods. At the moment, these methods are regarded as the most effective methods for nonsmooth optimization. Next we present the variable metric bundle methods developed by Lukšan and Vlček (Lukšan and Vlček (1999a), Vlček and Lukšan (1999)). The methods presented are hybrids of variable metric and bundle methods. Finally we describe the main ideas of bundle-Newton methods.

3.1 Variable Metric Methods

Standard variable metric methods or quasi-Newton methods have originally been developed for smooth unconstrained optimization (see, e.g., Fletcher (1987)). However, for example Lemaréchal (1982) has shown that these methods are quite efficient and robust for nonsmooth problems as well.

Let us first assume that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function whose gradient $\nabla f(\mathbf{x})$ is available for all $\mathbf{x} \in \mathbb{R}^n$. Variable metric methods are iterative methods based on Newton's method (see, e.g., Fletcher (1987)). In these methods, the Hessian of the objective function is approximated by symmetric positive definite matrices B_k ($k \in \mathbb{N}$ is the iteration number). At each iteration, the current approximation B_k is updated to a new approximation B_{k+1} satisfying the so-called *secant* or *quasi-Newton equation*

$$B_{k+1}\mathbf{s}_k = \mathbf{u}_k, \tag{3.1}$$

where $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{u}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{x}_k$ is the current iteration point, $\mathbf{x}_{k+1}$ is the next iteration point and $\nabla f(\mathbf{x}_k)$ and $\nabla f(\mathbf{x}_{k+1})$ are the corresponding gradients at those points, respectively. The idea of the secant equation (3.1) is that the approximation of the Hessian B_k acts as close as possible like the true Hessian of the objective function. In one dimensional case, this means that the tangent is approximated by a secant (see Figure 1).

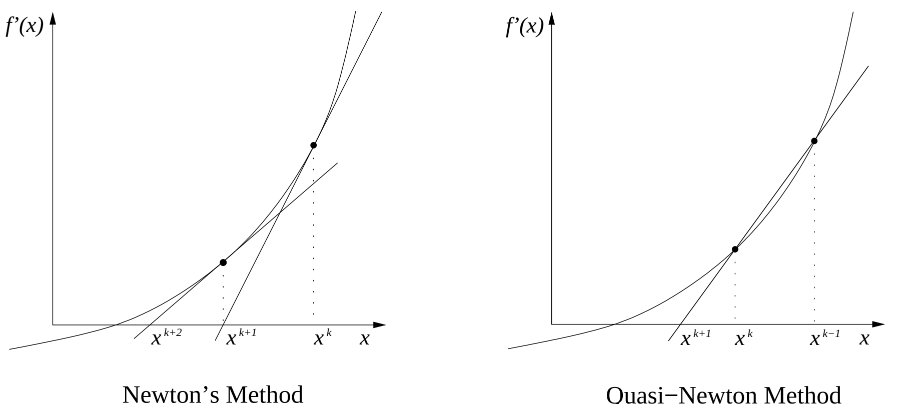

Figure 1: Solving equation $\nabla f(x) = 0$.

The search direction $\mathbf{d}_k$ can be determined by solving a linear system

$$B_k \mathbf{d}_k = -\nabla f(\mathbf{x}_k).$$

Alternatively, instead of matrices B_k we can approximate matrices $D_k = B_k^{-1}$. In this case, the search direction $\mathbf{d}_k$ can be computed directly by

$$\mathbf{d}_k = -D_k \nabla f(\mathbf{x}_k),$$

where D_k is a symmetric positive definite approximation of the inverse of the Hessian.

The next iteration point is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k,$$

where the step size $t_k > 0$ is chosen by some line search algorithm so that it satisfies the Wolfe conditions:

$$f(\mathbf{x}_k + t_k \mathbf{d}_k) - f(\mathbf{x}_k) \leq \varepsilon_L t_k \mathbf{d}_k^T \nabla f(\mathbf{x}_k) \tag{3.2}$$

and

$$\mathbf{d}_k^T \nabla f(\mathbf{x}_k + t_k \mathbf{d}_k) \geq \varepsilon_R \mathbf{d}_k^T \nabla f(\mathbf{x}_k), \tag{3.3}$$

where $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$ are some fixed line search parameters. These conditions guarantee that the step size t_k that gives a significant reduction in f always exist and it can be determined in a finite number of steps (see, e.g., Fletcher (1987)).

The approximation D_k of the inverse of the Hessian matrix is updated recursively by the formula

$$D_{k+1} = \gamma_k \left(D_k + \frac{\rho_k}{\gamma_k} \frac{1}{b_k} \mathbf{s}_k \mathbf{s}_k^T - \frac{1}{a_k} D_k \mathbf{u}_k \mathbf{u}_k^T D_k \right. \\ \left. + \frac{\eta_k}{a_k} \left(\frac{a_k}{b_k} \mathbf{s}_k - D_k \mathbf{u}_k \right) \left(\frac{a_k}{b_k} \mathbf{s}_k - D_k \mathbf{u}_k \right)^T \right), \quad (3.4)$$

where, as before, $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{u}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $a_k = \mathbf{u}_k^T D_k \mathbf{u}_k$, $b_k = \mathbf{u}_k^T \mathbf{s}_k$ and where γ_k, η_k and ρ_k are some positive parameters (see, e.g., Lukšan and Spedicato (2000)). The formula (3.4) defines a three-parameter class, the so-called *Huang-Oren class* of variable metric updates (see, e.g., Huang (1970)).

If we either assume ρ_k and γ_k to be fixed at every iteration or if we set

$$\rho_k = \frac{\mathbf{s}_k^T \mathbf{u}_k}{2(f_k - f_{k+1} + \mathbf{s}_k^T \nabla f_{k+1})} \qquad \text{and} \qquad (3.5)$$

$$\gamma_k = \rho_k \sqrt{\frac{c_k}{a_k}}, \qquad (3.6)$$

where $f_k = f(\mathbf{x}_k)$, $f_{k+1} = f(\mathbf{x}_{k+1})$, $\nabla f_{k+1} = \nabla f(x_{k+1})$ and $c_k = \mathbf{s}_k^T D_k^{-1} \mathbf{s}_k$, then the formula (3.4) defines a one-parameter class, the so-called *scaled Broyden class* (the original Broyden class corresponds to the values $\rho_k = \gamma_k = 1$) (see, e.g., Lukšan and Spedicato (2000)).

There exist three classical values for parameter η_k that are in common use. By setting $\eta_k = 0$, we get the scaled *Davidon-Fletcher-Powel* (DFP) update

$$D_{k+1} = \gamma_k \left(D_k + \frac{\rho_k}{\gamma_k} \frac{1}{b_k} \mathbf{s}_k \mathbf{s}_k^T - \frac{1}{a_k} D_k \mathbf{u}_k \mathbf{u}_k^T D_k \right).$$

The original formula was first suggested as a part of a method by Davidon (1959) and later it was presented as described here (with scaling factors $\rho_k = \gamma_k = 1$) by Fletcher and Powell (1963).

By setting $\eta_k = 1$ in (3.4), we get the scaled *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) update

$$D_{k+1} = \gamma_k \left(D_k + \left(\frac{\rho_k}{\gamma_k} + \frac{a_k}{b_k} \right) \frac{1}{b_k} \mathbf{s}_k \mathbf{s}_k^T - \frac{1}{b_k} \left(D_k \mathbf{u}_k \mathbf{s}_k^T + \mathbf{s}_k \mathbf{u}_k^T D_k \right) \right). \quad (3.7)$$

The original formula with scaling factors $\rho_k = \gamma_k = 1$ was developed by Broyden (1970), Fletcher (1970), Goldfarb (1970) and Shanno (1970) and since that, the BFGS method has been generally considered to be the most effective of many variable metric methods (see, e.g., Fletcher (1987) and Nocedal (1992)).

Finally, by setting $\eta_k = (\rho_k/\gamma_k)/(\rho_k/\gamma_k - a_k/b_k)$, we get the scaled *symmetric rank-one* (SR1) update

$$D_{k+1} = \gamma_k \left(D_k + \left(\frac{\rho_k}{\gamma_k} - \frac{a_k}{b_k} \right)^{-1} \frac{1}{b_k} \left(\frac{\rho_k}{\gamma_k}\mathbf{s}_k - D_k\mathbf{u}_k \right) \left(\frac{\rho_k}{\gamma_k}\mathbf{s}_k - D_k\mathbf{u}_k \right)^T \right). \tag{3.8}$$

The original formula was first discovered by Davidon (1959) in his seminal paper of quasi-Newton methods and it was re-discovered by several authors in the late 1960s.

The details of updating matrices B_k and more information on variable metric methods for smooth optimization in general can be found, for example, in Fletcher (1987), Lukšan and Spedicato (2000) and Nocedal (1992).

Let us next leave the differentiability and suppose only that the objective function is locally Lipschitz continuous. By replacing the gradient $\nabla f(\mathbf{x})$ by an arbitrary subgradient $\boldsymbol{\xi}$ the variable metric methods described above can also be used for nonsmooth optimization. The values from formulae (3.5) and (3.6) and parameter η_k computed by

$$\eta_k = \frac{\max\{0, \sqrt{c_k/a_k} - (b_k)^2/(a_k c_k)\}}{1 - (b_k)^2/(a_k c_k)} \tag{3.9}$$

have turned out to be suitable in nonsmooth case (Lukšan and Vlček (2001)). However, the formula (3.6) should not be used in all iterations. Controlled scaling proposed in Lukšan (1990), which combines the value of the formula (3.6) with the value $\gamma_k = 1$ has been shown to be more advantageous (Lukšan and Vlček (2001)).

Now we present a variable metric algorithm suitable also for nonsmooth optimization.

Algorithm 3.1. (Variable Metric Method).

Data: Choose a final accuracy tolerance $\varepsilon > 0$ and positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$.

Step 0: (*Initialization.*) Choose a starting point $\mathbf{x}_1 \in \mathbb{R}^n$ and a symmetric, positive definite matrix D_1, for example, $D_1 = I$. Compute $f_1 = f(\mathbf{x}_1)$ and $\boldsymbol{\xi}_1 \in \partial f(\mathbf{x}_1)$. Set the iteration counter $k = 1$.

Step 1: (*Direction finding.*) If $\|\boldsymbol{\xi}_k\| \leq \varepsilon$, then stop. Otherwise, set $\mathbf{d}_k = -D_k\boldsymbol{\xi}_k$.

Step 2: (*Line search.*) Determine a step size t_k satisfying the Wolfe conditions (3.2) and (3.3). Set $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$. Compute $f_{k+1} = f(\mathbf{x}_{k+1})$ and $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{x}_{k+1})$.

If $|f_{k+1} - f_k| \leq \varepsilon$ in several consecutive iterations, then terminate the computation.

Step 3: (*Update.*) Form the updated matrix D_{k+1} by (3.4) with parameters (3.5), (3.6) and (3.9) and with controlled scaling. Increase k by one and go to Step 1.

It can be proved under mild assumptions (see, e.g., Byrd et al. (1987)) that the variable metric method presented in Algorithm 3.1 is globally and superlinearly convergent in the smooth case. Although this result cannot be generalized for a nonsmooth case, Algorithm 3.1 has been found to be surprisingly robust for solving nonsmooth problems (see, e.g., Lemaréchal (1982) and Lukšan and Vlček (1999a, 2001)). In a nonsmooth case, Algorithm 3.1 usually terminates in Step 2, since sequence $(\|\boldsymbol{\xi}_k\|)$ does not necessarily converge to zero (see Lukšan and Vlček (2001)).

3.2 Bundle Methods

Since a locally Lipschitz continuous function is differentiable almost everywhere by Rademacher's theorem (Theorem 2.2.10), then usually $\boldsymbol{\xi} = \{\nabla f(\mathbf{x})\}$. For a nonsmooth objective function f the gradient $\nabla f(\mathbf{x})$ can change discontinuously and it may not be small in the neighborhood of the minimum of the function. For this reason, values $f(\mathbf{x}_k)$ and $\boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$ at a single point $\mathbf{x}_k$ do not offer sufficient information of the local behavior of the function f. The basic idea of bundle methods is to approximate the whole subdifferential $\partial f(\mathbf{x}_k)$ by gathering together a bundle of subgradients calculated close to the iteration point $\mathbf{x}_k$.

In this section, we describe a general bundle method that produces a sequence $(\mathbf{x}_k) \in \mathbb{R}^n$ that, in the convex case, converges to the global minimum of f (if it exists). In the nonconvex case, since the optimality condition of Theorem 2.3.5 is not sufficient without some convexity assumptions, the method is only guaranteed to find a stationary point of the objective function. We assume that at each point $\mathbf{x} \in \mathbb{R}^n$ we can evaluate a value of the objective function $f(\mathbf{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$.

3.2.1 Direction Finding

In this subsection, we describe how to find a search direction $\mathbf{d}_k \in \mathbb{R}^n$ in the iteration k for a locally Lipschitz continuous objective function. We assume that in addition to the current iteration point $\mathbf{x}_k$ we have some auxiliary points $\mathbf{y}_j \in \mathbb{R}^n$ from previous iterations and subgradients $\boldsymbol{\xi}_j \in \partial f(\mathbf{y}_j)$ for $j \in \mathcal{J}_k$, where the index set $\mathcal{J}_k$ is a nonempty subset of $\{1, \ldots, k\}$.

In (2.3) we have defined the linearization of the objective function f at $\mathbf{x}$. However, for this representation we need to know the whole subdifferential $\partial f(\mathbf{x})$, which is normally unknown. Thus, we have to approximate it somehow. By using the auxiliary points $\mathbf{y}_j \in \mathbb{R}^n$ and the subgradients $\boldsymbol{\xi}_j \in \partial f(\mathbf{y}_j)$ for $j \in \mathcal{J}_k$ we can define a convex piecewise linear approximation of the original objective function f by

$$\hat{f}_k(\mathbf{x}) = \max\{\, f(\mathbf{y}_j) + \boldsymbol{\xi}_j^T(\mathbf{x} - \mathbf{y}_j) \mid j \in \mathcal{J}_k \,\}, \tag{3.10}$$

which can be written in the equivalent form

$$\hat{f}_k(\mathbf{x}) = \max\{\, f(\mathbf{x}_k) + \boldsymbol{\xi}_j^T(\mathbf{x} - \mathbf{x}_k) - \alpha_j^k \mid j \in \mathcal{J}_k \,\},$$

where

$$\alpha_j^k = f(\mathbf{x}_k) - f(\mathbf{y}_j) - \boldsymbol{\xi}_j^T(\mathbf{x} - \mathbf{y}_j) \qquad \text{for all } j \in \mathcal{J}_k \tag{3.11}$$

is a so-called *linearization error*. If the function f is convex, the linearization error $\alpha_j^k \geq 0$ for all $j \in \mathcal{J}_k$ and $f(\mathbf{x}) \geq \hat{f}_k(\mathbf{x})$ (see, e.g., Mäkelä and Neittaanmäki (1992)).

For nonconvex functions the linearization error (3.11) can be negative. We use so-called *subgradient locality measures* to generalize the linearization errors for nonconvex functions:

$$\beta_j^k = \max\{\, |\alpha_j^k|, \gamma(s_j^k)^\omega \,\} \geq 0, \tag{3.12}$$

where $j \in \mathcal{J}_k$, $\gamma \geq 0$ ($\gamma = 0$ if f is convex), $\omega \geq 1$ and s_j^k is a so-called *distance measure* such that

$$s_j^k = \begin{cases} \|\mathbf{x}_j - \mathbf{y}_j\| + \sum_{i=j}^{k-1} \|\mathbf{x}_{i+1} - \mathbf{x}_i\| & \text{for } j = 1, \ldots, k-1, \\ \|\mathbf{x}_k - \mathbf{y}_j\| & \text{for } j = k. \end{cases}$$

For convex objective functions the subgradient locality measures and the linearization errors coincide (Mäkelä and Neittaanmäki (1992)).

The linearization $\hat{f}_k$ is not suitable itself for determining a new approximation of the function f in order to find its minimum. The minimum of $\hat{f}_k$ may not exist, since the function $\hat{f}_k$ is piecewise linear and if it exists, it can be too far from the minimum of the objective function f. For this reason, a stabilizing term $\frac{1}{2}(\mathbf{x}-\mathbf{x}_k)^T G_k(\mathbf{x}-\mathbf{x}_k)$, which keeps the approximation local enough, has to be added to function $\hat{f}_k$. Thus we obtain a piecewise quadratic function

$$f_Q^k(\mathbf{x}) = \tfrac{1}{2}(\mathbf{x}-\mathbf{x}_k)^T G_k(\mathbf{x}-\mathbf{x}_k) + \hat{f}_k(\mathbf{x}),$$

where G_k is a nonsingular and symmetric $n \times n$ matrix. Some details of the choice of the matrix G_k are to be presented in Subsection 3.2.4.

The search direction $\mathbf{d}_k$ can be determined by minimizing the piecewise quadratic function $f_Q^k(\mathbf{x})$. This minimization problem is equivalent (see, e.g., Mäkelä (2002)) to the quadratic subproblem (minimization proceeds over $\mathbf{d}$ and v)

$$\begin{cases} \text{minimize} & \tfrac{1}{2}\mathbf{d}^T G_k \mathbf{d} + v \\ \text{subject to} & -\beta_j^k + \mathbf{d}^T \boldsymbol{\xi}_j \le v \quad \text{for all } j \in \mathcal{J}_k. \end{cases} \tag{3.13}$$

By duality this is equivalent to finding Lagrange multipliers λ_j^k for $j \in \mathcal{J}_k$ that solve the quadratic dual problem

$$\begin{cases} \text{minimize} & \tfrac{1}{2}\left(\sum_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j\right)^T G_k^{-1} \left(\sum_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j\right) + \sum_{j\in\mathcal{J}_k} \lambda_j \beta_j^k \\ \text{subject to} & \sum_{j\in\mathcal{J}_k} \lambda_j = 1 \quad \text{and} \\ & \lambda_j \ge 0, \quad \text{for all } j \in \mathcal{J}_k. \end{cases} \tag{3.14}$$

The solution of the problem (3.13) can be expressed in the form (see, e.g., Mäkelä (2002))

$$\begin{aligned} \mathbf{d}_k &= -\sum_{j\in\mathcal{J}_k} \lambda_j^k G_k^{-1} \boldsymbol{\xi}_j \qquad \text{and} \\ v_k &= -\mathbf{d}_k^T G_k \mathbf{d}_k - \sum_{j\in\mathcal{J}_k} \lambda_j^k \beta_j^k. \end{aligned}$$

The direction finding problem (3.13) seems now to be suitable for generating a descent direction. However, we still have to decide how to choose the index set $\mathcal{J}_k$. As mentioned in the beginning of this subsection, the index set must be a nonempty subset of $\{1, \dots, k\}$. Thus, the simplest way might be to choose directly

$$\mathcal{J}_k = \{1, \dots, k\}.$$

However, in practice this strategy imposes serious problems with storage and computations after a large number of iterations. So, in practice the size of the index set $\mathcal{J}_k$ have to be bounded, that is, $|\mathcal{J}_k| \leq m_\xi$, where $|\mathcal{J}_k|$ is the number of the elements in the set $\mathcal{J}_k$. The set $\mathcal{J}_k$ is usually determined such that if the iteration number $k \leq m_\xi$, then the index set is chosen directly

$$\mathcal{J}_k = \{1, \dots, k\}$$

and if $k > m_\xi$, then the index set is chosen such that

$$\mathcal{J}_k = \mathcal{J}_{k-1} \cup \{k\} \setminus \{k - m_\xi\}.$$

If $\mathcal{J}_k \neq \{1, \dots, k\}$, one possibility to guarantee the global convergence of the bundle method is to use the following *subgradient aggregation strategy* (see, e.g., Kiwiel (1985)). We define the *aggregate values* f_a^k, $\boldsymbol{\xi}_a^k$ and s_a^k as follows. At the first iteration let $\mathbf{x}_1 \in \mathbb{R}^n$ be a starting point supplied by the user. Then, we initialize the algorithm by

$$\begin{aligned}
&\mathbf{y}_1 = \mathbf{x}_1, \\
&f_a^1 = f_1^1 = f(\mathbf{y}_1), \\
&\boldsymbol{\xi}_a^1 = \boldsymbol{\xi}_1 \in \partial f(\mathbf{y}_1), \\
&s_1^1 = s_a^1 = 0 \qquad \text{and} \\
&\mathcal{J}_1 = \{1\}.
\end{aligned}$$

At iteration $k+1$ the new aggregate values f_a^{k+1}, $\boldsymbol{\xi}_a^{k+1}$ and s_a^{k+1} are defined by the formulae

$$\begin{aligned}
f_a^{k+1} &= \tilde{f}_a^k + (\mathbf{x}_{k+1} - \mathbf{x}_k)^T \tilde{\boldsymbol{\xi}}_a^k, \\
\boldsymbol{\xi}_a^{k+1} &= \tilde{\boldsymbol{\xi}}_a^k \qquad \text{and} \\
s_a^{k+1} &= \tilde{s}_a^k + \|\mathbf{x}_{k+1} - \mathbf{x}_k\|,
\end{aligned}$$

where the values of $\tilde{f}_a^k$, $\tilde{\boldsymbol{\xi}}_a^k$ and $\tilde{s}_a^k$ are obtained by solving the Lagrange multipliers λ_j^k for $j \in \mathcal{J}_k$ and λ_a^k of the direction finding problem given below. However, first we have to define the *aggregate locality measure* by

$$\beta_a^k = \max\{\, |f(\mathbf{x}_k) - f_a^k|,\, \gamma (s_a^k)^\omega \,\}.$$

Now, by using these aggregate values, the search direction $\mathbf{d}_k$ can be determined by minimizing the quadratic subproblem (minimization proceeds over $\mathbf{d}$ and v)

$$\begin{cases} \text{minimize} & \frac{1}{2}\mathbf{d}^T G_k \mathbf{d} + v \\ \text{subject to} & -\beta_j^k + \mathbf{d}^T \boldsymbol{\xi}_j \leq v \quad \text{for all } j \in \mathcal{J}_k \text{ and} \\ & -\beta_a^k + \mathbf{d}^T \boldsymbol{\xi}_a^k \leq v, \end{cases} \tag{3.15}$$

which by duality is equivalent to finding Lagrange multipliers λ_j^k for $j \in \mathcal{J}_k$ and λ_a^k that solve the problem

$$\begin{cases} \text{minimize} & \frac{1}{2}\left(\sum_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j + \lambda_a \boldsymbol{\xi}_a^k\right)^T G_k^{-1} \left(\sum_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j + \lambda_a \boldsymbol{\xi}_a^k\right) \\ & + \sum_{j\in\mathcal{J}_k} \lambda_j \beta_j^k + \lambda_a \beta_a^k \\ \text{subject to} & \sum_{j\in\mathcal{J}_k} \lambda_j + \lambda_a = 1, \\ & \lambda_j \geq 0, \quad \text{for all } j \in \mathcal{J}_k \text{ and} \\ & \lambda_a \geq 0. \end{cases} \tag{3.16}$$

Then, the solution of the problem (3.15) can be expressed in the form (see, e.g., Lukšan and Vlček (2000a))

$$\begin{aligned} \mathbf{d}_k &= -G_k^{-1}\tilde{\boldsymbol{\xi}}_a^k \qquad \text{and} \\ v_k &= -\mathbf{d}_k^T G_k \mathbf{d}_k - \tilde{\beta}_a^k, \end{aligned}$$

where

$$\begin{aligned} \tilde{\boldsymbol{\xi}}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k \boldsymbol{\xi}_j + \lambda_a^k \boldsymbol{\xi}_a^k \qquad \text{and} \\ \tilde{\beta}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k \beta_j^k + \lambda_a^k \beta_a^k. \end{aligned}$$

In addition to the values $\tilde{\boldsymbol{\xi}}_a^k$ and $\tilde{\beta}_a^k$, the aggregate values

$$\begin{aligned} \tilde{f}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k f_j^k + \lambda_a^k f_a^k \qquad \text{and} \\ \tilde{s}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k s_j^k + \lambda_a^k s_a^k \end{aligned}$$

are defined by using the Lagrange multipliers $\lambda_j^k \geq 0$ for $j \in \mathcal{J}_k$ and $\lambda_a^k \geq 0$.

3.2.2 Line Search

When the direction vector $\mathbf{d}_k$ has been determined, we need to calculate a new approximation to the minimum of the objective function. To guarantee the global convergence of the bundle method it is not possible to simply set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$, but it is necessary to use some line search procedure which generates two points

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}_k \qquad \text{and} \\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}_k, \end{aligned}$$

where $0 \leq t_L^k \leq t_R^k \leq 1$ are step sizes. There exist several different line search procedures suitable for bundle methods (see, e.g., Kiwiel (1985)). In this work, we describe one of them, since it is suitable also for the variable metric bundle methods to be described in the next section.

First, we search for a largest number $t_L^k \in [0,1]$ such that (compare with (3.2))

$$t_L^k > 0 \qquad \text{and} \qquad f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \leq -\varepsilon_L t_L^k w_k, \tag{3.17}$$

where $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k > 0$ is the predicted descent of f at $\mathbf{x}_k$ (the parameter w_k is used also as a stopping parameter and we give some details of it in next section). If such a step size exists we take a *serious step*:

$$t_R^k = t_L^k, \qquad \text{and} \qquad \mathbf{x}_{k+1} = \mathbf{y}_{k+1}.$$

Otherwise, we take a *null step* if (compare with (3.3))

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \mathbf{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{3.18}$$

where $\varepsilon_R \in (\varepsilon_L, 1)$ is a fixed line search parameter and

$$\beta_{k+1} = \max\{\, |f(\mathbf{x}_k) - f(\mathbf{y}_{k+1}) + (\mathbf{y}_{k+1} - \mathbf{x}_k)^T \boldsymbol{\xi}_{k+1}|,\, \gamma \|\mathbf{y}_{k+1} - \mathbf{x}_k\|^{\omega} \,\}.$$

In the case of a null step we set

$$\mathbf{x}_{k+1} = \mathbf{x}_k \qquad \text{and} \qquad \mathbf{y}_{k+1} = \mathbf{x}_k + t_R^k \mathbf{d}_k.$$

In the case of a serious step there occurs a significant decrease of the objective function. For this reason, there is no need to detect the discontinuities in the gradient and, thus, we set $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{x}_{k+1})$. If we take a null step, we do not update the actual solution ($\mathbf{x}_{k+1} = \mathbf{x}_k$) but information about the objective function is increased. In null steps there exists a discontinuity in the gradient of f. In this case, the last requirement in (3.18) ensures that $\mathbf{x}_k$ and $\mathbf{y}_{k+1}$ lie on the opposite sides of this discontinuity and the new subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1})$ will force a significant modification to the next direction finding problem.

In the convex case, the line search procedure can be replaced by a simple step size selection, which is either accepted (serious step) or not (null step) (see, e.g., Mäkelä (2002)).

3.2.3 Stopping Criterion

For smooth functions, a necessary condition for a local minimum is that the gradient has to be zero and by continuity it becomes small when we are close to an optimal point. This is no longer true when we replace the gradient by an arbitrary subgradient. Due to subgradient aggregation we have quite a useful approximation to the gradient, namely the aggregate subgradient $\tilde{\boldsymbol{\xi}}_a^k$. However, the direct test $\|\tilde{\boldsymbol{\xi}}_a^k\| < \varepsilon$ for some $\varepsilon > 0$ is too uncertain, if the current piecewise linear approximation is too rough. Therefore, we use the stabilizing matrix G_k and the aggregate subgradient locality measure $\tilde{\beta}_a^k$ to improve the accuracy of the norm of the aggregate subgradient. The aggregate subgradient locality measure $\tilde{\beta}_a^k$ approximates the accuracy of the current linearization: If the value of the locality measure is large, then the linearization is rough and if it is near zero then the linearization is quite accurate and we can stop the algorithm if the norm of the aggregate subgradient is small. Thus, the stopping parameter at iteration k is defined by (see, e.g., Lukšan and Vlček (2000a))

$$w_k = \tfrac{1}{2}(\tilde{\boldsymbol{\xi}}_a^k)^T (G_k)^{-1} \tilde{\boldsymbol{\xi}}_a^k + \tilde{\beta}_a^k,$$

and the stopping criterion is

$$\text{If } w_k < \varepsilon, \text{ for given } \varepsilon > 0, \text{ then stop.}$$

3.2.4 The Choice of Matrix G_k

The last open question is how to choose the stabilizing matrix G_k. As mentioned in Subsection 3.2.1, all the matrices G_k are supposed to be nonsingular and symmetric. In addition, to ensure the global convergence of the bundle method, we have to assume that all the matrices G_k are positive definite and bounded. Moreover, if the k-th step is a null step, then we assume that $h^T G_{k+1}^{-1} h \leq h^T G_k^{-1} h$, for all $h \in \mathbb{R}^n$. These assumptions are relative strong, but they can be weakened for different versions of bundle methods.

There exist several versions of bundle methods that differs mainly with the choice of matrix the G_k. For example, in the most frequently used *proximal bundle* method (see, e.g., Kiwiel (1990) and Mäkelä and Neittaanmäki (1992)), the matrix G_k is diagonal and of the form $G_k = u_k I$. The restrictions given above are satisfied if weights $u_k > 0$ lie in the compact interval that does contain zero and $u_{k+1} \geq u_k$ is valid in the null step. A thorough overview of various bundle methods is given in Mäkelä (2002).

3.2.5 Algorithm

Now we present a general algorithm for bundle methods. In addition to the algorithm given below, the line search algorithm (Algorithm 3.3, p. 32) is also needed.

Algorithm 3.2. (Bundle Method).

Data: Choose a final accuracy tolerance $\varepsilon > 0$, positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$, a distance measure parameter $\gamma > 0$, a locality measure parameter $\omega \geq 1$ and the maximum number of stored subgradients $m_\xi \geq 1$.

Step 0: (*Initialization.*) Choose a starting point $\mathbf{x}_1 \in \mathbb{R}^n$ and a symmetric, positive definite matrix G_1 (for example $G_1 = I$). Set

$$\mathbf{y}_1 = \mathbf{x}_1$$

and compute

$$\begin{aligned} &f_1 = f(\mathbf{y}_1) \qquad \text{and} \\ &\boldsymbol{\xi}_1 \in \partial f(\mathbf{y}_1). \end{aligned}$$

Set

$$s_1^1 = s_a^1 = 0, \qquad f_1^1 = f_a^1 = f_1, \qquad \boldsymbol{\xi}_a^1 = \boldsymbol{\xi}_1$$

and $\mathcal{J}_1 = \{1\}$. Set the iteration counter $k = 1$.

Step 1: (*Direction finding*) Determine multipliers $\lambda_j^k,\ j \in \mathcal{J}_k$ and λ_a^k ($\lambda_a^k \neq 0$ only if $\mathcal{J}_k \neq \{1, \dots, k\}$), aggregate values $\tilde{\boldsymbol{\xi}}_a^k,\ \tilde{\beta}_a^k,\ \tilde{f}_a^k,\ \tilde{s}_a^k$, a direction vector $\mathbf{d}_k$ and a value v_k by solving the quadratic optimization problem (3.15) (the last constraint is used only if $\mathcal{J}_k \neq \{1, \dots, k\}$).

Step 2: (*Stopping criterion*) Set

$$w_k = \tfrac{1}{2}(\tilde{\boldsymbol{\xi}}_a^k)^T (G_k)^{-1} \tilde{\boldsymbol{\xi}}_a^k + \tilde{\beta}_a^k.$$

If $w_k \leq \varepsilon$, then stop.

Step 3: (*Line search*) Determine step sizes $t_L^k \in [0,1]$ and $t_R^k \in [t_L^k, 1]$ by the line search Algorithm 3.3. Set

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}^k, \\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}^k \end{aligned}$$

and compute

$$\begin{aligned} & f_{k+1} = f(\mathbf{y}_{k+1}) \qquad \text{and} \\ & \boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1}). \end{aligned}$$

Step 4: (*Linearization Updating*) Calculate the linearization values

$$\begin{aligned} f_j^{k+1} &= f_j^k + (\mathbf{x}_{k+1} - \mathbf{x}_k)^T \boldsymbol{\xi}_j, \quad \text{for } j \in \mathcal{J}_k, \\ f_a^{k+1} &= \tilde{f}_a^k + (\mathbf{x}_{k+1} - \mathbf{x}_k)^T \tilde{\boldsymbol{\xi}}_a^k, \\ f_{k+1}^{k+1} &= f_{k+1} + (\mathbf{x}_{k+1} - \mathbf{y}_{k+1})^T \boldsymbol{\xi}_{k+1}, \\ \boldsymbol{\xi}_a^{k+1} &= \tilde{\boldsymbol{\xi}}_a^k, \\ s_j^{k+1} &= s_j^k + \|\mathbf{x}_{k+1} - \mathbf{x}_k\|, \quad \text{for } j \in \mathcal{J}_k, \\ s_a^{k+1} &= \tilde{s}_a^k + \|\mathbf{x}_{k+1} - \mathbf{x}_k\|, \\ s_{k+1}^{k+1} &= \|\mathbf{y}_{k+1} - \mathbf{x}_{k+1}\|. \end{aligned}$$

Step 5: (*Matrix Updating*) Determine the matrix G_{k+1} satisfying the assumptions discussed in Subsection 3.2.4.

Step 6: (*Updating*) If $|\mathcal{J}_k| < m_\xi$, then set

$$\mathcal{J}_{k+1} = \mathcal{J}_k \cup \{k+1\}.$$

If $|\mathcal{J}_k| = m_\xi$, then set

$$\mathcal{J}_{k+1} = \mathcal{J}_k \cup \{k+1\} \setminus \{k+1-m_\xi\}.$$

Set $k = k+1$ and go to Step 1.

Under mild assumptions it can be proved that the number of consecutive null steps in Algorithm 3.2 is finite and that every cluster point in the sequence $(\mathbf{x}_k)$ is a stationary point of the objective function (see, e.g., Kiwiel (1985)).

Note that Algorithm 3.2 requires relatively large bundles ($m_\xi \sim n$) to be computationally efficient and, thus, the quadratic subproblem (3.15) is time-consuming.

For further study of bundle methods we refer to Kiwiel (1985), Mäkelä and Neittaanmäki (1992), Hiriart-Urruty and Lemaréchal (1993) and Lukšan and Vlček (2000a).

3.3 Variable Metric Bundle Methods

In this section, we present variable metric bundle methods by Lukšan and Vlček (Lukšan and Vlček (1999a), Vlček and Lukšan (1999)). The methods presented are hybrids of the variable metric method (Section 3.1) and the bundle method (Section 3.2). We present methods for both convex and nonconvex nonsmooth unconstrained optimization problems. As the bundle methods the variable metric bundle methods produce a sequence $(\mathbf{x}_k) \in \mathbb{R}^n$ which, in the convex case, converges to the global minimum of the objective function f (if it exists). In the nonconvex case, the method is only guaranteed to find a stationary point of the objective function. We assume that at each point $\mathbf{x} \in \mathbb{R}^n$ we can evaluate a value of the objective function $f(\mathbf{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$.

The basic idea of variable metric bundle methods for nonsmooth optimization is to use some properties of bundle methods to improve the robustness and efficiency of the variable metric method. Furthermore, the time-consuming quadratic subproblem (3.15) of bundle methods need not to be solved.

There are three main differences between variable metric bundle methods and standard variable metric methods. The first difference is the use of null steps. This admits obtaining sufficient information of the nonsmooth objective function in the cases the descent condition (3.17) is not satisfied. The other differences are a simple aggregation of subgradients and the application of locality measures (3.12). These modifications guarantee the convergence of the aggregate subgradients to zero and make it possible to evaluate a termination criterion.

3.3.1 Direction Finding

In this subsection, we describe how to find a search direction $\mathbf{d}_k$ by using the variable metric bundle methods. The basic idea in direction finding is the same as with standard variable metric methods (see Section 3.1) and the approximations D_k of the inverse of the Hessian matrix are formed by using the usual variable metric updates.

However, due to the use of null steps some modifications have to be made: After we have taken a null step, the approximation D_k of the inverse of the Hessian matrix is formed by using the symmetric-rank-one (SR1) update (3.8), since this update formula preserves the boundedness of generated matrices as required for the proof of global convergence (see Lukšan and Vlček (1999a), Vlček and Lukšan (1999)). In addition, we use an aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ to calculate the search direction

$$\mathbf{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k.$$

Because the boundedness of the generated matrices is not required after a serious step, the more efficient standard Broyden-Fletcher-Goldfarb-Shanno (BFGS) update (3.7) is used together with the original subgradient $\boldsymbol{\xi}_k$ (note that after a serious step $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$). Thus, after a serious step, the search direction $\mathbf{d}_k$ is defined by

$$\mathbf{d}_k = -D_k\boldsymbol{\xi}_k.$$

3.3.2 Selection of Step Sizes

In this subsection, we consider how to calculate a new iteration point $\mathbf{x}_{k+1}$ when the search direction $\mathbf{d}_k$ has been calculated. Similarly to bundle methods we use a procedure that generates two points: a new iteration point $\mathbf{x}_{k+1}$ and a new auxiliary point $\mathbf{y}_{k+1}$.

In convex case, there is no need to use any two-point line search procedure as in standard bundle methods (see Subsection 3.2.2). Thus, the determination of the auxiliary points $\mathbf{y}_k$ differs from both standard bundle methods and variable metric bundle methods for nonconvex but locally Lipschitz continuous functions. In a convex case, the auxiliary points $\mathbf{y}_k$ are determined by

$$\begin{aligned}\mathbf{y}_1 &= \mathbf{x}_1,\\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_k\mathbf{d}_k, \qquad \text{for all } k \geq 1,\end{aligned}$$

where $t_k \in [t_{min}, t_{max}]$ is an appropriately chosen step size (for details, see Lukšan and Vlček (1999a)) and $t_{max} > 1$ and $t_{min} \in (0, 1)$ are the upper and lower bound for the step size t_k. An essential condition for a serious step to be taken is to have (compare with (3.17))

$$f(\mathbf{y}_{k+1}) \leq f(\mathbf{x}_k) - \varepsilon_L t_k w_k, \tag{3.19}$$

where $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k > 0$ represents the desirable amount of descent. The parameter w_k is also used as a stopping parameter (see, e.g., Vlček and Lukšan (1999)). If the required condition (3.19) is satisfied, then

$$\mathbf{x}_{k+1} = \mathbf{y}_{k+1}$$

and a serious step is taken. Otherwise, a null step is taken, that is, $\mathbf{x}_{k+1} = \mathbf{x}_k$ but information about the objective function is increased.

In a nonconvex case, the auxiliary points $\mathbf{y}_k$ are determined quite similarly to standard bundle methods (see Subsection 3.2.2), that is

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{x}_1, \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}_k, \qquad \text{for all } k \geq 1, \\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}_k, \qquad \text{for all } k \geq 1, \end{aligned}$$

where $t_R^k \in (0, t_I^k]$, $t_L^k \in [0, t_R^k]$ are appropriately chosen step sizes, $t_I^k \in [t_{min}, t_{max})$ is an initial step size and $t_{max} > 1$ and $t_{min} \in (0, 1)$ are the upper and lower bound for the initial step size t_I^k, respectively. The possibility of using step sizes greater than 1 is useful here, because the information about the objective function f, included in matrix D_k, is not sufficient for a proper step size determination in the nonsmooth case. Even if the selection of the initial step sizes t_I^k is not needed for proving the global convergence, the efficiency of the algorithm is very sensitive to how it is realized. A detailed description of the choice of the initial step size can be found in Vlček and Lukšan (1999).

An essential condition for a serious step to be taken in the case of a nonconvex objective function is similar to that of bundle methods (see (3.17)). That is,

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \leq -\varepsilon_L t_L^k w_k, \tag{3.20}$$

where, as in the convex case, $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k > 0$ represents the desirable amount of descent. If the required condition (3.20) is satisfied, then

$$\mathbf{x}_{k+1} = \mathbf{y}_{k+1}$$

and a serious step is taken. Also a condition for a null step to be taken is similar to that of bundle methods (see (3.18)). Thus, a null step is taken, if

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \mathbf{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{3.21}$$

where $\varepsilon_R \in (\varepsilon_L, 1)$ is a fixed line search parameter and β_{k+1} is the locality measure similar to bundle methods (see (3.12)). In the case of a null step

$$\mathbf{x}_{k+1} = \mathbf{x}_k$$

but information about the objective function is increased.

Next we present a line search algorithm, which is used to determine the ultimate step sizes t_L^k and t_R^k in variable metric bundle methods for nonconvex locally Lipschitz functions (see Vlček and Lukšan (1999)). The algorithm given below can also be used with standard bundle methods.

Suppose that we have positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$, a distance measure parameter $\gamma \geq 0$ ($\gamma = 0$ if f is convex), a locality measure parameter $\omega \geq 1$ and the desirable amount of descent w_k. In addition, we have a lower bound for serious steps $t_{min} \in (0, 1)$ and the initial step size t_I^k. Note that in bundle methods the initial step size t_I^k is fixed to 1.

Algorithm 3.3. (Line Search).

Step 0: (*Initialization.*) Set $t_A = 0$ and $t = t_U = t_I^k$. Choose $\kappa \in (0, 1/2)$, $\varepsilon_A \in (0, \varepsilon_R - \varepsilon_L)$ and $\varepsilon_T \in (\varepsilon_L, \varepsilon_R - \varepsilon_A)$.

Step 1: (*New values.*) Compute $f(\mathbf{x}_k + t\mathbf{d}_k)$, $\boldsymbol{\xi} \in \partial f(\mathbf{x}_k + t\mathbf{d}_k)$ and

$$\beta = \max\{\, |f(\mathbf{x}_k) - f(\mathbf{x}_k + t\mathbf{d}_k) + t\mathbf{d}_k^T \boldsymbol{\xi})|,\ \gamma\,(t\|\mathbf{d}_k\|)^\omega \,\}.$$

If $f(\mathbf{x}_k + t\mathbf{d}_k) \leq f(\mathbf{x}_k) - \varepsilon_T t w_k$, then set $t_A = t$. Otherwise set $t_U = t$.

Step 2: (*Serious Step.*) If

$$f(\mathbf{x}_k + t\mathbf{d}_k) \leq f(\mathbf{x}_k) - \varepsilon_L t w_k$$

and either

$$t \geq t_{min} \qquad \text{or} \qquad \beta > \varepsilon_A w_k,$$

then set $t_R^k = t_L^k = t$ and terminate the computation.

Step 3: (*Null Step.*) If

$$-\beta + \mathbf{d}_k^T \boldsymbol{\xi} \geq -\varepsilon_R w_k,$$

then set $t_R^k = t$, $t_L^k = 0$ and terminate the computation.

Step 4: (*Interpolation.*) Choose

$$t \in [t_A + \kappa(t_U - t_A), t_U - \kappa(t_U - t_A)]$$

by using some interpolation procedure and go to Step 1.

It can be proved under some semi-smoothness hypothesis (see, e.g., Vlček and Lukšan (1999)) that the Algorithm 3.3 terminates in a finite number of iterations, finding step sizes t_L^k and t_R^k satisfying $f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \leq -\varepsilon_L t_L^k w_k$ and, in case of $t_L^k = 0$ (null step), also (3.21).

3.3.3 Aggregation

In this subsection, we briefly describe the simple aggregation procedure used with the variable metric bundle methods.

The aggregation procedure uses three subgradients and two locality measures. We denote by m the lowest index j satisfying $\mathbf{x}_j = \mathbf{x}_k$ (index of the iteration after the latest serious step). If we have the basic subgradient $\boldsymbol{\xi}_m \in \partial f(\mathbf{x}_k)$, the auxiliary subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1})$ and the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$, then the next aggregate subgradient $\tilde{\boldsymbol{\xi}}_{k+1}$ is defined as a convex combination of these three subgradients:

$$\tilde{\boldsymbol{\xi}}_{k+1} = \lambda_1^k \boldsymbol{\xi}_m + \lambda_2^k \boldsymbol{\xi}_{k+1} + \lambda_3^k \tilde{\boldsymbol{\xi}}_k.$$

In addition, if we have the current locality measure β_{k+1} and the current aggregate locality measure $\tilde{\beta}_k$, then the next aggregate locality measure $\tilde{\beta}_{k+1}$ is defined as a convex combination of these two locality measures:

$$\tilde{\beta}_{k+1} = \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k.$$

The multipliers $\lambda_i^k \geq 0$ for $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ can be determined by minimizing a quadratic function φ that depends on the three subgradients and the two locality measures mentioned above. The quadratic function φ is defined by

$$\begin{aligned}\varphi(\lambda_1, \lambda_2, \lambda_3) = {} & (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)^T D_k (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k) \\ & + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k).\end{aligned} \tag{3.22}$$

Thus, the quadratic subproblem (3.15) appearing in standard bundle methods has been reduced to the minimization of the simple function (3.22). Note that the aggregate values are computed only if the last step was a null step. Otherwise, we set $\tilde{\boldsymbol{\xi}}_{k+1} = \boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{x}_{k+1})$ and $\tilde{\beta}_{k+1} = 0$.

3.3.4 Algorithm

Now we present a variable metric bundle algorithm. The algorithm is suitable for minimizing both convex and nonconvex but locally Lipchitz objective functions. Note that it can still be simplified a little for convex functions.

Algorithm 3.4. (Variable Metric Bundle Method).

Data: Select positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$. Choose a final accuracy tolerance $\varepsilon \geq 0$, a distance measure parameter $\gamma \geq 0$ ($\gamma = 0$ if f is convex) and a locality measure parameter $\omega \geq 1$.

Step 0: (*Initialization.*) Choose the starting point $\mathbf{x}_1 \in \mathbb{R}^n$ and the positive definite matrix D_1 (e.g., $D_1 = I$). Set

$$\begin{aligned} &\mathbf{y}_1 = \mathbf{x}_1 \qquad \text{and} \\ &\beta_1 = 0. \end{aligned}$$

Compute

$$\begin{aligned} &f_1 = f(\mathbf{x}_1) \qquad \text{and} \\ &\boldsymbol{\xi}_1 \in \partial f(\mathbf{x}_1). \end{aligned}$$

Set the iteration counter $k = 1$.

Step 1: (*Serious step initialization.*) Set the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$ and the aggregate subgradient locality measure $\tilde{\beta}_k = 0$. Set the index variable for null steps $m = k$.

Step 2: (*Stopping criterion.*) Set

$$w_k = \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k.$$

If $w_k \leq \varepsilon$, then stop.

Step 3: (*Line search.*) Set

$$\mathbf{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k.$$

(Note that after a serious step, we have $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$).

Determine step sizes t_L^k and t_R^k by the line search Algorithm 3.3 to take either a serious step or a null step (depending on whether (3.20) or (3.21) holds). Calculate the corresponding values

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}_k,\\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}_k,\\ f_{k+1} &= f(\mathbf{x}_{k+1}),\\ \boldsymbol{\xi}_{k+1} &\in \partial f(\mathbf{y}_{k+1}).\end{aligned}$$

Set $\mathbf{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$. If $t_L^k > 0$ (serious step), set $\beta_{k+1} = 0$ and go to Step 6. Otherwise, calculate the locality measure β_{k+1} (see (3.12)).

Step 4: (*Aggregation.*) Determine multipliers $\lambda_i^k \geq 0$, $i \in \{1,2,3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function

$$\begin{aligned}\varphi(\lambda_1,\lambda_2,\lambda_3) = {} & (\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k)^T D_k(\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k)\\ & + 2(\lambda_2\beta_{k+1} + \lambda_3\tilde{\beta}_k).\end{aligned}$$

Set

$$\begin{aligned}\tilde{\boldsymbol{\xi}}_{k+1} &= \lambda_1^k\boldsymbol{\xi}_m + \lambda_2^k\boldsymbol{\xi}_{k+1} + \lambda_3^k\tilde{\boldsymbol{\xi}}_k \qquad \text{and}\\ \tilde{\beta}_{k+1} &= \lambda_2^k\beta_{k+1} + \lambda_3^k\tilde{\beta}_k.\end{aligned}$$

Step 5: (*SR1 update.*) Let $\mathbf{v}_k = D_k\mathbf{u}_k - t_R^k\mathbf{d}_k$. If $\tilde{\boldsymbol{\xi}}_k^T\mathbf{v}_k < 0$, then set

$$D_{k+1} = D_k - \frac{\mathbf{v}_k\mathbf{v}_k^T}{\mathbf{u}_k^T\mathbf{v}_k},$$

otherwise set $D_{k+1} = D_k$. Set $k = k+1$ and go to Step 2.

Step 6: (*BFGS update.*) If $\mathbf{u}_k^T\mathbf{d}_k > 0$, then set

$$D_{k+1} = D_k + \left(t_L^k + \frac{\mathbf{u}_k^T D_k\mathbf{u}_k}{\mathbf{u}_k^T\mathbf{d}_k}\right)\frac{\mathbf{d}_k\mathbf{d}_k^T}{\mathbf{u}_k^T\mathbf{d}_k} - \frac{D_k\mathbf{u}_k\mathbf{d}_k^T + \mathbf{d}_k\mathbf{u}_k^T D_k}{\mathbf{u}_k^T\mathbf{d}_k}.$$

Otherwise, set $D_{k+1} = D_k$. Set $k = k+1$ and go to Step 1.

In a convex case, the line search procedure in Step 3 can be replaced by a simple step size selection, which is either accepted (serious step) or not (null step). In the case of a null step, the value β_{k+1} should be the linearization error $f(\mathbf{x}_k) - f(\mathbf{y}_{k+1}) + t_k\mathbf{d}_k^T\boldsymbol{\xi}_{k+1}$ corresponding to bundle methods. However, this leads to theoretical difficulties when the step size t_k is greater than 1 and, therefore, the linearization error is divided by t_k. In a nonconvex case, the locality measure similar to bundle methods (3.12) is used.

The condition

$$\tilde{\boldsymbol{\xi}}_k^T \mathbf{v}_k < 0,$$

in Step 5 (or $\mathbf{u}_k^T \mathbf{d}_k > t_R^k \mathbf{d}_k^T D_k^{-1} \mathbf{d}_k$), which implies that $\mathbf{u}_k^T \mathbf{v}_k > 0$, ensures the positive definiteness of the matrices D_{k+1} obtained by the SR1 update. Similarly, the condition

$$\mathbf{d}_k^T \mathbf{u}_k > 0$$

in Step 6 ensures the positive definiteness of the matrices D_{k+1} obtained by the BFGS update ($\mathbf{d}_k^T \mathbf{u}_k > 0$ holds whenever f is convex). Therefore, all the matrices generated by Algorithm 3.4 are positive definite.

Under mild assumptions it can be proved that in Algorithm 3.4 every cluster point in the sequence $(\mathbf{x}_k)$ is a stationary point of the objective function (see Lukšan and Vlček (1999a) and Vlček and Lukšan (1999)).

As mentioned in Subsection 3.3.3, the aggregation procedure uses only three subgradients and two locality measures to compute the new aggregate values. In practice, this means that the minimum size of the bundle m_ξ is 2 and a larger bundle is used only for the step size selection, which does not require time-consuming operations (see Lukšan and Vlček (1999a) and Vlček and Lukšan (1999)).

3.4 Bundle-Newton methods

Finally, we describe the main ideas of the second order bundle-Newton methods. The idea of the bundle-Newton method is very similar to that of bundle methods described in Section 3.2. For this reason, we only describe the most essential differences of these methods. For more details, we refer to Lukšan and Vlček (1998).

We assume that at each point $\mathbf{x} \in \mathbb{R}^n$ we can evaluate, in addition to the value of the function $f(\mathbf{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$, also an $n \times n$ symmetric matrix $B(\mathbf{x})$ approximating the Hessian matrix $H(\mathbf{x})$ of the objective function. For example, at the kink point $\mathbf{x}$ (that is the point where the function fails to be (twice) differentiable) of a piecewise twice differentiable function, we can set $B(\mathbf{x}) = H(\mathbf{y})$, where $\mathbf{y}$ belongs to an arbitrarily small neighborhood of $\mathbf{x}$.

Instead of piecewise linear approximation (3.10) of f we use a piecewise quadratic approximation of the form

$$\begin{aligned}\tilde{f}_k(\mathbf{x}) &= \max\{f(\mathbf{y}_j) + \boldsymbol{\xi}_j^T(\mathbf{x}-\mathbf{y}_j) + \frac{1}{2}\varrho_j(\mathbf{x}-\mathbf{y}_j)^T B_j(\mathbf{x}-\mathbf{y}_j) \mid j \in \mathcal{J}_k\,\} \\ &= \max\{f(\mathbf{x}_k) + \boldsymbol{\xi}_j^T(\mathbf{x}-\mathbf{x}_k) + \frac{1}{2}\varrho_j(\mathbf{x}-\mathbf{x}_k)^T B_j(\mathbf{x}-\mathbf{x}_k) - \alpha_j^k \mid j \in \mathcal{J}_k\,\},\end{aligned}$$

where $B_j = B(\mathbf{y}_j)$, $\varrho_j \in [0,1]$ is a *damping parameter* and, as before, $\mathbf{y}_j \in \mathbb{R}^n$ are some auxiliary points from previous iteration, $\boldsymbol{\xi}_j \in \partial f(\mathbf{y}_j)$ are the corresponding subgradients of those points and $\mathcal{J}_k$ is a nonempty subset of $\{1,...,k\}$. For all $j \in \mathcal{J}_k$ the linearization error α_j^k is defined as

$$\alpha_j^k = f(\mathbf{x}_k) - f(\mathbf{y}_j) - \boldsymbol{\xi}_j^T(\mathbf{x}_k-\mathbf{y}_j) - \frac{1}{2}\varrho_j(\mathbf{x}_k-\mathbf{y}_j)^T B_j(\mathbf{x}_k-\mathbf{y}_j). \qquad (3.23)$$

Note that now α_j^k may be negative even in the convex case. Therefore, the linearization error (3.23) is replaced by the subgradient locality measure (3.12) with the difference that now we have $\gamma > 0$ and so we preserve the property $\min_{\mathbf{x}\in\mathbb{R}^n} \tilde{f}_k(\mathbf{x}) \le f(\mathbf{x}_k)$ (see Lukšan and Vlček (1998)).

The search direction $\mathbf{d}_k$ can be determined as a solution of the problem

$$\begin{cases}\text{minimize} & \tilde{f}_k(\mathbf{x}_k+\mathbf{d}) \\ \text{such that} & \mathbf{d} \in \mathbb{R}^n.\end{cases} \qquad (3.24)$$

Since there already exist some second order information in the model, no regularizing quadratic term is required like in standard bundle methods (see Section 3.2). The problem (3.24) is in fact a nonlinear min-max problem, which can be solved approximately by the Lagrange-Newton method (see, e.g., Fletcher (1987)). Thus, we get a (smooth) quadratic optimization problem of finding the solution $(\mathbf{d}_k, v_k) \in \mathbb{R}^{n+1}$ of the problem

$$\begin{cases}\text{minimize} & \frac{1}{2}\mathbf{d}^T W_k \mathbf{d} + v \\ \text{subject to} & -\beta_j^k + \mathbf{d}^T\boldsymbol{\xi}_j^k \le v \quad \text{for all} \quad j \in \mathcal{J}_k,\end{cases} \qquad (3.25)$$

where β_j^k is the subgradient locality measure, $\boldsymbol{\xi}_j^k = \boldsymbol{\xi}_j + \varrho_j B_j(\mathbf{x}_k - \mathbf{y}_j)$,

$$W_k = \sum_{j\in\mathcal{J}_{k-1}} \lambda_j^{k-1}\varrho_j B_j$$

and λ_j^{k-1} for $j \in \mathcal{J}_{k-1}$ are the Lagrange multipliers of (3.25) from the previous iteration $k-1$. Since the method needs a positive definite matrix in (3.25), the matrix W_k is replaced by its positive definite modification when necessary (see Lukšan and Vlček (1998)).

The line search procedure of the bundle-Newton method differs from that given earlier for standard bundle methods and variable metric bundle methods. The line search algorithm used with bundle-Newton method is presented in Lukšan and Vlček (1998).

Under mild assumptions it can be proved that in bundle-Newton method every cluster point in the sequence $(\mathbf{x}_k)$ is a stationary point of the objective function (see Lukšan and Vlček (1998)).

4 Smooth Large-Scale Optimization

Many practical applications involve large dimensions. Efficient algorithms for small-scale problems do not necessarily translate into efficient algorithms in the large-scale setting, since large-scale problems often involve sparse matrices. For example, standard variable metric methods are unsuitable for large-scale problems, since they utilize dense approximations of the Hessian matrices. However, for variable metric methods, there exist three basic approaches to optimize smooth large-scale problems. The first idea consists of exploiting the structure of partially separable functions. This approach was initiated in Griewank and Toint (1982). The second approach is to preserve the sparsity pattern of the Hessian by special updates. This approach was proposed in Toint (1977). The third possibility is that of limited memory updating, in which only a few vectors are used and stored to represent the variable metric approximation of the Hessian matrix. This approach was first introduced in Nocedal (1980).

In this chapter, we present limited memory variable metric methods. In many cases, these methods are more useful than the two other approaches, since they do not require knowledge of the sparsity structure of the Hessian (sparse variable metric updates) and they ignore the structure of the problem (partitioned variable metric methods) (see, e.g., Nocedal (1997)). We give first the basic ideas of limited memory methods as they are given in Nocedal (1980). Then we present compact matrix representations to variable metric updates and at the end of this chapter we apply these compact matrix representations to the limited memory methods (see Byrd et al. (1994)). In Chapter 5, the same ideas are used to construct a new method for nonsmooth large-scale optimization.

In this chapter we assume that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function whose gradient $\nabla f(\mathbf{x})$ is available for all $\mathbf{x} \in \mathbb{R}^n$.

4.1 Limited Memory Variable Metric Methods

Various limited memory methods have been proposed in the literature; some of them combine conjugate gradient and quasi-Newton steps (see, e.g., Buckley and LeNir (1983)), and others are very closely related to quasi-Newton methods (see, e.g., Nocedal (1980) and Liu and Nocedal (1989)). The basic idea of limited memory methods is that the variable metric update of the approximated Hessian is not constructed explicitly. The updates use the information of the last few iterations to define a variable metric approximation. In practice, this means that the approximations of the Hessian matrices are not so accurate than those of standard variable metric methods but both the storage space required and the number of operations used are significantly smaller.

The most commonly used limited memory method is the limited memory BFGS method (L-BFGS). This method is very similar to the standard BFGS method (see Section 3.1). The only difference is the matrix update. Instead of storing matrices D_k approximating the inverse of the Hessian matrix, we store m_c correction pairs $(\mathbf{s}_k, \mathbf{u}_k)$. Here, m_c is the number of stored corrections supplied by the user (usually $3 \leq m_c \leq 20$), $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{u}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{x}_k$ is the current iteration point, $\mathbf{x}_{k+1}$ is the next iteration point and $\nabla f(\mathbf{x}_k)$ and $\nabla f(\mathbf{x}_{k+1})$ are the corresponding gradients at those points, respectively. When the available storage space is used up, the oldest pair $(\mathbf{s}_{k-m_c}, \mathbf{u}_{k-m_c})$ is deleted and a new one is inserted. The stored correction pairs are used to define the matrices D_k implicitly through the *inverse BFGS update* formula given in the form (see, e.g., Fletcher (1987))

$$D_{k+1} = V_k^T D_k V_k + \frac{1}{b_k}\mathbf{s}_k\mathbf{s}_k^T, \tag{4.1}$$

where

$$b_k = \mathbf{u}_k^T\mathbf{s}_k \qquad \text{and} \qquad V_k = I - \frac{1}{b_k}\mathbf{u}_k\mathbf{s}_k^T.$$

Suppose now that we have the current iterate $\mathbf{x}_k$ and we have stored at most m_c pairs $(\mathbf{s}_i, \mathbf{u}_i)$, where $i = 1, ..., k-1$, if $k \leq m_c$, and $i = k - m_c, \ldots, k-1$, if $k > m_c$. We first define the basic matrix $D_k^{(0)}$, which is usually a diagonal matrix of the form $D_k^{(0)} = \vartheta_k I$ (see, e.g., Nocedal (1997)), where

$$\vartheta_k = \frac{\mathbf{u}_{k-1}^T\mathbf{s}_{k-1}}{\mathbf{u}_{k-1}^T\mathbf{u}_{k-1}}.$$

Then the basic matrix is updated (at most) m_c times by using the BFGS formula (4.1) with the vectors $\mathbf{s}_i$ and $\mathbf{u}_i$. Thus, for $k \leq m_c$, the approximation D_k of the inverse of the Hessian matrix can be written as

$$D_k = \left(\prod_{i=1}^{k-1} V_i\right)^T D_k^{(0)} \left(\prod_{i=1}^{k-1} V_i\right) + \sum_{l=1}^{k-1} \frac{1}{b_l} \left(\prod_{i=l+1}^{k-1} V_i\right)^T \mathbf{s}_l \mathbf{s}_l^T \left(\prod_{i=l+1}^{k-1} V_i\right) \qquad (4.2)$$

and for $k > m_c$, D_k can be written as

$$\begin{aligned} D_k = &\left(\prod_{i=k-m_c}^{k-1} V_i\right)^T D_k^{(0)} \left(\prod_{i=k-m_c}^{k-1} V_i\right) \\ &+ \sum_{l=k-m_c}^{k-1} \frac{1}{b_l} \left(\prod_{i=l+1}^{k-1} V_i\right)^T \mathbf{s}_l \mathbf{s}_l^T \left(\prod_{i=l+1}^{k-1} V_i\right). \qquad (4.3) \end{aligned}$$

If $D_k^{(0)}$ is positive definite, then all the matrices defined by (4.2) and (4.3) are positive definite (provided $\mathbf{u}_i^T \mathbf{s}_i > 0$ for all i) (see, e.g., Nocedal (1980)).

We shall now present an algorithm for the limited memory BFGS method.

Algorithm 4.1. (L-BFGS Method).

Data: Choose a final accuracy tolerance $\varepsilon > 0$, positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$ and the maximum number of stored correction pairs $m_c > 1$.

Step 0: (*Initialization.*) Choose the starting point $\mathbf{x}_1 \in \mathbb{R}^n$ and the symmetric, positive definite matrix $D_1^{(0)}$, e.g. $D_1^{(0)} = I$. Compute $f_1 = f(\mathbf{x}_1)$ and $\nabla f_1 = \nabla f(\mathbf{x}_1)$. Set the iteration counter $k = 1$.

Step 1: (*Direction finding.*) If $\|\nabla f_k\| \leq \varepsilon$, then stop. Otherwise, compute $\mathbf{d}_k = -D_k \nabla f_k$ by Algorithm 4.2.

Step 2: (*Line search.*) Determine step size $t_k > 0$ satisfying the Wolfe conditions:

$$f(\mathbf{x}_k + t_k \mathbf{d}_k) - f_k \leq \varepsilon_L t_k \mathbf{d}_k^T \nabla f_k$$

and

$$\mathbf{d}_k^T \nabla f(\mathbf{x}_k + t_k \mathbf{d}_k) \geq \varepsilon_R \mathbf{d}_k^T \nabla f_k$$

(try the step size $t_k = 1$ first). Set $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k$. Compute $f_{k+1} = f(\mathbf{x}_{k+1})$ and $\nabla f_{k+1} = \nabla f(\mathbf{x}_{k+1})$.

Step 3: (*Update.*) If $k > m_c$, delete the oldest correction pair $(\mathbf{s}_{k-m_c}, \mathbf{u}_{k-m_c})$. Compute and store $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{u}_k = \nabla f_{k+1} - \nabla f_k$. Calculate the basic matrix $D_{k+1}^{(0)} = \vartheta_{k+1} I$, where

$$\vartheta_{k+1} = \frac{\mathbf{u}_k^T \mathbf{s}_k}{\mathbf{u}_k^T \mathbf{u}_k}.$$

Increase k by one and go to Step 1.

Note that in Step 1 the matrices D_k are not formed explicitly but the search direction $\mathbf{d}_k = -D_k \nabla f_k$ is calculated recursively. Suppose now that the current iteration is k and that we have the number of stored corrections $m_k = \min\{\, k-1, m_c \,\}$ and the m_k pairs of difference vectors $(\mathbf{s}_i, \mathbf{u}_i)$, which we label for simplicity $(\mathbf{s}_1, \mathbf{u}_1), \ldots, (\mathbf{s}_{m_k}, \mathbf{u}_{m_k})$. In addition, we have an initial positive definite diagonal matrix $D_k^{(0)}$ and the current gradient ∇f_k. We now present an efficient algorithm for direction finding by Nocedal (1980).

Algorithm 4.2. (Direction Finding I).

Step 0: Set $\mathbf{y} = \nabla f_k$.

Step 1: For $i = m_k$ to 1 set (*backward recurrence*)

$$\sigma_i = \frac{\mathbf{s}_i^T \mathbf{y}}{\mathbf{u}_i^T \mathbf{s}_i} \qquad \text{(store } \sigma_i\text{) and set}$$
$$\mathbf{y} = \mathbf{y} - \sigma_i \mathbf{u}_i.$$

Step 2: Set $\mathbf{r} = D_k^{(0)} \mathbf{y}$.

Step 3: For $i = 1$ to m_k set (*forward recurrence*)

$$\nu = \frac{\mathbf{u}_i^T \mathbf{r}}{\mathbf{u}_i^T \mathbf{s}_i} \qquad \text{and}$$
$$\mathbf{r} = \mathbf{r} + (\sigma_i - \nu)\mathbf{s}_i.$$

Step 4: Set $\mathbf{d}_k = -\mathbf{r}$.

Note that, if $m_k = 0$, Steps 1 and 3 are not executed and, thus, the search direction at the first iteration is $\mathbf{d}_k = -D_1^{(0)} \nabla f_1$.

This representation of the BFGS update requires only $O(nm_c)$ storage space. Assuming $m_c \ll n$ this is much less than the $O(n^2)$ storage space required for the standard BFGS implementation. The search direction can be computed implicitly using at most $O(nm_c)$ operations. Also this is much less than the $O(n^2)$ operations normally needed to compute $-D_k\nabla f(\mathbf{x}_k)$ when the whole matrix D_k is stored. This makes the limited memory BFGS method suitable for large-scale problems, since it has been observed in practice that small values of m_c ($m_c \in [3,7]$) give satisfactory results (see, e.g., Liu and Nocedal (1989) and Gilbert and Lemaréchal (1989)). Furthermore, the limited memory BFGS method given above has been proved to be globally and linearly convergent on convex problems for any starting point (see, Liu and Nocedal (1989)).

4.2 Compact Representation of Matrices

The recursive formula given in the previous section is very efficient for unconstrained optimization. However, if we wish to use updates other than BFGS or if we need to solve problems with constraints, there are many advantages of the compact representation of limited memory matrices (see, e.g., Byrd et al. (1994)). When constraints are present, the recursive formula is much less economical for some of the required calculations. For example, the recursive formula does not take good advantage of the sparsity of the constraints. Furthermore, if we want to use the direct approximation of the Hessian, B_k, instead of the inverse D_k, we have to use the compact representation of the matrices, since there exist no analogous recursion for the direct approximation of the Hessian B_k.

In this section, we first consider the updating process in a general setting. We show that both the inverse BFGS and the inverse SR1 updates can be presented in a compact matrix form. At the end of this section we apply these results to limited memory methods for smooth large-scale optimization and in Chapter 5 the same ideas are used to construct a new method for nonsmooth large-scale optimization.

4.2.1 Compact Representation of BFGS and SR1 Updates

In this subsection, we describe the representations of inverse BFGS and inverse SR1 updates. We show that both of these updates can be represent in a compact matrix form.

So far, we have dealt with difference vectors $\mathbf{s}_i$ and $\mathbf{u}_i$ and avoided storing any matrices. Now we define the $n \times (k-1)$ matrices S_k and U_k by

$$\begin{aligned} S_k &= \begin{bmatrix} \mathbf{s}_1 & \dots & \mathbf{s}_{k-1} \end{bmatrix} \qquad \text{and} \\ U_k &= \begin{bmatrix} \mathbf{u}_1 & \dots & \mathbf{u}_{k-1} \end{bmatrix}, \end{aligned} \tag{4.4}$$

where, as before, $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{u}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.

Let R_k be an upper triangular matrix of order $k-1$ given in the form

$$(R_k)_{ij} = \begin{cases} \mathbf{s}_i^T \mathbf{u}_j & \text{if } i \le j \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

and let C_k be a diagonal matrix of order $k-1$ such that

$$C_k = \operatorname{diag}[\mathbf{s}_1^T \mathbf{u}_1, \dots, \mathbf{s}_{k-1}^T \mathbf{u}_{k-1}]. \tag{4.6}$$

The following theorem gives a compact representation of the matrix D_k obtained after $k-1$ inverse BFGS updates.

Theorem 4.2.1. *Let the matrix D_1 be symmetric and positive definite. Assume that the $k-1$ pairs $(\mathbf{s}_i, \mathbf{u}_i)_{i=1}^{k-1}$ satisfy $\mathbf{s}_i^T \mathbf{u}_i > 0$. Let the matrix D_1 be updated $k-1$ times by using the inverse BFGS update formula (4.1) with the pairs $(\mathbf{s}_i, \mathbf{u}_i)_{i=1}^{k-1}$. Then the resulting matrix D_k is given by*

$$D_k = D_1 + \begin{bmatrix} S_k & D_1 U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T (C_k + U_k^T D_1 U_k) R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ U_k^T D_1 \end{bmatrix}, \tag{4.7}$$

where S_k, U_k, R_k and C_k are defined as in (4.4), (4.5) and (4.6).

Proof. See Byrd et al. (1994).

Note that the conditions $\mathbf{s}_i^T \mathbf{u}_i > 0$, $i = 1, \dots, k-1$, in Theorem 4.2.1 ensure that R_k is nonsingular and, thus, (4.7) is well defined. This is consistent with the well-known result that the BFGS update formula preserves positive definiteness if $\mathbf{s}_i^T \mathbf{u}_i > 0$ for all i (see, e.g., Fletcher (1987)).

Theorem 4.2.1 gives us a matrix representation of the approximation D_k of the inverse of the Hessian matrix. The direct approximation B_k can also be easily obtained. For details, see Byrd et al. (1994).

Next we prove that also the inverse symmetric rank-one (SR1) update can be presented in a matrix form. The *inverse SR1 update* formula is given by

$$D_{k+1} = D_k - \frac{(D_k \mathbf{u}_k - \mathbf{s}_k)(D_k \mathbf{u}_k - \mathbf{s}_k)^T}{(D_k \mathbf{u}_k - \mathbf{s}_k)^T \mathbf{u}_k}. \tag{4.8}$$

Note that this update is well defined only if the denominator $(D_k\mathbf{u}_k - \mathbf{s}_k)^T\mathbf{u}_k \neq 0$. In recent implementations of the SR1 method, the update is simply skipped if the denominator is very small (see, e.g., Fayez Khalfan et al. (1993)). Since the SR1 update does not in general preserve the positive definiteness of the generated matrices, there is no reason to enforce the curvature condition $\mathbf{s}_k^T\mathbf{u}_k > 0$ as with BFGS update. Thus, we consider the sequence of updates to an arbitrary symmetric matrix D_1 subject only to the assumption that the update is well defined.

The following theorem is modified from the result obtained for the direct SR1 approximation of the Hessian in Byrd et al. (1994).

Theorem 4.2.2. *Let the symmetric matrix D_1 be updated $k-1$ times by means of the inverse SR1 update formula (4.8) using the vectors $(\mathbf{s}_i, \mathbf{u}_i)_{i=1}^{k-1}$, and assume that each update is well defined, that is $(D_j\mathbf{u}_j - \mathbf{s}_j)^T\mathbf{u}_j \neq 0$ for $j = 1, \ldots, k$. Then the resulting matrix D_k is given by*

$$D_k = D_1 - (D_1U_k - S_k)(U_k^TD_1U_k - R_k - R_k^T + C_k)^{-1}(D_1U_k - S_k)^T, \quad (4.9)$$

where S_k, U_k, R_k and C_k are defined as in (4.4), (4.5) and (4.6), and the matrix $M_k = (U_k^TD_1U_k - R_k - R_k^T + C_k)$ is nonsingular.

Proof. We prove this via induction. Suppose $k = 2$. Then the right hand side of (4.9) is

$$D_1 - \frac{(D_1\mathbf{u}_1 - \mathbf{s}_1)(D_1\mathbf{u}_1 - \mathbf{s}_1)^T}{(D_1\mathbf{u}_1 - \mathbf{s}_1)^T\mathbf{u}_1} = D_2,$$

and so the base case is valid.

Now, assume that (4.9) is valid for some k. We define

$$Q_k = \begin{bmatrix}\mathbf{q}_1 & \ldots & \mathbf{q}_{k-1}\end{bmatrix} = D_1U_k - S_k \quad (4.10)$$

and

$$M_k = U_k^TD_1U_k - R_k - R_k^T + C_k. \quad (4.11)$$

Note that the matrix M_k is symmetric. We can now write the SR1 formula (4.9) as

$$D_k = D_1 - Q_kM_k^{-1}Q_k^T.$$

Next we show that (4.9) is valid for $k+1$.

The next approximation of the Hessian, D_{k+1}, can be calculated by applying the SR1 update (4.8) to D_k. So, we have

$$\begin{aligned}
D_{k+1} &= D_1 - Q_k M_k^{-1} Q_k^T \\
&\quad - \frac{\left(D_1\mathbf{u}_k - \mathbf{s}_k - Q_k M_k^{-1} Q_k^T \mathbf{u}_k\right)\left(D_1\mathbf{u}_k - \mathbf{s}_k - Q_k M_k^{-1} Q_k^T \mathbf{u}_k\right)^T}{\left(D_1\mathbf{u}_k - \mathbf{s}_k\right)^T \mathbf{u}_k - \mathbf{u}_k^T Q_k M_k^{-1} Q_k^T \mathbf{u}_k} \\
&= D_1 - Q_k M_k^{-1} Q_k^T - \frac{\left(\mathbf{q}_k - Q_k M_k^{-1}\mathbf{w}_k\right)\left(\mathbf{q}_k - Q_k M_k^{-1}\mathbf{w}_k\right)^T}{\mathbf{q}_k^T\mathbf{u}_k - \mathbf{w}_k^T M_k^{-1}\mathbf{w}_k} \\
&= D_1 - Q_k M_k^{-1} Q_k^T - \frac{1}{\delta_k}\left(\mathbf{q}_k\mathbf{q}_k^T - \mathbf{q}_k\mathbf{w}_k^T M_k^{-1} Q_k^T - Q_k M_k^{-1}\mathbf{w}_k\mathbf{q}_k^T\right. \\
&\quad \left. + Q_k M_k^{-1}\mathbf{w}_k\mathbf{w}_k^T M_k^{-1} Q_k^T\right) \\
&= D_1 - \frac{1}{\delta_k}\left(\mathbf{q}_k\mathbf{q}_k^T - \mathbf{q}_k\mathbf{w}_k^T M_k^{-1} Q_k^T - Q_k M_k^{-1}\mathbf{w}_k\mathbf{q}_k^T\right. \\
&\quad \left. + Q_k\left(\delta_k M_k^{-1} + M_k^{-1}\mathbf{w}_k\mathbf{w}_k^T M_k^{-1}\right) Q_k^T\right),
\end{aligned}$$

where we have defined

$$\mathbf{w}_k = Q_k^T\mathbf{u}_k, \tag{4.12}$$

and where the denominator

$$\begin{aligned}
\delta_k &= \mathbf{q}_k^T\mathbf{u}_k - \mathbf{w}_k^T M_k^{-1}\mathbf{w}_k \\
&= (D_k\mathbf{u}_k - \mathbf{s}_k)^T\mathbf{u}_k
\end{aligned} \tag{4.13}$$

is non-zero by assumption. We may express this as

$$D_{k+1} = D_1 - \frac{1}{\delta_k}\begin{bmatrix} Q_k & q_k \end{bmatrix}\begin{bmatrix} M_k^{-1}\left(\delta_k I + \mathbf{w}_k\mathbf{w}_k^T M_k^{-1}\right) & -M_k^{-1}\mathbf{w}_k \\ -\mathbf{w}_k^T M_k^{-1} & 1 \end{bmatrix}\begin{bmatrix} Q_k^T \\ q_k^T \end{bmatrix}. \tag{4.14}$$

Note that the matrix $\begin{bmatrix} Q_k & q_k \end{bmatrix} = Q_{k+1}$.

From definitions (4.10), (4.11) and (4.12) we see that the new matrix M_{k+1} is given by

$$\begin{aligned}
M_{k+1} &= U_{k+1}^T D_1 U_{k+1} - R_{k+1} - R_{k+1}^T + C_{k+1} \\
&= \begin{bmatrix} M_k & U_k^T D_1\mathbf{u}_k - S_k^T\mathbf{u}_k \\ (U_k^T D_1\mathbf{u}_k - S_k^T\mathbf{u}_k)^T & \mathbf{u}_k^T D_1\mathbf{u}_k - \mathbf{s}_k^T\mathbf{u}_k \end{bmatrix} \\
&= \begin{bmatrix} M_k & \mathbf{w}_k \\ \mathbf{w}_k^T & \mathbf{q}_k^T\mathbf{u}_k \end{bmatrix}.
\end{aligned}$$

By direct multiplication, using (4.10), (4.12) and (4.13) we see that

$$\begin{bmatrix} M_k & \mathbf{w}_k \\ \mathbf{w}_k^T & \mathbf{q}_k^T\mathbf{u}_k \end{bmatrix} \begin{bmatrix} M_k^{-1}\left(\delta_k I + \mathbf{w}_k\mathbf{w}_k^T M_k^{-1}\right) & -M_k^{-1}\mathbf{w}_k \\ -\mathbf{w}_k^T M_k^{-1} & 1 \end{bmatrix} \frac{1}{\delta_k} = I. \tag{4.15}$$

Therefore, M_{k+1} is invertible with M_{k+1}^{-1} given in the second matrix in (4.15), but this is the same matrix that appears in (4.14). Thus, we see that (4.14) is equivalent to (4.9) with k replaced by $k+1$. Hence, (4.9) is valid also for $k+1$. □

4.2.2 Compact Representation of Limited Memory Matrices

In the previous subsection we showed that both the inverse BFGS and the inverse SR1 updates can be presented in the compact matrix forms (4.7) and (4.9), respectively. Now, it is straight forward to describe a limited memory implementation for these updates. We keep the m_c most recent correction pairs $(\mathbf{s}_i, \mathbf{u}_i)$ in order to implicitly define the approximation of the inverse of the Hessian matrix at each iteration. At every iteration this set of pairs is updated by deleting the oldest pair $(\mathbf{s}_{k-m_c}, \mathbf{u}_{k-m_c})$ and adding a new one $(\mathbf{s}_k, \mathbf{u}_k)$. We assume that the maximum number of stored corrections m_c is constant, even if it is possible to adapt all the formulae of this section to the case where m_c is varying at every iteration (see, e.g., Kolda et al. (1998)).

We define $n \times m_k$ correction matrices S_k and U_k by

$$\begin{aligned} S_k &= \begin{bmatrix} \mathbf{s}_{k-m_k} & \dots & \mathbf{s}_{k-1} \end{bmatrix} \qquad \text{and} \\ U_k &= \begin{bmatrix} \mathbf{u}_{k-m_k} & \dots & \mathbf{u}_{k-1} \end{bmatrix}, \end{aligned} \tag{4.16}$$

where, as before, $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{u}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $m_k = \min\{\, k-1, m_c \,\}$ is the current number of stored corrections.

We assume that the basic matrix $D_k^{(0)}$ is given in the form $D_k^{(0)} = \vartheta_k I$, for some $\vartheta_k > 0$, as is commonly done in practice (see, e.g., Gilbert and Lemaréchal (1989) and Liu and Nocedal (1989)). From (4.7) we see that the inverse limited memory BFGS update can be expressed as

$$D_k = \vartheta_k I + \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T(C_k + \vartheta_k U_k^T U_k)R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \vartheta_k U_k^T \end{bmatrix}, \tag{4.17}$$

where R_k is the upper triangular matrix of order m_k given in the form

$$(R_k)_{ij} = \begin{cases} (\mathbf{s}_{k-m_k-1+i})^T(\mathbf{u}_{k-m_k-1+j}) & \text{if } i \le j \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

and where C_k is the diagonal matrix of order m_k such that

$$C_k = \operatorname{diag}[\mathbf{s}_{k-m_k}^T \mathbf{u}_{k-m_k}, \ldots, \mathbf{s}_{k-1}^T \mathbf{u}_{k-1}]. \tag{4.19}$$

When the new iteration point $\mathbf{x}_{k+1}$ is generated, the new correction matrices S_{k+1} and U_{k+1} are obtained by deleting the oldest corrections $\mathbf{s}_{k-m_k}$ and $\mathbf{u}_{k-m_k}$ from S_k and U_k if $m_{k+1} = m_k$ (that is, $k > m_c$) and by adding the most recent corrections $\mathbf{s}_k$ and $\mathbf{u}_k$ to the matrices.

Next we describe some procedures for the calculation of the search direction $\mathbf{d}_k = -D_k \nabla f(\mathbf{x}_k)$ when the compact matrix representation of limited memory BFGS updates is used. In addition to the two $n \times m_c$ matrices S_k and U_k, the $m_c \times m_c$ matrices $U_k^T U_k$, R_k and C_k are stored. Since in practice m_c is clearly smaller than n, the storage space required by these three auxiliary matrices is insignificant but the savings in computation are considerable when comparing with the standard BFGS implementation.

At the k-th iteration we have to update the limited memory representation of D_{k-1} to get D_k and calculate the search direction $\mathbf{d}_k = -D_k \nabla f(\mathbf{x}_k)$. To update D_{k-1} we delete a column from S_{k-1} and U_{k-1} and add a new column to each of these matrices. Then we make the corresponding updates to R_{k-1}, $U_{k-1}^T U_{k-1}$ and C_{k-1}. These updates can be done in $O(m_c^2)$ operations by saving a small amount of additional information, namely the m_c-vectors $S_{k-1}^T \nabla f(\mathbf{x}_{k-1})$ and $U_{k-1}^T \nabla f(\mathbf{x}_{k-1})$ from the previous iteration. For example, the new triangular matrix R_k is formed from R_{k-1} (see (4.18)) by deleting the first row and column if $m_k = m_{k-1}$ and by adding a new column as the right column and a new row as the last row. The new column is given by

$$S_k^T \mathbf{u}_{k-1} = S_k^T (\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k-1}))$$

and the new row has zero in its first $m_k - 1$ components. The product $S_k^T \mathbf{u}_{k-1}$ can be computed efficiently since we already know $m_k - 1$ components of $S_k^T \nabla f(\mathbf{x}_{k-1})$ from $S_{k-1}^T \nabla f(\mathbf{x}_{k-1})$. We only need to calculate $\mathbf{s}_{k-1}^T \nabla f(\mathbf{x}_{k-1})$ and do the subtractions. The product $\mathbf{s}_{k-1}^T \nabla f(\mathbf{x}_{k-1})$ can be calculated in $O(m_c^2)$ by using the formula given in Byrd et al. (1994). The matrix $U_k^T U_k$ can be updated in a similar way. In this case, both the new column and the new row are given by $U_k^T \mathbf{u}_{k-1}$.

Now we give an efficient algorithm by Byrd et al. (1994) for updating the matrix D_k by limited memory BFGS formula and for computing the search direction $\mathbf{d}_k = -D_k \nabla f(\mathbf{x}_k)$.

Let the current iteration point be $\mathbf{x}_k$ and let the current number of stored corrections be $m_k = \min\{\, k-1, m_c \,\}$. Suppose that we have the previous corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$, the current gradient $\nabla f_k = \nabla f(\mathbf{x}_k)$, the matrices S_{k-1}, U_{k-1}, R_{k-1}, $U_{k-1}^T U_{k-1}$ and C_{k-1} and the vectors $S_{k-1}^T \nabla f_{k-1}$ and $U_{k-1}^T \nabla f_{k-1}$ from the previous iteration. In addition, suppose that the initial matrix $D_k^{(0)} = \vartheta_k I$.

Algorithm 4.3. (Direction Finding II).

Step 1: Obtain S_k and U_k by updating S_{k-1} and U_{k-1}.

Step 2: Compute m_k-vectors $S_k^T \nabla f_k$ and $U_k^T \nabla f_k$.

Step 3: Compute m_k-vectors $S_k^T \mathbf{u}_{k-1}$ and $U_k^T \mathbf{u}_{k-1}$ by using the fact

$$\mathbf{u}_{k-1} = \nabla f_k - \nabla f_{k-1}.$$

Step 4: Update $m_k \times m_k$ matrices R_k and $U_k^T U_k$.

Step 5: Update C_k by deleting the first element of C_{k-1} if $m_k = m_{k-1}$ and adding $\mathbf{s}_{k-1}^T \mathbf{u}_{k-1} = (S_k^T \mathbf{u}_{k-1})_{m_k}$ as the last element (note that C_k is a diagonal matrix and, thus, stored as a vector).

Step 6: Compute ϑ_k:

$$\vartheta_k = \frac{\mathbf{u}_{k-1}^T \mathbf{s}_{k-1}}{\mathbf{u}_{k-1}^T \mathbf{u}_{k-1}}.$$

Note that both $\mathbf{u}_{k-1}^T \mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}^T \mathbf{u}_{k-1}$ have already been calculated.

Step 7: Compute

$$\mathbf{p} = \begin{bmatrix} (R_k^{-1})^T (C_k + \vartheta_k U_k^T U_k) R_k^{-1} (S_k^T \nabla f_k) - \vartheta_k (R_k^{-1})^T (U_k^T \nabla f_k) \\ -R_k^{-1} (S_k^T \nabla f_k) \end{bmatrix}.$$

Step 8: Compute

$$\mathbf{d}_k = -D_k \nabla f_k = -\vartheta_k \nabla f_k - \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \mathbf{p}.$$

Note that in the first iteration ($k = 1$) the search direction is not calculated by the Algorithm 4.3 but it is directly defined as $\mathbf{d}_1 = -\nabla f(\mathbf{x}_1)$.

The algorithm given above requires the same amount of work per iteration as the two loop recursion in Algorithm 4.2. Thus, the two algorithms are equally efficient for unconstrained problems. In constrained optimization it is very common to have problems where the gradients of the constraints are sparse. In these cases, the compact matrix representations are more useful than

Algorithm 4.2, since Algorithm 4.2 does not take advantage of the sparsity of the vectors involved. For further study of constrained optimization with limited memory variable metric methods we refer to Byrd et al. (1994), Byrd et al. (1995) and Lalee et al. (1998).

So far, we have only given a representation to a limited memory BFGS update. However, limited memory DFP and SR1 updates can be expressed in the compact matrix form as well (see, e.g., Byrd et al. (1994), Kolda et al. (1998) and Lukšan and Spedicato (2000)). For example, from 4.9 we see that the inverse limited memory SR1 update can be written as

$$D_k = D_k^{(0)} - (D_k^{(0)}U_k - S_k)(U_k^T D_k^{(0)} U_k - R_k - R_k^T + C_k)^{-1}(D_k^{(0)}U_k - S_k)^T, \tag{4.20}$$

where S_k, U_k, R_k and C_k are defined as in (4.16), (4.18) and (4.19). We do not give any algorithms for computing products involving limited memory SR1 matrices, because the ideas are very similar to those described above with the limited memory BFGS method.

The compact representations of limited memory matrices based on the direct approximations of the Hessian matrix instead of inverses can also be easily obtained. For details see, for example, Byrd et al. (1994).

5 Nonsmooth Large-Scale Optimization

None of the general nonsmooth methods presented in Chapter 3 is very efficient for large-scale optimization (see, e.g., Mäkelä et al. (1999), Kärkkäinen et al. (2001)). This is supported by numerical tests concerning bundle methods, variable metric bundle methods and the bundle-Newton method to be presented in Chapter 6. Standard variable metric methods and variable metric bundle methods cannot be used since they utilize dense approximations of the Hessian matrices and bundle methods are not applicable because they need to solve a rather expensive quadratic direction finding (3.15) at every iteration, which is a time-consuming procedure. Furthermore, we were not able to find any general solver for large-scale nonsmooth optimization problems from literature.

In this chapter, we present a new method for large-scale nonsmooth unconstrained optimization. This method is a hybrid of the variable metric bundle method (Section 3.3) and the limited memory variable metric method (Subsection 4.2.2). We go through the algorithm step by step and describe both its theoretical properties and some details of the implementation. Note that we recall many formulae and procedures given before to make this chapter more self-contained.

The new method will be called the limited memory variable metric bundle method and its basic idea is very simple. We use all the ideas of variable metric bundle methods but the matrix updating is done by using the limited memory approach. Thus, we have a method that does not have to solve the time-consuming quadratic subproblem (3.15) appearing in standard bundle methods and that uses only few vectors to represent the variable metric approximation of the Hessian matrix. In this way, we avoid storing and manipulating large $n \times n$ matrices as in variable metric bundle methods.

As mentioned in the beginning of the Chapter 4, there exist three basic approach to deal with the variable metric approximation of the Hessian matrix in large-scale settings. We chose this limited memory approach since it does not need any information of the structure of problem or its Hessian and, thus, the only assumptions required are that the objective function f is locally Lipschitz continuous and that we can evaluate the value of the objective function $f(\mathbf{x})$ and its arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$ at each point $\mathbf{x} \in \mathbb{R}^n$.

5.1 Direction Finding and Matrix Updating

In this section we describe how to update the approximation D_k of the inverse of the Hessian matrix and find the search direction $\mathbf{d}_k$ when the limited memory variable metric bundle method is used. The basic idea of direction finding is the same as with the original variable metric bundle methods (see Subsection 3.3.1), that is, $\mathbf{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k$ but the approximations D_k of the inverse of the Hessian matrix are formed by using the compact matrix representation of limited memory updates (see Subsection 4.2.2).

We use the compact representation of limited memory matrices, since in addition to the BFGS updating formula we need also the SR1 updating formula and for SR1 updates there exist no recursive updating formula analogous to that given in Algorithm 4.2. Moreover, this approach makes the further development, where the new method is generalized for constrained problems, much easier.

First we describe how to update limited memory matrices. We recall how to present the inverse limited memory BFGS and the inverse limited memory SR1 updates in a compact matrix form. After that we give some ideas of how the search direction $\mathbf{d}_k$ can be calculated by using these different updates. After discussing these calculations separately, we then link them together.

As before, we denote by m_c the maximum number of stored corrections supplied by the user ($3 \le m_c \le 20$) and by $m_k = \min\{\, k-1, m_c \,\}$ the current number of stored corrections. We keep the m_c most recent correction pairs $(\mathbf{s}_i, \mathbf{u}_i)$ to implicitly define the approximation of the inverse of the Hessian matrix at each iteration. At every iteration this set of pairs is updated by deleting the oldest pair $(\mathbf{s}_{k-m_c}, \mathbf{u}_{k-m_c})$, if $m_k = m_c$, and adding a new one $(\mathbf{s}_k, \mathbf{u}_k)$.

The $n \times m_k$ correction matrices S_k and U_k are defined as in Subsection 4.2.2, that is,

$$\begin{aligned} S_k &= \begin{bmatrix} \mathbf{s}_{k-m_k} & \dots & \mathbf{s}_{k-1} \end{bmatrix} \qquad \text{and} \\ U_k &= \begin{bmatrix} \mathbf{u}_{k-m_k} & \dots & \mathbf{u}_{k-1} \end{bmatrix} \end{aligned}$$

but now the difference vectors $\mathbf{s}_k$ and $\mathbf{u}_k$ are given by

$$\begin{aligned} \mathbf{s}_k &= \mathbf{y}_{k+1} - \mathbf{x}_k \qquad \text{and} \\ \mathbf{u}_k &= \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m, \end{aligned} \tag{5.1}$$

where $\mathbf{y}_{k+1}$ is the current auxiliary iteration point, $\mathbf{x}_k$ is the current iteration point and $\boldsymbol{\xi}_{k+1}$ and $\boldsymbol{\xi}_m$ are the corresponding subgradients of these points (m is the index of the iteration after the latest serious step).

As in (4.17), we define the inverse limited memory BFGS update by the formula

$$D_k = \vartheta_k I + \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T (C_k + \vartheta_k U_k^T U_k) R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \vartheta_k U_k^T \end{bmatrix}, \tag{5.2}$$

where R_k is an upper triangular matrix of order m_k given in the form

$$(R_k)_{ij} = \begin{cases} (\mathbf{s}_{k-m_k-1+i})^T (\mathbf{u}_{k-m_k-1+j}) & \text{if } i \le j \\ 0 & \text{otherwise,} \end{cases}$$

C_k is a diagonal matrix of order m_k such that

$$C_k = \operatorname{diag}\,[\mathbf{s}_{k-m_k}^T \mathbf{u}_{k-m_k}, \dots, \mathbf{s}_{k-1}^T \mathbf{u}_{k-1}]$$

and the multiplier $\vartheta_k > 0$ is given by

$$\vartheta_k = \frac{\mathbf{u}_{k-1}^T \mathbf{s}_{k-1}}{\mathbf{u}_{k-1}^T \mathbf{u}_{k-1}}.$$

In addition, the inverse limited memory SR1 update is defined as in (4.20). That is,

$$D_k = \vartheta_k I - (\vartheta_k U_k - S_k)(\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k - S_k)^T, \quad (5.3)$$

where the matrices S_k, U_k, R_k and C_k are defined as before. In our implementation, we use the value $\vartheta_k = 1$ for every k with SR1 update.

When the new auxiliary iteration point $\mathbf{y}_{k+1}$ is generated, the new correction matrices S_{k+1} and U_{k+1} are obtained by deleting the oldest corrections $\mathbf{s}_{k-m_k}$ and $\mathbf{u}_{k-m_k}$ from S_k and U_k if $m_{k+1} = m_k$ (that is, $k > m_c$) and by adding the most recent corrections $\mathbf{s}_k$ and $\mathbf{u}_k$ to the matrices.

Next, we describe some procedures for updating the compact representation of limited memory matrices. In addition to the two $n \times m_c$ matrices S_k and U_k, the $m_c \times m_c$ matrices R_k, $U_k^T U_k$ and C_k are stored. Since in practice m_c is clearly smaller than n, the storage space required by these three auxiliary matrices is insignificant but the savings in computation effort are considerable. The computations needed to update the matrices R_k, U_k and C_k are quite similar to those given in Subsection 4.2.2.

At the k-th iteration we have to update the limited memory representation of D_{k-1} to get D_k. Thus, we delete a column from S_{k-1} and U_{k-1} and add a new column to each of these matrices. Then we make the corresponding updates to R_{k-1}, $U_{k-1}^T U_{k-1}$ and C_{k-1}. These updates can be done in $O(m_c^2)$ operations by saving the m_c-vectors $S_{k-1}^T \boldsymbol{\xi}_m$ and $U_{k-1}^T \boldsymbol{\xi}_m$ from the previous iteration. The new triangular matrix R_k is formed from R_{k-1} by deleting the first row and the first column if $m_k = m_{k-1}$ and by adding a new column to the right and a new row to the bottom. The new column is given by

$$S_k^T \mathbf{u}_{k-1} = S_k^T (\boldsymbol{\xi}_k - \boldsymbol{\xi}_m)$$

and the new row has zero in its first $m_k - 1$ components. The matrix $U_k^T U_k$ can be updated in a similar way. In this case, both the new column and the new row are given by $U_k^T \mathbf{u}_{k-1}$. The products $S_k^T \mathbf{u}_{k-1}$ and $U_k^T \mathbf{u}_{k-1}$ can be computed efficiently since we already know $m_k - 1$ components of $S_k^T \boldsymbol{\xi}_m$ and $U_k^T \boldsymbol{\xi}_m$ from $S_{k-1}^T \boldsymbol{\xi}_m$ and $U_{k-1}^T \boldsymbol{\xi}_m$, respectively. We only need to calculate $\mathbf{s}_{k-1}^T \boldsymbol{\xi}_m$ and $\mathbf{u}_{k-1}^T \boldsymbol{\xi}_m$ and do the subtractions. The diagonal matrix C_k is updated by deleting the first element of C_{k-1} and adding $\mathbf{s}_{k-1}^T \mathbf{u}_{k-1}$ as the last element (note that C_k is stored as a vector).

After we have taken a null step, the approximation D_k of the inverse of the Hessian matrix is formed by using the compact representation of SR1 update (5.3), since this update formula preserves the boundedness of the generated matrices. In addition, we use an aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ to calculate the search direction

$$\mathbf{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k.$$

Now we give an efficient algorithm for updating the limited memory SR1 matrix D_k and for computing the search direction $\mathbf{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k$. This algorithm is used whenever the previous step was a null step.

Suppose that the number of current corrections is m_k and that we have the current iteration point $\mathbf{x}_k$, the previous corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$, the current (auxiliary) subgradient $\boldsymbol{\xi}_k \in \partial f(\mathbf{y}_k)$, the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ ($\tilde{\boldsymbol{\xi}}_k \neq \boldsymbol{\xi}_k$ in the case of a null step), the basic subgradient $\boldsymbol{\xi}_m \in \partial f(\mathbf{x}_k)$, the $n \times m_k$ matrices S_{k-1} and U_{k-1}, the $m_k \times m_k$ matrices R_{k-1}, $U_{k-1}^T U_{k-1}$ and C_{k-1} and the vectors $S_{k-1}^T\boldsymbol{\xi}_m$ and $U_{k-1}^T\boldsymbol{\xi}_m$ available.

Algorithm 5.1. (SR1 Updating and Direction Finding).

Step 1: If $-\mathbf{d}_{k-1}^T\mathbf{u}_{k-1} - \tilde{\boldsymbol{\xi}}_{k-1}^T\mathbf{s}_{k-1} \geq 0$, then set $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$ $C_k = C_{k-1}$, $S_k^T\boldsymbol{\xi}_m = S_{k-1}^T\boldsymbol{\xi}_m$ and $U_k^T\boldsymbol{\xi}_m = U_{k-1}^T\boldsymbol{\xi}_m$ and go to Step 6.

Step 2: Obtain S_k and U_k by updating S_{k-1} and U_{k-1}.

Step 3: Compute m_k-vectors $S_k^T\boldsymbol{\xi}_k$ and $U_k^T\boldsymbol{\xi}_k$.

Step 4: Compute m_k-vectors $S_k^T\mathbf{u}_{k-1}$ and $U_k^T\mathbf{u}_{k-1}$ by using the fact

$$\mathbf{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_m.$$

Store m_k-vectors $S_k^T\boldsymbol{\xi}_m$ and $U_k^T\boldsymbol{\xi}_m$.

Step 5: Update $m_k \times m_k$ matrices R_k, $U_k^T U_k$ and C_k.

Step 6: Set $\vartheta_k = 1.0$.

Step 7: Compute m_k-vectors $S_k^T\tilde{\boldsymbol{\xi}}_k$ and $U_k^T\tilde{\boldsymbol{\xi}}_k$.

Step 8: Compute

$$\mathbf{p} = (\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k^T\tilde{\boldsymbol{\xi}}_k - S_k^T\tilde{\boldsymbol{\xi}}_k).$$

Step 9: Compute

$$\mathbf{d}_k = -\vartheta_k\tilde{\boldsymbol{\xi}}_k + (\vartheta_k U_k - S_k)\mathbf{p}.$$

The condition (see Step 1)

$$-\mathbf{d}_i^T\mathbf{u}_i - \tilde{\boldsymbol{\xi}}_i^T\mathbf{s}_i < 0 \qquad \text{for all } i \tag{5.4}$$

assures the positive definiteness of matrices obtained by the limited memory SR1 update.

The assertion above may need some argumentation: Let us denote $\mathbf{v}_i = D_i\mathbf{u}_i - \mathbf{s}_i$. Thus, the condition (5.4) can be equally expressed by

$$\tilde{\boldsymbol{\xi}}_i^T\mathbf{v}_i < 0 \qquad \text{for all } i.$$

Now, the condition (5.4) implies that

$$\begin{aligned}\mathbf{d}_i^T\mathbf{u}_i &= -\tilde{\boldsymbol{\xi}}_i^T\mathbf{v}_i - \tilde{\boldsymbol{\xi}}_i^T\mathbf{s}_i = -\tilde{\boldsymbol{\xi}}_i^T\mathbf{v}_i - t_R^i\tilde{\boldsymbol{\xi}}_i^T\mathbf{d}_i = -\tilde{\boldsymbol{\xi}}_i^T\mathbf{v}_i + t_R^i\tilde{\boldsymbol{\xi}}_i^T D_i\tilde{\boldsymbol{\xi}}_i \\ &> t_R^i\tilde{\boldsymbol{\xi}}_i^T D_i\tilde{\boldsymbol{\xi}}_i = t_R^i\tilde{\boldsymbol{\xi}}_i^T D_iB_iD_i\tilde{\boldsymbol{\xi}}_i = t_R^i\mathbf{d}_i^TB_i\mathbf{d}_i,\end{aligned} \tag{5.5}$$

where $B_i = D_i^{-1}$ and $t_R^i > 0$ is the step size.

For all $i > 1$ the matrix B_{i+1} defined by the direct SR1 formula is given by

$$B_{i+1} = B_i + \frac{(\mathbf{u}_i - t_R^iB_i\mathbf{d}_i)(\mathbf{u}_i - t_R^iB_i\mathbf{d}_i)}{t_R^i\mathbf{d}_i^T(\mathbf{u}_i - t_R^iB_i\mathbf{d}_i)}.$$

The matrix B_{i+1} is positive definite if B_i is positive definite and the denominator is greater than zero, that is,

$$t_R^i\mathbf{d}_i^T\mathbf{u}_i - (t_R^i)^2\mathbf{d}_i^TB_i\mathbf{d}_i > 0.$$

The latter is obviously guaranteed, if the condition (5.5) is valid.

The nonsingular matrix D_{i+1} is positive definite, if its inverse B_{i+1} is positive definite and thus, the condition (5.4) guarantees the positive definiteness of matrices formed by the limited memory SR1 update.

Because the boundedness of generated matrices is not required after a serious step, we use the limited memory BFGS update formula (5.2) to compute the approximation D_k of the inverse of the Hessian matrix and the search direction $\mathbf{d}_k$ is calculated by using the original subgradient $\boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$. Thus, the search direction is defined by

$$\mathbf{d}_k = -D_k\boldsymbol{\xi}_k.$$

Due to fact that after a serious step the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$ and

$$\begin{aligned} \mathbf{s}_{k-1} &= \mathbf{x}_k - \mathbf{x}_{k-1} \qquad \text{and} \\ \mathbf{u}_{k-1} &= \boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1}, \end{aligned}$$

(note that in the case of a serious step this representation of $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$ is not conflicting with (5.1)) the calculations used are very similar to those given in Subsection 4.2.2. In fact, all those calculations can be done by replacing the gradient $\nabla f(\mathbf{x})$ by an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$. However, rather than updating and inverting the upper triangular matrix R_k at every iteration, we update and store the inverse of R_k, that is R_k^{-1}. The new triangular matrix R_k^{-1} is formed from R_{k-1}^{-1} by deleting the first row and the first column if $m_k = m_{k-1}$ and by adding a new column to the right and a new row to the bottom. Let us denote by $(R_{k-1}^{-1})'$ the $(m_k - 1) \times (m_k - 1)$ lower right submatrix of R_{k-1}^{-1} (that is, the matrix formed by deleting the first row and the first column from R_{k-1}^{-1}) and by $(S_k^T \mathbf{u}_{k-1})'$ the first $m_k - 1$ components of the vector $S_k^T \mathbf{u}_{k-1}$. In addition, let $\sigma_{k-1} = 1/(\mathbf{s}_{k-1}^T \mathbf{u}_{k-1})$. Then the new update R_k^{-1} is given by

$$R_k^{-1} = \begin{bmatrix} (R_{k-1}^{-1})' & -\sigma_{k-1}(R_{k-1}^{-1})'(S_k^T \mathbf{u}_{k-1})' \\ 0 & \sigma_{k-1} \end{bmatrix}.$$

Now we give an efficient algorithm for updating the limited memory BFGS matrix D_k and for computing the search direction $\mathbf{d}_k = -D_k \boldsymbol{\xi}_k$. This algorithm is used whenever the previous step was a serious step.

Suppose that the number of current corrections is m_k and that we have the current iteration point $\mathbf{x}_k$, the previous corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$, the current subgradient $\boldsymbol{\xi}_k$ ($\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$ in the case of a serious step), the previous subgradient $\boldsymbol{\xi}_{k-1} \in \partial f(\mathbf{x}_{k-1})$ the $n \times m_k$ matrices S_{k-1} and U_{k-1}, the $m_k \times m_k$ matrices R_{k-1}^{-1}, $U_{k-1}^T U_{k-1}$ and C_{k-1} and the vectors $S_{k-1}^T \boldsymbol{\xi}_{k-1}$ and $U_{k-1}^T \boldsymbol{\xi}_{k-1}$ available. In addition, suppose that we have the previous multiplier ϑ_{k-1}.

Algorithm 5.2. (BFGS Updating and Direction Finding).

Step 1: If $\mathbf{u}_{k-1}^T \mathbf{s}_{k-1} \leq 0$, then set $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$ $C_k = C_{k-1}$ and $\vartheta_k = \vartheta_{k-1}$, compute $S_k^T \boldsymbol{\xi}_k$ and $U_k^T \boldsymbol{\xi}_k$ and go to Step 7.

Step 2: Obtain S_k and U_k by updating S_{k-1} and U_{k-1}.

Step 3: Compute and store m_k-vectors $S_k^T \boldsymbol{\xi}_k$ and $U_k^T \boldsymbol{\xi}_k$.

Step 4: Compute m_k-vectors $S_k^T \mathbf{u}_{k-1}$ and $U_k^T \mathbf{u}_{k-1}$ by using the fact

$$\mathbf{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1}.$$

Step 5: Update $m_k \times m_k$ matrices R_k^{-1}, $U_k^T U_k$ and C_k.

Step 6: If $\mathbf{u}_{k-1}^T \mathbf{u}_{k-1} > 0$, compute ϑ_k

$$\vartheta_k = \frac{\mathbf{u}_{k-1}^T \mathbf{s}_{k-1}}{\mathbf{u}_{k-1}^T \mathbf{u}_{k-1}}.$$

Note that both $\mathbf{u}_{k-1}^T \mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}^T \mathbf{u}_{k-1}$ have already been calculated. Otherwise, set $\vartheta_k = 1.0$.

Step 7: Compute two intermediate values

$$\begin{aligned} \mathbf{p}_1 &= R_k^{-1} S_k^T \boldsymbol{\xi}_k, \\ \mathbf{p}_2 &= (R_k^{-1})^T (C_k \mathbf{p}_1 + \vartheta_k U_k^T U_k \mathbf{p}_1 - \vartheta_k U_k^T \boldsymbol{\xi}_k). \end{aligned}$$

Step 8: Compute

$$\mathbf{d}_k = \vartheta_k U_k \mathbf{p}_1 - S_k \mathbf{p}_2 - \vartheta_k \boldsymbol{\xi}_k.$$

Note that the condition (see Step 1)

$$\mathbf{u}_i^T \mathbf{s}_i > 0 \qquad \text{for all } i \tag{5.6}$$

assures the positive definiteness of matrices obtained by the limited memory BFGS update (see e.g. Byrd et al. (1994)).

In order to use both the Algorithms 5.1 and 5.2 with the same stored information, some modifications have to be done. Firstly, we have to update and store both matrices R_k and R_k^{-1} at each iteration regardless of the update formula we are using. In addition, since we use the same correction matrices S_k and U_k for calculations of both the BFGS and the SR1 updates, we have to test both the positive definiteness conditions (5.4) and (5.6) in each case before we update the matrices. However, the numerical experiments showed that the direct skipping of the updates, if both the required conditions are not satisfied, leads to the situation where the BFGS update is usually skipped due to condition (5.4) required for SR1 update. Therefore, we use the most recent corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$ to calculate the new search direction $\mathbf{d}_k$ whenever the required positive definiteness condition is valid but the matrices are not updated if both of the conditions (5.4) and (5.6) are not satisfied.

Both the Algorithms 5.1 and 5.2 uses at most $O(nm_c)$ operations to calculate the search direction $\mathbf{d}_k$. Supposing that $m_c \ll n$ this is much less than $O(n^2)$ operations needed with the original variable metric bundle method.

5.2 Line Search

In this section, we consider how to calculate a new iteration point $\mathbf{x}_{k+1}$ when the search direction $\mathbf{d}_k$ has been calculated. Similarly to bundle methods and variable metric bundle methods we use the procedure that generates two points: a new iteration point $\mathbf{x}_{k+1}$ and a new auxiliary point $\mathbf{y}_{k+1}$.

The auxiliary points $\mathbf{y}_k$ are determined similarly to the variable metric bundle method for a nonconvex objective function. That is

$$\begin{aligned}\mathbf{y}_1 &= \mathbf{x}_1,\\ \mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}_k, \qquad \text{for all } k \geq 1,\\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}_k, \qquad \text{for all } k \geq 1,\end{aligned}$$

where $t_R^k \in (0, t_I^k]$, $t_L^k \in [0, t_R^k]$ are appropriately chosen step sizes, $t_I^k \in [t_{min}, t_{max})$ is the initial step size and $t_{max} > 1$ and $t_{min} \in (0,1)$ are the upper and lower bound for the initial step size t_I^k. As with the original variable metric bundle method, we have the possibility of using step size greater than 1 here, since the information about the objective function f, included in the compact representation of matrix D_k, is not sufficient for a proper step size determination in the nonsmooth case. The initial step size t_I^k is selected exactly the same way as in the original variable metric bundle method for nonconvex objective functions and a detailed description of the selection of this step size can be found in Vlček and Lukšan (1999).

As before, an essential condition for a serious step to be taken is to have

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \leq -\varepsilon_L t_L^k w_k, \tag{5.7}$$

where $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k > 0$ represents the desirable amount of descent. The parameter w_k is also used as a stopping parameter and we will describe it in Subsection 5.4. If the required condition (5.7) is satisfied, then

$$\mathbf{x}_{k+1} = \mathbf{y}_{k+1}$$

and a serious step is taken. A null step is taken, if

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \mathbf{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{5.8}$$

where $\varepsilon_R \in (\varepsilon_L, 1)$ is a fixed line search parameter and β_{k+1} is the subgradient locality measure similar to bundle methods, that is,

$$\beta_{k+1} = \max\{|f(\mathbf{x}_k) - f(\mathbf{y}_{k+1}) + (\mathbf{y}_{k+1} - \mathbf{x}_k)^T \boldsymbol{\xi}_{k+1})|, \gamma \|\mathbf{y}_{k+1} - \mathbf{x}_k\|^{\omega}\}, \tag{5.9}$$

where $\gamma \geq 0$ ($\gamma = 0$ if f is convex) is a distance measure parameter and $\omega \geq 1$ is a locality measure parameter supplied by the user.

In the case of a null step

$$\mathbf{x}_{k+1} = \mathbf{x}_k$$

but information about the objective function is increased.

The ultimate step sizes t_L^k and t_R^k for the limited memory variable metric bundle method can be determined by using the line search algorithm quite similar to Algorithm 3.3. In fact, the only difference to the algorithm given before is that in order to avoid many consecutive null steps, we have added an additional Step $2\frac{1}{2}$:

Step $2\frac{1}{2}$: (A Test For an Additional Interpolation.) If $f(\mathbf{x}_k + t\mathbf{d}_k) > f(\mathbf{x}_k)$ and $m < k$ and $ind < 10$, then set $ind = ind + 1$ and go to Step 4.

Here k is the number of the current iteration, m is the index of the iteration after the latest serious step and *ind* is the number of the additional interpolation procedures already made at the iteration k.

In practice, the role of this additional step is that if we have already taken a null step at the previous iteration we rather try to find a step size t suitable for a serious step (that is, (5.7) is valid) even if the condition (5.8) required for a null step was satisfied.

5.3 Subgradient Aggregation

In this section, we describe the aggregation procedure used with the limited memory variable metric bundle method.

Likewise with the original variable metric bundle methods, the aggregate values are computed only, if the last step was a null step. Otherwise, we set $\tilde{\boldsymbol{\xi}}_{k+1} = \boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{x}_{k+1})$ and $\tilde{\beta}_{k+1} = 0$.

In principle, the aggregation procedure used with the limited memory variable metric bundle method is the same as that with the original variable metric bundle methods (see Subsection 3.3.3). This means that we minimize the simple quadratic function

$$\begin{aligned}\varphi(\lambda_1, \lambda_2, \lambda_3) = {} & (\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k)^T D_k(\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k) \\ & + 2(\lambda_2\beta_{k+1} + \lambda_3\tilde{\beta}_k)\end{aligned} \quad (5.10)$$

to get the Lagrange multipliers $\lambda_i^k \geq 0$ for $i \in \{1,2,3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ that are used to determine the new aggregate values

$$\begin{aligned}\tilde{\boldsymbol{\xi}}_{k+1} &= \lambda_1^k \boldsymbol{\xi}_m + \lambda_2^k \boldsymbol{\xi}_{k+1} + \lambda_3^k \tilde{\boldsymbol{\xi}}_k \quad \text{and} \\ \tilde{\beta}_{k+1} &= \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k.\end{aligned}$$

Here, as before, $\boldsymbol{\xi}_m \in \partial f(\mathbf{x}_k)$ is the basic subgradient, $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1})$ is the the current auxiliary subgradient, $\tilde{\boldsymbol{\xi}}_k$ is the current aggregate subgradient and β_{k+1} is the current locality measure and $\tilde{\beta}_k$ is the current aggregate locality measure.

However, since the matrix D_k is not formed explicitly here, the practical implementation of the aggregation procedure differs from that of the original method.

The aggregation procedure is computed as follows: We minimize the function (5.10), or $\tilde{\varphi}(\lambda_1, \lambda_2) = \varphi(\lambda_1, \lambda_2, 1 - \lambda_1 - \lambda_2)$. If the intersection of straight lines $\partial\tilde{\varphi}/\partial\lambda_1 = 0$ and $\partial\tilde{\varphi}/\partial\lambda_2 = 0$ is not a feasible point (that is the point, where $\lambda_i^k \geq 0$ for $i \in \{1,2,3\}$ and $\sum_{i=1}^3 \lambda_i^k = 1$), the convexity of $\tilde{\varphi}$ implies that we can restrict our attention to the lines $\lambda_1 = 0$, $\lambda_2 = 0$ and $\lambda_1 + \lambda_2 = 1$ ($\lambda_3 = 0$) (see, Vlček and Lukšan (1999)). For example, if we have the first null step after any serious step we can apply the minimization along the line $\lambda_1 = 0$ due to $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k = \boldsymbol{\xi}_m$ and $\tilde{\beta}_k = 0$. Thus, if $\mathbf{u}_k^T D_k \mathbf{u}_k > 0$, we set

$$\lambda_2^k = \min\left\{1.0, \max\left\{0.0, \frac{\mathbf{d}_k^T \mathbf{u}_k - \beta_{k+1}}{\mathbf{u}_k^T D_k \mathbf{u}_k}\right\}\right\}.$$

Otherwise, we set $\lambda_2^k = 0$ for $\mathbf{d}_k^T \mathbf{u}_k - \beta_{k+1} < 0$ or $\lambda_2^k = 1$ for $\mathbf{d}_k^T \mathbf{u}_k - \beta_{k+1} \geq 0$. Note that we have already calculated $\mathbf{d}_k$, $\mathbf{u}_k$ and β_{k+1}. The product $\mathbf{u}_k^T D_k \mathbf{u}_k$ can be calculated efficiently by using the limited memory BFGS representation of the matrix D_k and the formula given in Byrd et al. (1994). All these calculations can be done within $O(nm_c)$ operations.

In the case of more than one consecutive null steps, the aggregation has to be done by using the limited memory SR1 update. Due to the fact that $\tilde{\boldsymbol{\xi}}_k \neq \boldsymbol{\xi}_m$, the calculations needed to determine the Lagrange multipliers are somewhat more complicated than those given above. However, also in this case all the calculations can be done within $O(nm_c)$ operations. With a large number of variables ($n \gg m_c$) this is much less that $O(n^2)$ operations used with the original variable metric bundle methods.

Similarly to the original variable metric bundle methods, the aggregation procedure used with the limited memory variable metric bundle method uses only three subgradients and two locality measures to calculate the new aggregate values. Thus, the minimum size of the bundle is 2 and, as before, a larger bundle is only used in the selection of the step sizes.

5.4 Stopping Criterion

Similarly to the other bundle methods (see Subsection 3.2.3), we use the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ to get some approximation of the gradient of the objective function. As before, the direct test $\|\tilde{\boldsymbol{\xi}}_a^k\| < \varepsilon$ for some $\varepsilon > 0$ is too uncertain and, therefore, we use the approximation D_k of the inverse of the Hessian matrix and the aggregate subgradient locality measure $\tilde{\beta}_a^k$ to improve the accuracy of the norm of the aggregate subgradient. Since in practice, the matrix D_k is not formed explicitly we use the direction vector $\mathbf{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k$ instead. Thus, the stopping parameter at iteration k is defined by

$$w_k = -2\tilde{\boldsymbol{\xi}}_k^T \mathbf{d}_k + 4\tilde{\beta}_k \qquad (5.11)$$

and the stopping criterion is

$$\text{If } w_k < \varepsilon, \text{ for given } \varepsilon > 0, \text{ then stop.}$$

The multipliers 2 and 4 in (5.11) are chosen experimentally such that the accuracy of the new method would be approximately the same as with the methods described in Chapter 3.

5.5 Algorithm

We are now ready to present the limited memory variable metric bundle method for nonsmooth large-scale unconstrained optimization.

Algorithm 5.3. (L-Variable Metric Bundle Method).

Data: Select positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$. Choose a final accuracy tolerance $\varepsilon \geq 0$, a distance measure parameter $\gamma > 0$ ($\gamma = 0$ if f is convex) and a locality measure parameter $\omega \geq 1$. Choose the maximum number of stored corrections m_c.

Step 0: (*Initialization.*) Choose a starting point $\mathbf{x}_1 \in \mathbb{R}^n$. Set $\beta_1 = 0$ and $\mathbf{y}_1 = \mathbf{x}_1$. Compute

$$f_1 = f(\mathbf{x}_1) \qquad \text{and}$$
$$\boldsymbol{\xi}_1 \in \partial f(\mathbf{x}_1).$$

Set the iteration counter $k = 1$.

Step 1: (*Serious step initialization.*) Set an aggregate subgradient $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$ and an aggregate subgradient locality measure $\tilde{\beta}_k = 0$. Set an index variable for null steps $m = k$.

Step 2: (*Direction finding.*) Compute

$$\mathbf{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k$$

by Algorithm 5.2, if $m = k$ (BFGS update) and by Algorithm 5.1 otherwise (SR1 update). Note that $\mathbf{d}_1 = -\tilde{\boldsymbol{\xi}}_1$.

Step 3: (*Stopping criterion.*) Set

$$w_k = -2\tilde{\boldsymbol{\xi}}_k^T \mathbf{d}_k + 4\tilde{\beta}_k.$$

If $w_k \leq \varepsilon$, then stop.

Step 4: (*Line search.*) Determine step sizes t_L^k and t_R^k to obtain either a serious step or a null step (that is, either (5.7) or (5.8) is valid). Set the corresponding values

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}_k, \\
\mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}_k, \\
f_{k+1} &= f(\mathbf{x}_{k+1}), \\
\boldsymbol{\xi}_{k+1} &\in \partial f(\mathbf{y}_{k+1}).
\end{aligned}$$

Set $\mathbf{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$ and $\mathbf{s}_k = \mathbf{y}_{k+1} - \mathbf{x}_k = t_R^k \mathbf{d}_k$. If $t_L^k > 0$ (serious step), set $\beta_{k+1} = 0$, $k = k+1$ and go to Step 1. Otherwise, calculate the locality measure β_{k+1} by (5.9).

Step 5: (*Aggregation.*) Determine multipliers $\lambda_i^k \geq 0$, $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function

$$\begin{aligned}
\varphi(\lambda_1, \lambda_2, \lambda_3) = \; & (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)^T D_k (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k) \\
& + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k),
\end{aligned}$$

where D_k is obtained by the BFGS update, if $m = k$ and by the SR1 update, otherwise. Set

$$\begin{aligned}
\tilde{\boldsymbol{\xi}}_{k+1} &= \lambda_1^k \boldsymbol{\xi}_m + \lambda_2^k \boldsymbol{\xi}_{k+1} + \lambda_3^k \tilde{\boldsymbol{\xi}}_k \qquad \text{and} \\
\tilde{\beta}_{k+1} &= \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k.
\end{aligned}$$

Set $k = k+1$ and go to Step 2.

Note that in Steps 2 and 5 the matrices D_k are not formed explicitly but the search direction $\mathbf{d}_k$ and the aggregate values $\tilde{\boldsymbol{\xi}}_{k+1}$ and $\tilde{\beta}_{k+1}$ are calculated using the difference matrices S_k and U_k (see Sections 5.1 and 5.3).

As said before, the limited memory variable metric bundle method uses the simple aggregation procedure and requires only three subgradients and two locality measures to calculate the new aggregate values. Thus, the time-consuming quadratic subproblem (3.15) appearing in standard bundle methods need not to be solved. Furthermore, both the search direction $\mathbf{d}_k$ and the aggregate values $\tilde{\boldsymbol{\xi}}_{k+1}$ and $\tilde{\beta}_{k+1}$ can be computed implicitly using at most $O(nm_c)$ operations. Assuming $m_c \ll n$ this is much less than $O(n^2)$ operations needed with the original variable metric bundle method, which stores and manipulates the whole matrix D_k. These improvements make the limited memory variable metric bundle method suitable for large-scale problems. This assertion is supported by numerical tests to be presented in next chapter.

6 Numerical Experiments

In order to get some kind of an impression of how the different methods described earlier operate in practice, we tested them with several problems. In this chapter we first introduce the software and the test problems used and then we draw conclusions from numerical experiments.

6.1 Tested Software

In this section we introduce the programs used in our experiments. All the methods included are described in Chapters 3, 4 and 5. The experiments were performed in a SGI Origin 2000/128 supercomputer (MIPS R12000, 600 Mflop/s/processor). The algorithms were implemented in FORTRAN77 with double-precision arithmetic. The pieces of software tested are presented in Table 1.

Variable Metric Bundle Method. The tested variable metric bundle algorithm `PVAR` is from the software package `UFO` (Universal Functional Optimization, see Lukšan et al. (2000)). The basic ideas of the variable metric bundle methods are described in Section 3.3. For more details, we refer to Lukšan and Vlček (1999a) and Vlček and Lukšan (1999).

Table 1: Tested pieces of software

Software	Author(s)	Method
`PVAR`	Lukšan & Vlček	Variable metric bundle
`PNEW`	Lukšan & Vlček	Bundle-Newton
`PBUN`	Lukšan & Vlček	Proximal bundle
`PBNCGC`	Mäkelä	Proximal bundle
`L-BFGS`	Nocedal	Limited memory BFGS
`LVMBM`	Haarala	L-Variable metric bundle

Bundle-Newton Method. The tested bundle-Newton algorithm `PNEW` is also from the software package `UFO`. The program utilizes the subgradient aggregation strategy of Kiwiel (1985) to bound the number of stored subgradients and it employs the solver `ULQDF1` implementing the dual projected gradient method proposed in Lukšan (1984) for solving the quadratic subproblem. The Hessian matrices were calculated numerically using difference approximations. The bundle-Newton method is shortly described in Section 3.4. For a more detailed description we refer to Lukšan and Vlček (1998).

Proximal Bundle Methods. The first tested proximal bundle algorithm `PBUN` is also from the software package `UFO`. The program utilizes as well the subgradient aggregation strategy of Kiwiel (1985) and uses the solver `ULQDF1` to solve quadratic subproblems. The survey of the bundle methods is given in Section 3.2. For a more detailed description of the proximal bundle algorithm `PBUN` we refer to Vlček (1995) and Lukšan and Vlček (2000b).

The second tested proximal bundle algorithm `PBNCGC` is from the software package `NSOLIB` (NonSmooth Optimization LIBrary, see Mäkelä (1993)). Also this program utilizes the subgradient aggregation strategy of Kiwiel (1985). For solving the quadratic subproblem, the program employs the quadratic solver `QPDF4`, which is based on the dual active-set method described in Kiwiel (1986). For a detailed description of the method see Mäkelä and Neittaanmäki (1992).

Limited memory BFGS Method. `L-BFGS` is a limited-memory quasi-Newton program for smooth large-scale unconstrained optimization. The program has been developed at the Optimization Technology Center as a joint venture of Argonne National Laboratory and the Northwestern University. The basic ideas of the smooth limited memory variable metric methods are given in Section 4.1 and for a detailed description of the method we refer to Nocedal (1980) and Liu and Nocedal (1989).

Limited Memory Variable Metric Bundle Method. The limited memory variable metric bundle method `LVMBM` is our new method for nonsmooth large-scale unconstrained optimization. This method is described in detail in Section 5. The implementation of the program is our own expect the bundle updating and the step size selections, where we have used the original pieces by Lukšan and Vlček.

In every program the optimization was terminated if any of the following conditions was satisfied:

- The maximum number of iterations were performed.
- The maximum number of function calls were performed.
- The problem was solved with the desired accuracy, that is, $w_k \le \varepsilon$, where $w_k = \frac{1}{2}\|\tilde{\boldsymbol{\xi}}_a^k\|^2 + \tilde{\beta}_a^k$ with the bundle-Newton and both the proximal bundle methods (`PNEW`, `PBUN` and `PBNCGC`), $w_k = \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k$ with the variable metric bundle method (`PVAR`), $w_k = 2\tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 4\tilde{\beta}_k$ with the limited memory variable metric bundle method (`LVMBM`) and $w_k = \|\nabla f(\mathbf{x})\| / \max\{1, \|\mathbf{x}\|\}$ with the limited memory BFGS method (`L-BFGS`).

In addition, for the `UFO` programs and for the limited memory variable metric bundle program `LVMBM` the optimization was terminated if

- $|f_{k+1} - f_k| \le 1.0 \cdot 10^{-8}$ in 10 subsequent iterations,

and for solely the `UFO` programs also if

- $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \le 1.0 \cdot 10^{-16}$ in 20 subsequent iterations.

6.2 Test Problems

The optimization problems used in the numerical experiments are listed in Table 2 together with references. All these problems can be formulated with any number of variables (n).

None of the nonsmooth optimization methods `PVAR`, `PBUN`, `PNEW` and `PBNCGC` has been developed for large-scale optimization. On the other hand, we wanted to get some information of the behavior of the new method `LVMBM` especially with large-scale problems. For this reason, we used the smooth

Table 2: Test problems

No	Problem	References
1	Chained Rosenbrock function	Lukšan and Vlček (1999b)
2	Chained Wood function	Lukšan and Vlček (1999b)
3	Chained Powel singular function	Lukšan and Vlček (1999b)
4	Chained Cragg and Levy function	Lukšan and Vlček (1999b)
5	Generalized Broyden tridiagonal function	Lukšan and Vlček (1999b)
6	Generalized Broyden banded function	Lukšan and Vlček (1999b)
7	7-dimensional generalization of the Broyden tridiagonal function	Lukšan and Vlček (1999b)
8	Sparse modification of the Nazareth trigonometric function	Lukšan and Vlček (1999b)
9	Another trigonometric function	Lukšan and Vlček (1999b)
10	Toint trigonometric function	Lukšan and Vlček (1999b)
11	Augmented Lagrangian function	Lukšan and Vlček (1999b)
12	Generalization of the Brown function 1	Lukšan and Vlček (1999b)
13	Generalization of the Brown function 2	Lukšan and Vlček (1999b)
14	Discrete boundary value problem	Lukšan and Vlček (1999b)
15	Discretization of a variational problem	Lukšan and Vlček (1999b)
16	Banded trigonometric problem	Lukšan and Vlček (1999b)
17	Variational problem 1	Lukšan and Vlček (1999b)
18	Variational problem 2	Lukšan and Vlček (1999b)
19	Variational problem 3	Lukšan and Vlček (1999b)
20	Variational problem 4	Lukšan and Vlček (1999b)
21	Variational problem 5 (Calvar 1)	Lukšan and Vlček (1999b)
22	Variational problem 6 (Calvar 2)	Lukšan and Vlček (1999b)
23	Image restoration problem 1	Kärkkäinen et al. (2000)
24	Image restoration problem 2	Kärkkäinen et al. (2000)
25	Nonsmooth chained Rosenbrock function	Grothey (2001)
26	n1actfs	Grothey (2001)

optimization method `L-BFGS` as a benchmark. Thus, the programs were first tested with a set of smooth minimization problems 1 – 22. A detailed description of these smooth problems used can be found in Lukšan and Vlček (1999b).

In addition, the programs for nonsmooth optimization (all the programs in Table 1 except `L-BFGS`) were tested with four nonsmooth minimization problems 23 – 26. A detailed description of convex image restoration problems 23 and 24 can be found in Kärkkäinen et al. (2000) and a description of nonconvex problems 25 and 26 can be found in Grothey (2001).

6.3 Numerical Results

The conclusions of the numerical experiments are given in the next two sections where the smooth and the nonsmooth tests are discussed separately.

6.3.1 Smooth Problems

The programs given in Table 1 were tested with the set of smooth problems with the numbers of variables 10, 100 and 1000 and in case of the limited memory programs `L-BFGS` and `LVMBM` also with the number of variables 10000.

We tested the bundle methods `PVAR`, `PNEW`, `PBUN`, `PBNCGC` and `LVMBM` with different sizes of bundles (m_ξ) and the limited memory methods `L-BFGS` and `LVMBM` also with different numbers of stored corrections (m_c). The tested sizes of bundles were 10 and $n+3$ for bundle-Newton method `PNEW` and for both proximal bundle methods `PBUN` and `PBNCGC` and 2 and $n+3$ for variable metric bundle methods `PVAR` and `LVMBM`. For the limited memory methods `L-BFGS` and `LVMBM` the maximum number of stored corrections were first set to 3 and then to 7. In what follows, we denote these different modifications by `L-BFGS(3)`, `L-BFGS(7)`, `LVMBM(3)` and `LVMBM(7)`, when we need to separate them.

As a stopping criterion, we used $\varepsilon = 10^{-6}$ in all the cases and the maximum number of iterations was set to 20000 with $n \leq 1000$ and to 50000 with $n = 10000$. In both cases the maximum number of function evaluations was set to 50000. The other parameters used were chosen experimentally (for details see Appendix C). The results are summarized in Tables 3 – 24 of Appendix A. Some of the results are also displayed in Figures 2 and 3. In Figure 2, we give the CPU time elapsed for the set of smooth problems in proportion to the number of variables for all the programs with the small bundle. In Figure 3, we have calculated the average amount of iterations needed for problems of different sizes. The problems where the optimization has failed are not included in data. Since for almost every tested program there existed one problem in the set of problems which needed much more iterations than the others, we also removed in all cases the problem which used most iterations.

All the tested programs worked well for small- and medium-scale problems ($n \leq 100$). However, when the dimension of the problems increased the computational time of the methods other than those using limited memory approach expanded rapidly (see Figure 2).

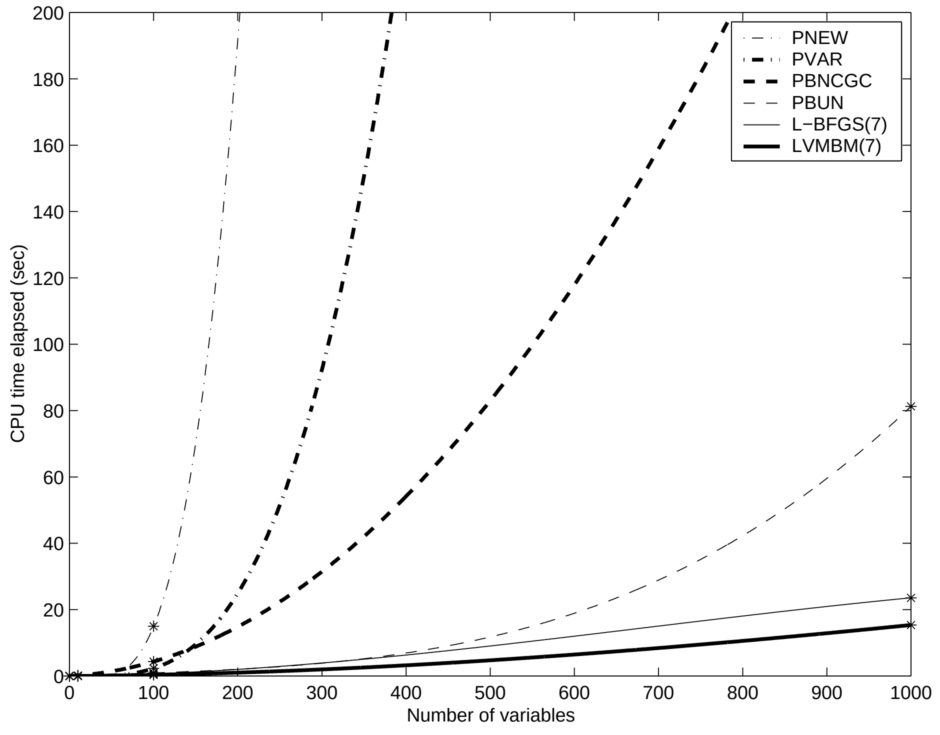

Figure 2: CPU time elapsed for the set of smooth problems.

The smallest number of used iterations was usually with the bundle-Newton program PNEW (see Figure 3). However, due to the matrix operations each individual iteration was more costly than with the other methods tested. So, the computational time of PNEW was able to compete with the other methods only with a small number of variables. Even for the problem dimension $n = 100$, the program PNEW was clearly the most time-consuming of the programs tested (see Figure 2).

Even if the program PVAR usually used less iterations and function calls than the proximal bundle methods PBUN and PBNCGC, it used less CPU time only with the number of variables less or equal to 100 (see Figure 2). The reason for this is that also the variable metric bundle method PVAR uses time-consuming matrix operations (nevertheless not so many as PNEW).

Since the approximation of the inverse of the Hessian matrix is not so accurate with the limited memory variable metric bundle method LVMBM as in the original variable metric bundle method PVAR, we were expecting that the amount of iterations used with the limited memory program LVMBM would have been greater. Yet, quite surprisingly, the program LVMBM usually used less iterations than the program PVAR. This might be due to the fact that the line search of the program LVMBM does not willingly take consecutive null steps but rather tries to find the step size suitable for a serious step. The computational times of these two methods were equal with the small number of variables ($n = 10$) but, as said before, when the dimensions of the problems increased the computational time of the program PVAR expanded rapidly. For all tested numbers of variables the new limited memory variable metric bundle method LVMBM with the small bundle was the most efficient method (see Figure 2).

With all the tested numbers of variables the program LVMBM(7) (with $m_\xi = 2$) was about 1.7 times faster than the other limited memory program L-BFGS(7). Since both the limited memory programs use only $O(nm_c)$ operations to calculate the search direction, their computational times should linearly depend on n (with some fixed m_c). However, as it can be seen in Figure 2, the computational times of these methods are not exactly linear. This is due to the fact that with both the methods the number of iterations used increased with the dimension of the problem.

For $n = 100$ the limited memory variable metric bundle program LVMBM(7) (with $m_\xi = 2$) was about 3 times faster than the program PBUN (with $m_\xi = 10$), 13 times faster than the program PBNCGC (with $m_\xi = 10$), 7 times faster than the program PVAR (with $m_\xi = 2$) and 43 times faster than the program PNEW (with $m_\xi = 10$), and for $n = 1000$, the corresponding values were 5, 21, 253 and 2093, respectively.

Although the programs PBUN and PBNCGC are realizations of the same method, the difference in their functioning is remarkable. PBUN was clearly more efficient when comparing the iteration numbers, function evaluations and CPU times. However, PBUN had slight difficulties to reach the desired accuracy in larger problems.

As it can be seen in Figure 3, the number of used iterations and function calls of PNEW did not usually depend on the dimension of the problem. Here the chained Rosenbrock function (problem 1) was an exception (see Tables 4, 11 and 18 of Appendix A). On the other hand, when the dimension of the problem became ten times larger, PBUN needed about 5.2 times, PBNCGC about 5.9 times, PVAR about 7.2 times, LVMBM about 4.1 times and L-BFGS about 5.0 times more iterations and function evaluations.

With both proximal bundle methods PBNCGC and PBUN the number of iterations usually decreased when the size of the bundle was increased. However, with large problems the CPU time grew with the size of the bundle. This is due to the quadratic programming subproblem (3.15) which becomes very

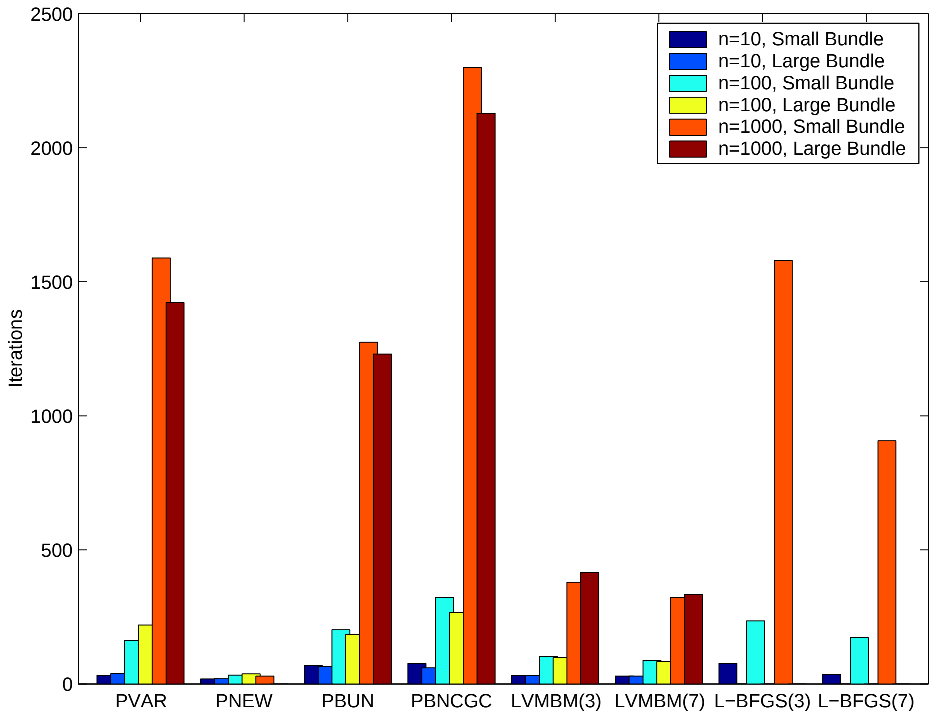

Figure 3: Average iterations used for the smooth problems.

time-consuming when the size of the problem and the size of the bundle increase. With the program PVAR there was not so large a change in CPU times when the size of the bundle was increased as with the proximal bundle programs PBUN and PBNCGC and with the bundle-Newton program PNEW. In fact, the CPU time of PVAR was approximately doubled in each case when the size of the bundle was increased from 2 to $n + 3$. This due to the fact that the variable metric bundle method uses only three subgradients and two locality measures to solve the quadratic subproblem (3.22). That means the minimum size of the bundle is 2, and the larger bundle is used only for the step size selections, which is not a time-consuming procedure. With the limited memory variable metric bundle program LVMBM the number of iterations was approximately the same regardless of the size of the bundle but the computational time expanded rapidly when the size of the bundle was increased. On closer consideration, we noticed that the CPU time increase of LVMBM was always less than that of the original variable metric bundle program PVAR. This is due to fact that the bundle updating in these programs is exactly the same and the line search in the program LVMBM finds more serious steps and, thus, the number of iterations used is smaller than with the program PVAR.

Anyhow, with the program LVMBM the results obtained with the small bundle ($m_\xi = 2$) were in all cases as good as with the larger bundle and, thus, there is no reason to use a larger bundle.

The iterations needed with the program LVMBM were approximately half of that needed with the other limited memory program L-BFGS. Both of the programs worked well with both of tested number of stored corrections $m_c = 3$ and $m_c = 7$. However, with the program L-BFGS the number of iterations used and the CPU time elapsed were always less with the larger number of stored corrections. The same effect can be seen with the program LVMBM but it is not so evident and the computational times are approximately the same with both the numbers of stored corrections.

In order to see what kind of an effect the choice of parameter values had, the smooth problems 1 – 22 were also tested with the default parameters of the programs. For all the UFO programs PVAR, PBUN and PNEW and for both the limited memory programs LVMBM and L-BFGS there exist some recommended values for parameters in the implementation of these programs. In the implementation of the proximal bundle method PBNCGC there exists no such recommended values. Thus, the default parameters for PBNCGC used in our experiment were chosen on the basis that they had been found to be good in our previous work. In these experiments, the number of variables was set to 500 and we tested the problems first with the default values of the parameters (see Appendix B) and then with the experimentally chosen values of the parameters (see Appendix C). The results are collected in Tables 25 – 30 of Appendix A.

With the variable metric bundle program PVAR there were five failures when the default parameters were used. This program also had some difficulties to recognize a good solution: Even though there were nine problems that according to the termination parameter were solved with the desired accuracy, three of these results were clearly worse than those obtained with the other methods. Note that when parameters were tuned PVAR succeeded to solve all the problems properly.

With the bundle-Newton program PNEW, 18 problems from 22 were solved with the desired accuracy when the default parameters were used. Three of the problems that failed were terminated because the value of $\mathbf{x}$ did not change and two of these could not be solved with the desired accuracy even with the selected parameters. In addition, with one problem the optimization was terminated because the value of the objective function did not change. This problem could not be solved with the desired accuracy even with the selected parameters. Thus, with selected parameters the program `PNEW` succeeded to solve 19 problems with the desired accuracy.

The proximal bundle program `PBUN` was maybe the worst program tested when the default parameters were used. First of all, the program jammed when trying to solve problem 4. In addition, there were three failures in the method and also this program had some difficulties to recognize a good solution: Even though there were nine problems that according to the termination parameter were solved with the desired accuracy, six of these results were clearly worse than those obtained with the other methods. Note that with selected parameters the program `PBUN` succeeded to solve 20 problems with the desired accuracy. However, also in this case the values of the objective functions were often slightly greater than those obtained with the other methods.

The proximal bundle program `PBNCGC`, on the other hand, was the best method to be used with the default parameters. It failed to reach the desired accuracy only once and that was because the maximum number of function calls was not big enough. Otherwise, the program `PBNCGC` solved all the problems with the desired accuracy also with the default parameters. The numbers of used iterations and function calls were a little bit smaller when the selected parameters were used.

The limited memory BFGS program `L-BFGS` solved 19 problems with the desired accuracy when the default parameters were used. With three problems the program had a failure during the line search. Two of these problems could not be solved even with the selected parameters. Thus, with selected parameters the program `L-BFGS` succeed to solve 20 problems with the desired accuracy.

The new limited memory variable metric bundle program `LVMBM` was quite robust in our experiments. It solved 21 problems with the desired accuracy both with the default and the selected parameters. With one problem the program failed to reach the desired accuracy but also in this case the value of the objective function was as small as with the other methods. However, the problems used in these experiments were the same that have been used when testing the method in the construction phase and, thus, these results should be considered cautiously.

To sum up, for smooth problems the proximal bundle program `PBNCGC` found the local minimum in the most reliable way also with the small bundle. It was clearly the most robust program tested. However, with large-scale problems it was not computationally efficient.

The proximal bundle program PBUN was the most efficient of the preceding nonsmooth methods tested when the dimension of the problem was greater than 100. However, this method had some difficulties to reach the desired accuracy and it was very sensitive to the choice of parameters.

The variable metric bundle program PVAR was the most efficient method tested for small problems (together with LVMBM). However, its computational time increased rapidly with the dimension of the problem. The method was quite reliable but also this method needed a careful choice of parameters.

The bundle-Newton program PNEW was quite efficient with small problems. It was also reliable even with the default parameters. In most cases it found the minimum with the smallest objective function value. However, computational time of PNEW increased drastically with the dimension of the problem.

The limited memory BFGS program L-BFGS was an efficient method both with small- and large-scale problems. In many cases, the minimum with the smallest objective function value was found with this method. However, when the dimension of the problems increased the program had some difficulties with the line search and it failed to solve some of the problems.

In all the cases, the limited memory variable metric bundle method LVMBM with the small bundle was the most efficient method tested. It found the local minimum in a reliable way and in our experiments it was also very robust.

6.3.2 Nonsmooth Problems

The programs for nonsmooth optimization (all the programs in Table 1 except L-BFGS) were tested also with four nonsmooth minimization problems 23 – 26. The numbers of variables (n) used were 100, 300, 1000, 3000 and 10000 for convex problems 23 and 24 and 100 and 300 for nonconvex problems 25 and 26.

As with smooth problems, we tested the programs with different sizes of bundles (m_ξ) and in case of the limited memory variable metric bundle method LVMBM also with different numbers of stored corrections (m_c). For bundle-Newton method PNEW and for both the proximal bundle methods PBUN and PBNCGC the used sizes of bundles were 10, 50 and 100 and for variable metric bundle methods PVAR and LVMBM the used sizes of bundles were 2, 50 and 100. For the program LVMBM the maximum numbers of stored corrections were set to 3, 7 and 15 and, as before, we denote these different modifications by LVMBM(3), LVMBM(7) and LVMBM(15).

In our experiments, we used the following values of parameters:

- For convex problems 23 and 24, the distance measure parameter $\gamma = 0.0$ was used with the programs PBUN, PBNCGC and LVMBM.

  With the bundle-Newton program PNEW the distance measure parameter $\gamma = 0.1$ was used since the value of γ has to be positive. Also the distance measure exponent ω was set to 2 with the program PNEW.

  With the program PVAR the convex version of the program was used.

- For nonconvex problems 25 and 26 the nonconvex version of the program PVAR was used. The distance measure parameter $\gamma = 0.50$ was used with all the programs. Also for these problems the distance measure exponent ω was set to 2 with the program PNEW.

- For the proximal bundle programs PBUN and PBNCGC the line search parameter $\varepsilon_L = 0.05$ was used for problems 23 and 24. Otherwise, $\varepsilon_L = 0.01$ (default value) was used.

  For the limited memory variable metric bundle program LVMBM the line search parameter $\varepsilon_L = 0.05$ was used for problems 23 – 25. For problem 26, the default value $\varepsilon_L = 0.01$ was used.

- For all UFO programs (PVAR, PNEW and PBUN) the maximum step size XMAX was set to 1000 (default value) for problems 23 and 24 and to 2 for others.

  For limited memory variable metric bundle program LVMBM the maximum step size XMAX was set to 10 for problem 25. Otherwise, the default value XMAX $= 2$ was used.

- The maximum number of iterations and function evaluations were set to 20000 and 50000 for PVAR, LVMBM, PBUN and PBNCGC, and to 2000 and 5000 for PNEW, respectively.

  With the largest problems ($n = 10000$) both the maximum number of iterations and the maximum number of function evaluations were set to 50000.

- The stopping criterion was $\varepsilon = 10^{-5}$ when the number of variables was 100 and $\varepsilon = 10^{-4}$ in all the other cases.

All the programs were tested with all the problems when the number of variables was smaller or equal to 300. With larger problems we stopped the experiments if the CPU time used exceeded one hour. In practice this meant that the program PNEW was tested only with medium-scale problems ($n = 100$ and $n = 300$) and the largest problems solved with the program PVAR had the number of variables $n = 1000$. For the proximal bundle methods PBUN and PBNCGC the used maximum number of iterations (20000) was not big enough to solve the problems with more than 1000 variables. Thus, the larger problems (problems 23 and 24, $n \geq 3000$) were tested only with the new limited memory variable metric bundle method LVMBM. The largest problems (problems 23 and 24, $n = 10000$) were tested only with bundles of small and medium sizes ($m_\xi = 2$ and $m_\xi = 50$).

The results of the experiments are summarized in Tables 31 – 44 of Appendix A. Some of the results are also displayed in Figures 4 and 5. In Figure 4, we give the CPU times of problem 23 versus the number of variables for all the programs with the small bundle. The results where the maximum number of iterations was not big enough are not included on the data. In Figure 5, we give the number of iterations used for problem 23. These results are displayed for all the programs with all the used sizes of bundles and for 100, 300 and 1000 variables.

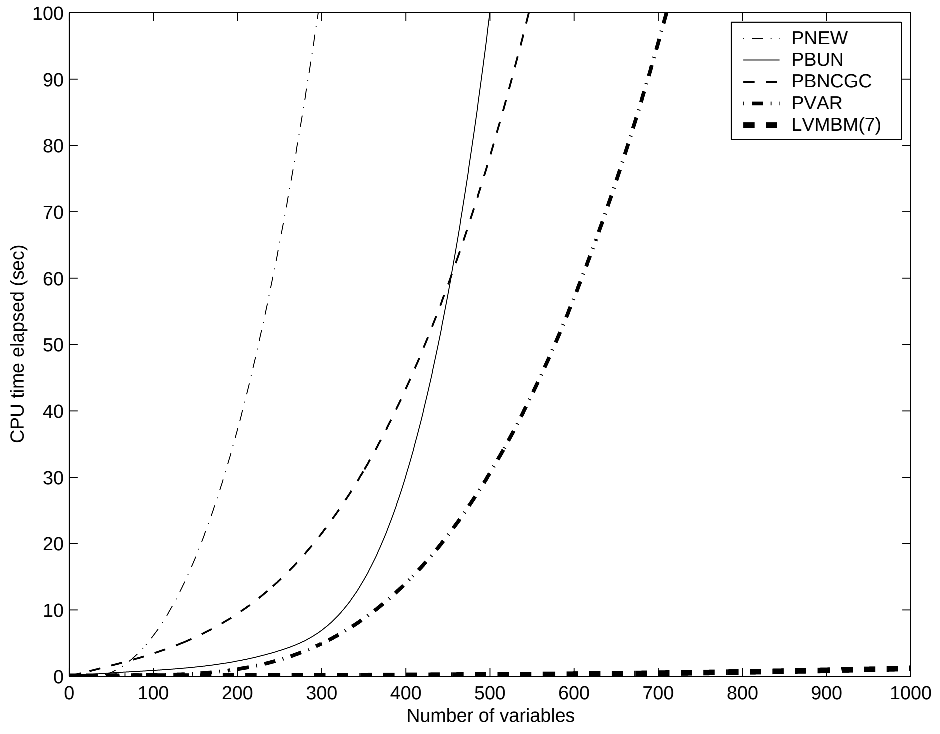

Figure 4: CPU time elapsed for the problem 23.

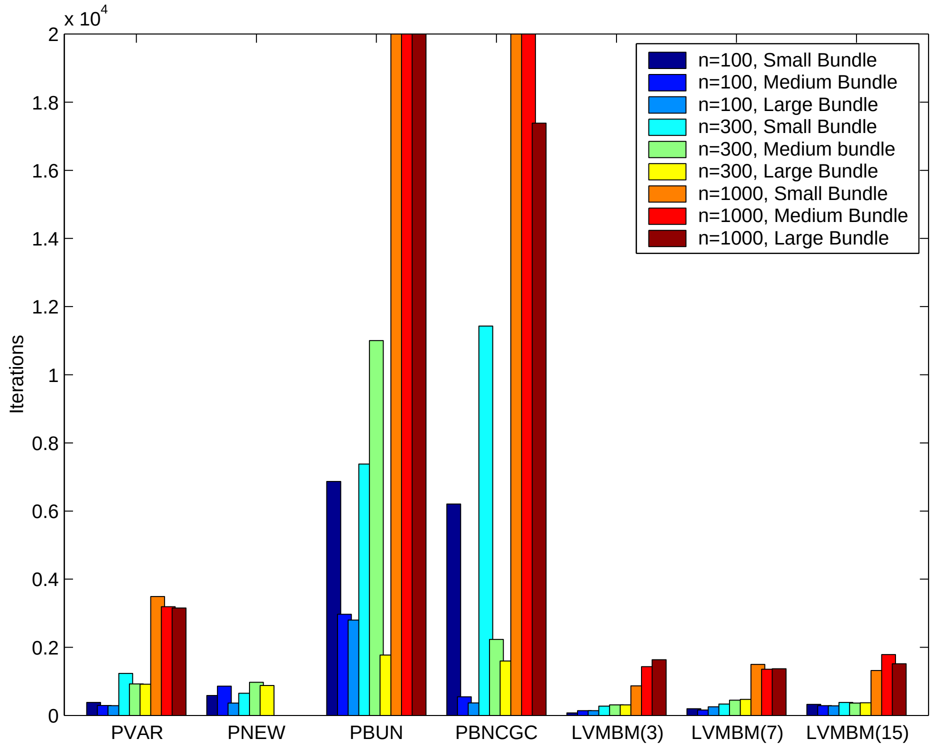

Figure 5: Iterations used for the problem 23.

From the numerical results we can conclude the superiority of the limited memory variable metric bundle program LVMBM when comparing the computational times. In all the cases, it used less CPU time and iterations than the other methods. However, the minimum of the objective function found with this method was usually a little bit greater than with the other methods.

The results of the programs obtained with the other nonsmooth problems behaved quite similarly to those of problem 23 (see Figures 4 and 5). For example, with the problem 24 for $n = 100$ the limited memory variable metric bundle program LVMBM(7) (with $m_\xi = 2$) was about 6 times faster than the program PVAR (with $m_\xi = 2$), 70 times faster than the program PBUN (with $m_\xi = 10$), 110 times faster than the program PBNCGC (with $m_\xi = 10$) and 230 times faster than the program PNEW (with $m_\xi = 10$). For $n = 300$, the corresponding values were 50, 110, 220 and 910, respectively. For $n = 1000$ the program LVMBM(7) was about 230 times faster than PVAR. With the proximal bundle programs PBUN and PBNCGC the optimization were terminated since the maximum number of iterations was reached and, as said before, the program PNEW was not tested with $n > 300$.

With the limited memory variable metric bundle program `LVMBM` the number of iterations was approximately the same regardless of the size of the bundle (m_ξ) or the number of stored corrections (m_c). As with smooth problems, the computational time of the program increased with the size of the bundle, but the results obtained with the small bundle were in all cases at least as good as with the larger bundles and, thus, there is no reason to use the size of the bundle greater than 2. With all the tested problems the computational time of the program `LVMBM` increased also with the number of stored corrections. With nonsmooth problems the values of the objective function obtained with the small number of stored corrections ($m_c = 3$) were in each case worse than those obtained with larger number of stored corrections. This result was even more evident with nonconvex problems. In fact, the new method had some difficulties to recognize a good solution in nonconvex case especially with small number of stored corrections. Thus, with nonsmooth and especially with nonconvex problems the number of stored corrections should be at least 7.

The variable metric bundle program `PVAR` worked well for problems with no more than 1000 variables. In these cases, it used less CPU time than the other methods except the program `LVMBM` and it usually needed fewer iterations than the other methods. It also found the minimum in the most reliable way. However, when the dimension of the problems increased the computational time of the program `PVAR` expanded rapidly.

Also with the bundle-Newton program `PNEW` the total amount of iterations stayed quite low. However, each individual iteration was so costly that `PNEW` was the most time-consuming of the methods tested already with the smallest problems ($n = 100$). In addition, in our experiments there were some failures and inaccurate results with this program. However, Mäkelä et al. (1999) have reported the superiority of the program `PNEW` when compared to the proximal bundle programs `PBUN` and `PBNCGC` when solving hemivariational inequalities. In their results the program `PNEW` was the most efficient and the most reliable method tested also with the large-scale problems.

In contrast to the results of smooth problems, the program `PBNCGC` usually required fewer iterations and function evaluations than the other proximal bundle program `PBUN`. The program `PBNCGC` was also quite reliable: if the maximum number of iterations was big enough, it found the local minimum in each case. With the program `PBUN` some inaccurate results occurred. With both the proximal bundle programs the number of iterations needed and the computational times elapsed expanded rapidly with the dimension of the problems. Each individual iteration of the program `PBNCGC` was more costly than that of PBUN is and, thus, with medium-scale problems $n \leq 300$ the program PBUN used less CPU time than the program PBNCGC.

All the tested programs solved the convex image restoration problems 23 and 24 properly for medium-scale cases ($n = 100$ and $n = 300$) (see Tables 31, 32, 36 and 37 of Appendix A). With the bundle-Newton program PNEW the optimization was usually terminated because the value of $\mathbf{x}$ did not change enough but also in these cases the minimum values of the objective functions were approximately as small as with the other programs.

For $n = 1000$, the variable metric bundle programs PVAR and LVMBM with all sizes of bundle and the proximal bundle program PBNCGC with the largest bundle solved the problems 23 and 24 with the desired accuracy. For the other proximal bundle program PBUN and for the program PBNCGC with smaller bundles the maximum number of iterations (20000) was reached. Thus, we tested the proximal bundle programs PBUN and PBNCGC also with 50000 iterations. In these cases, the CPU time used by program PBNCGC exceeded one hour and for program PBUN even this amount of iterations was not big enough.

For $n = 3000$ and $n = 10000$, the only method tested was the new limited memory variable metric bundle program LVMBM. The program solved both problems 23 and 24 properly also in these larger cases.

For $n = 100$, the variable metric bundle method PVAR with all sizes of bundle, both the proximal bundle methods PBUN and PBNCGC with the largest bundle and the limited memory variable metric bundle method LVMBM with the largest number of stored corrections succeeded to solve the nonsmooth chained Rosenbrock problem 25 with the desired accuracy (see Table 41 of Appendix A). With 3 stored corrections the limited memory variable metric bundle program LVMBM had some serious difficulties to recognize a good solution. That is that according to termination parameter the problem was solved with the desired accuracy but the value of the objective function at termination was much worse than that obtained with the other programs. With 7 stored corrections the results obtained with the program LVMBM were already clearly better than those obtained with 3 stored corrections and, as mentioned above, with 15 stored corrections the program LVMBM succeeded to solve problem 25 with the desired accuracy.

For $n = 300$, only the program PVAR succeeded to solve problem 25 properly, although also this program had some difficulties to recognize a good solution when the smallest bundle was used. As before, the values of the objective function obtained with the program LVMBM were much larger than they should have been and with 3 stored corrections they were not even close to an optimal solution (see Table 42 of Appendix A).

With problem 26 the programs PBUN and LVMBM converged to a different local minimum than the other programs. In these cases, the optimization was terminated because the value of the objective function did not change. The programs PBNCGC and PVAR converged to the global minimum of the objective function and solved the problem with the desired accuracy. The program PNEW failed to minimize this problem (see Tables 43 – 44 of Appendix A).

The new limited memory variable metric bundle method LVMBM was clearly the most efficient method tested for large-scale problems. For convex problems the new method found the minimum of the objective function in a reliable way. In nonconvex case, the new method had some difficulties to recognize a good solution especially with the small maximum number of stored corrections. Thus, it seems that with nonsmooth and especially with nonconvex problems the maximum number of stored corrections should be at least 7. Also the accuracy of the method still needs some modifications so that it can better be compared with the other methods. The other programs tested were quite efficient for the smallest problems used in the experiments. However, the computational times of these methods expanded rapidly with the dimension of the problem.

7 Conclusions

In this thesis, we have considered the optimization of nonsmooth but locally Lipschitz continuous objective functions with special emphasis on large-scale problems. The existing bundle methods are reliable and efficient methods for small- and medium-scale problems. However, in large-scale cases their computational demand expands. On the other hand, the subgradient methods suffer from some serious disadvantages and, thus, they are not necessarily the best choices for large-scale optimization. Furthermore, we have not been able to find any general solver for nonsmooth large-scale problems from literature. Thus, there is an evident need for a reliable and efficient solver for nonsmooth large-scale optimization problems.

In this thesis, we have described some existing methods for nonsmooth optimization and one method for large-scale smooth optimization. In addition, we have introduced a new limited memory variable metric bundle method for nonsmooth large-scale optimization. We have also tested the performance of these different optimization methods.

The new limited memory variable metric bundle method worked well for both the smooth and the convex nonsmooth optimization problems. In these cases, it was clearly the most efficient method tested when comparing the computational times and it also found the (local) minimum in a reliable way. With the smooth problems, the new limited memory variable metric bundle method `LVMBM` was almost twice as fast as the other limited memory variable metric method `L-BFGS` that has been developed for smooth large-scale minimization. In addition, for example for 1000 variables, the limited memory variable metric bundle method `LVMBM` was about 5 times faster than the fastest of the bundle methods `PBUN` and 250 times faster than the original variable metric bundle method `PVAR`. For nonsmooth problems, these differences were even more perceptible. For example, for one problem with 300 variables, the limited memory variable metric bundle method `LVMBM` was about 60 times faster than the original variable metric bundle program `PVAR`, 90 times faster than the proximal bundle program `PBUN`, 270 times faster than the other proximal bundle program `PBNCGC` and 1300 times faster than the bundle-Newton program `PNEW`.

With smooth problems the accuracy of the new method was comparable to the other methods tested. However, with nonsmooth problems the minimums found with the limited memory variable metric bundle method were usually slightly greater than those of the other methods. Thus, the accuracy of the method still needs some modifications so that it can better be compared with the other methods. The new method also had some difficulties to recognize a good solution in nonsmooth nonconvex case. This difficulty was most evident with the small maximum number of stored corrections. Thus, with nonsmooth and especially with nonconvex problems the maximum number of stored corrections should be at least 7.

In summary, we have introduced a new method for large-scale nonsmooth unconstrained optimization. Although the new method is quite useful already, there is a lot of further work required before the idea is complete. Possible areas of future development include the following:

- More nonsmooth numerical experiments.
- Implementation with better accuracy properties.
- Convergence analysis.
- Better implementation of bundle updating and line search.
- Alternative ways of scaling the updates (especially, the SR1 update).
- Constraint handling (simple bounds, linear constraints, nonlinear constraints).
- Parallelized version of the method.

A Computational Results in Tables

The results of the smooth problems 1 – 22 with 10, 100, 1000 and 10000 variables are collected in Tables 3 – 24. The parameters used are chosen experimentally (for details see Appendix C).

To see what kind of an effect the choice of parameter values had, the smooth problems 1 – 22 were also solved with the default parameters of the programs. The results are collected in Tables 25 – 30. The default parameter values used can be found in Appendix B and the experimentally chosen parameter values can be found in Appendix C. With the programs `PVAR` and `PNEW` the sizes of the bundle were set to $m_\xi = 2$ and $m_\xi = 10$, respectively, also with the default values. In these experiments, the number of variables was set to 500.

The results of the nonsmooth problems 23 – 26 with 100, 300, 1000, 3000 and 10000 variables are collected in Tables 31 – 44. The parameters used in these experiments can be found in Section 6.3.2.

In all the tables, n is the number of variables, m_ξ is the size of the bundle, No is the problem number, Ni denotes the number of iterations used, Nf denotes the number of objective function evaluations used, Ng denotes the number of subgradient evaluations used (in case of the programs `PVAR`, `PBUN`, `PBNCGC` and `LVMBM` this is the same as Nf), f is the value of the objective function at termination and Iterm is the cause of the termination (see Appendix D). In addition, with the program `L-BFGS` we denote the maximum number of stored corrections by m_c.

The CPU times (Time) are given in seconds. For the smooth problems the CPU times given are the total times used for the set of problems. For nonsmooth problems the CPU times are given individually.

Table 3: Numerical results of PVAR with 10 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	73	74	$8.1 \cdot 10^{-9}$	4	82	87	$1.4 \cdot 10^{-8}$	4
2	125	125	$4.5 \cdot 10^{-9}$	4	129	134	$1.5 \cdot 10^{-9}$	4
3	33	33	$1.8 \cdot 10^{-5}$	4	34	34	$6.8 \cdot 10^{-7}$	4
4	61	61	0.920932	4	101	146	0.920934	4
5	16	16	$9.2 \cdot 10^{-8}$	4	18	20	$6.1 \cdot 10^{-9}$	4
6	23	23	$5.3 \cdot 10^{-7}$	4	22	29	$2.3 \cdot 10^{-7}$	4
7	79	79	2.149359	4	47	47	3.019295	4
8	47	48	10.23278	4	51	53	10.23278	4
9	10	10	−133.5106	4	18	20	−133.5106	4
10	12	12	−139.0000	4	25	27	−139.0000	4
11	93	93	0.107766	4	125	130	0.107766	4
12	34	34	1.732180	4	52	61	1.732180	4
13	10	10	$9.1 \cdot 10^{-10}$	4	17	17	$4.4 \cdot 10^{-10}$	4
14	28	29	$2.0 \cdot 10^{-9}$	4	54	84	$4.4 \cdot 10^{-9}$	4
15	9	9	1.924609	4	9	9	1.924609	4
16	12	12	−8.051392	4	20	27	−8.051392	4
17	26	40	−0.038526	2	24	51	−0.038526	2
18	11	12	−0.025142	4	11	12	−0.025142	4
19	10	10	57.70502	4	10	10	57.70502	4
20	16	17	−1.012303	4	16	20	−1.012303	4
21	15	17	2.140725	4	16	20	2.140725	4
22	68	73	1.000000	4	51	62	1.000000	4
Time:			0.02				0.04	

Table 4: Numerical results of PNEW with 10 variables.

	$m_\xi = 10$					$m_\xi = 13$				
No	Ni	Nf	Ng	f	Iterm	Ni	Nf	Ng	f	Iterm
1	59	60	660	3.986579	4	64	65	715	3.986579	4
2	130	131	1441	$2.3 \cdot 10^{-25}$	4	130	131	1441	$2.3 \cdot 10^{-25}$	4
3	39	40	440	$1.3 \cdot 10^{-9}$	2	39	40	440	$1.3 \cdot 10^{-9}$	2
4	36	38	408	0.920932	4	43	45	485	0.920932	4
5	18	19	209	$2.6 \cdot 10^{-12}$	4	18	19	209	$2.6 \cdot 10^{-12}$	4
6	22	23	253	$7.5 \cdot 10^{-12}$	4	22	23	253	$7.5 \cdot 10^{-12}$	4
7	10	11	121	3.019295	4	10	11	121	3.019295	4
8	31	32	352	10.23278	4	31	32	352	10.23278	4
9	7	8	88	−133.5106	4	7	8	88	−133.5106	4
10	7	8	88	−139.0000	4	7	8	88	−139.0000	4
11	70	73	783	0.107766	4	67	70	750	0.107766	4
12	11	12	132	1.732180	4	11	12	132	1.732180	4
13	12	13	143	$6.7 \cdot 10^{-27}$	4	12	13	143	$6.7 \cdot 10^{-27}$	4
14	4	5	55	$5.2 \cdot 10^{-22}$	4	4	5	55	$5.2 \cdot 10^{-22}$	4
15	3	4	44	1.924609	4	3	4	44	1.924609	4
16	9	10	110	−8.051392	4	9	10	110	−8.051392	4
17	4	5	55	−0.038526	4	4	5	55	−0.038526	4
18	2	3	33	−0.025142	4	2	3	33	−0.025142	4
19	2	3	33	57.70502	4	2	3	33	57.70502	4
20	7	8	88	−1.012303	4	7	8	88	−1.012303	4
21	7	8	88	2.140725	4	7	8	88	2.140725	4
22	41	43	463	1.000000	4	41	43	463	1.000000	4
Time:				0.10					0.11	

Table 5: Numerical results of PBUN with 10 variables.

	$m_\xi = 10$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	427	434	$5.7 \cdot 10^{-7}$	2	253	257	$2.9 \cdot 10^{-7}$	2
2	188	189	$1.1 \cdot 10^{-5}$	4	268	269	$2.3 \cdot 10^{-7}$	2
3	132	133	$2.7 \cdot 10^{-6}$	4	60	61	$3.1 \cdot 10^{-5}$	4
4	46	47	0.920938	4	46	47	0.920938	4
5	25	27	$3.1 \cdot 10^{-11}$	4	25	27	$3.1 \cdot 10^{-11}$	4
6	21	22	$2.3 \cdot 10^{-7}$	4	21	22	$2.3 \cdot 10^{-7}$	4
7	17	18	3.019295	4	17	18	3.019295	4
8	341	346	10.23282	4	226	230	10.28055	4
9	13	14	−133.5106	4	13	14	−133.5106	4
10	26	27	−139.0000	4	26	27	−139.0000	4
11	231	233	0.107818	2	290	292	0.107796	2
12	84	85	1.732182	4	66	67	1.732186	4
13	12	13	$8.8 \cdot 10^{-16}$	4	12	13	$8.8 \cdot 10^{-16}$	4
14	95	101	$6.4 \cdot 10^{-5}$	4	116	123	$3.4 \cdot 10^{-5}$	4
15	40	41	1.924609	4	40	41	1.924609	4
16	16	17	−8.051392	4	16	17	−8.051392	4
17	26	27	−0.038526	4	26	27	−0.038526	4
18	25	27	−0.025142	4	25	27	−0.025142	4
19	16	17	57.70502	4	16	17	57.70502	4
20	21	23	−1.012303	4	21	23	−1.012303	4
21	29	30	2.140727	4	29	30	2.140727	4
22	33	38	1.000000	4	26	31	1.000000	4
Time:			0.10				0.09	

Table 6: Numerical results of PBNCGC with 10 variables.

	$m_\xi = 10$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	506	510	$4.4 \cdot 10^{-7}$	0	195	443	$7.4 \cdot 10^{-8}$	0
2	396	676	$7.1 \cdot 10^{-8}$	0	290	820	$1.8 \cdot 10^{-7}$	0
3	114	115	$1.5 \cdot 10^{-5}$	0	113	114	$7.6 \cdot 10^{-6}$	0
4	100	101	0.920938	0	100	101	0.920939	0
5	14	15	$1.9 \cdot 10^{-6}$	0	14	15	$1.9 \cdot 10^{-6}$	0
6	16	17	$5.1 \cdot 10^{-7}$	0	16	17	$5.1 \cdot 10^{-7}$	0
7	16	17	3.019295	0	16	17	3.019295	0
8	387	388	10.23278	0	283	284	10.23278	0
9	17	18	−133.5106	0	17	18	−133.5106	0
10	24	25	−139.0000	0	24	25	−139.0000	0
11	28	29	1.053642	0	28	29	1.053642	0
12	113	114	1.732180	0	92	93	1.732180	0
13	10	11	$2.3 \cdot 10^{-7}$	0	10	11	$2.3 \cdot 10^{-7}$	0
14	51	148	$2.3 \cdot 10^{-7}$	0	42	122	$1.8 \cdot 10^{-7}$	0
15	79	80	1.924609	0	85	86	1.924609	0
16	20	21	−8.051392	0	20	21	−8.051392	0
17	28	29	−0.038526	0	28	29	−0.038526	0
18	53	54	−0.025142	0	51	52	−0.025142	0
19	27	28	57.70502	0	27	28	57.70502	0
20	42	43	−1.012303	0	45	46	−1.012303	0
21	32	33	2.140725	0	32	33	2.140725	0
22	35	36	1.000000	0	25	26	1.000000	0
Time:			0.16				0.14	

Table 7: Numerical results of `LVMBM(7)` with 10 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	96	108	$1.1 \cdot 10^{-8}$	1	93	103	$5.6 \cdot 10^{-8}$	1
2	147	167	$2.7 \cdot 10^{-7}$	1	146	166	$2.7 \cdot 10^{-7}$	1
3	35	39	$7.2 \cdot 10^{-7}$	1	36	38	$9.6 \cdot 10^{-6}$	1
4	57	59	0.920933	1	55	58	0.920940	1
5	14	14	$1.1 \cdot 10^{-7}$	1	14	14	$1.1 \cdot 10^{-7}$	1
6	26	26	$5.9 \cdot 10^{-7}$	1	26	26	$5.9 \cdot 10^{-7}$	1
7	12	12	3.019295	1	12	12	3.019295	1
8	79	96	10.23278	1	73	87	10.23278	1
9	11	11	−133.5106	1	11	11	−133.5106	1
10	15	15	−139.0000	1	15	15	−139.0000	1
11	77	88	0.107766	1	76	86	0.107766	1
12	52	61	1.732180	1	47	56	1.732180	1
13	10	10	$7.2 \cdot 10^{-10}$	1	10	10	$7.2 \cdot 10^{-10}$	1
14	25	28	$6.6 \cdot 10^{-8}$	1	25	28	$6.6 \cdot 10^{-8}$	1
15	9	10	1.924609	1	9	10	1.924609	1
16	12	12	−8.051392	1	12	12	−8.051392	1
17	7	7	−0.038526	1	7	7	−0.038526	1
18	7	7	−0.025142	1	7	7	−0.025142	1
19	10	11	57.70502	1	10	11	57.70502	1
20	18	19	−1.012303	1	18	19	−1.012303	1
21	18	19	2.140725	1	18	19	2.140725	1
22	35	41	1.000000	1	45	50	1.000002	1
Time:			0.02				0.03	

Table 8: Numerical results of `LVMBM(3)` with 10 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	102	120	$7.8 \cdot 10^{-8}$	1	114	126	$1.7 \cdot 10^{-7}$	1
2	258	322	$2.7 \cdot 10^{-6}$	1	255	281	$2.3 \cdot 10^{-6}$	1
3	37	40	$2.1 \cdot 10^{-5}$	1	45	50	$1.1 \cdot 10^{-5}$	1
4	58	59	0.920933	1	58	59	0.920933	1
5	14	14	$1.1 \cdot 10^{-7}$	1	14	14	$1.1 \cdot 10^{-7}$	1
6	26	26	$5.9 \cdot 10^{-7}$	1	26	26	$5.9 \cdot 10^{-7}$	1
7	12	12	3.019295	1	12	12	3.019295	1
8	129	159	10.23278	1	111	125	10.23279	1
9	11	11	−133.5106	1	11	11	−133.5106	1
10	16	16	−139.0000	1	16	16	−139.0000	1
11	70	94	0.108245	1	59	69	0.107787	1
12	50	57	1.732180	1	57	73	1.732180	1
13	10	10	$8.5 \cdot 10^{-10}$	1	10	10	$8.5 \cdot 10^{-10}$	1
14	29	32	$6.2 \cdot 10^{-6}$	1	29	32	$6.2 \cdot 10^{-6}$	1
15	10	11	1.924609	1	10	11	1.924609	1
16	12	12	−8.051392	1	12	12	−8.051392	1
17	7	7	−0.038526	1	7	7	−0.038526	1
18	7	7	−0.025142	1	7	7	−0.025142	1
19	10	11	57.70502	1	10	11	57.70502	1
20	18	19	−1.012303	1	18	19	−1.012303	1
21	19	19	2.140725	1	19	19	2.140725	1
22	24	27	1.000000	1	27	30	1.000000	1
Time:			0.02				0.03	

Table 9: Numerical results of `L-BFGS` with 10 variables.

	$m_c = 3$				$m_c = 7$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	94	108	$4.7 \cdot 10^{-14}$	0	76	93	$5.4 \cdot 10^{-16}$	0
2	416	467	$8.3 \cdot 10^{-14}$	0	212	243	$1.5 \cdot 10^{-13}$	0
3	147	226	$2.7 \cdot 10^{-13}$	0	93	133	$5.5 \cdot 10^{-14}$	0
4	186	280	0.920932	0	64	85	0.920932	0
5	17	22	$1.2 \cdot 10^{-12}$	0	16	19	$1.5 \cdot 10^{-12}$	0
6	23	24	$1.1 \cdot 10^{-12}$	0	23	24	$2.0 \cdot 10^{-13}$	0
7	17	19	3.019295	0	13	14	3.019295	0
8	292	324	10.23278	0	138	155	10.23278	0
9	16	20	−133.5106	0	14	18	−133.5106	0
10	25	27	−139.0000	0	22	24	−139.0000	0
11	347	390	0.107766	0	35	37	1.053883	0
12	78	85	1.732179	0	47	56	1.732179	0
13	11	12	$1.7 \cdot 10^{-15}$	0	10	11	$1.4 \cdot 10^{-13}$	0
14	194	261	$9.0 \cdot 10^{-12}$	0	55	79	$6.2 \cdot 10^{-13}$	0
15	26	32	1.924609	0	13	16	1.924609	0
16	18	20	−8.051392	0	16	19	−8.051392	0
17	14	15	−0.038526	0	10	11	−0.038526	0
18	15	17	−0.025142	0	10	12	−0.025142	0
19	16	19	57.70502	0	12	14	57.70502	0
20	29	32	−1.012303	0	25	28	−1.012303	0
21	30	33	2.140725	0	24	29	2.140725	0
22	21	27	1.000000	0	23	27	1.000000	0
Time:			0.04				0.03	

Table 10: Numerical results of PVAR with 100 variables.

	$m_\xi = 2$				$m_\xi = 103$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	451	452	$1.9 \cdot 10^{-8}$	4	967	1082	$2.6 \cdot 10^{-7}$	4
2	518	518	$1.7 \cdot 10^{-5}$	4	572	596	$9.9 \cdot 10^{-8}$	4
3	58	60	$1.1 \cdot 10^{-6}$	4	624	768	$1.2 \cdot 10^{-6}$	4
4	161	162	25.20614	4	185	203	25.20613	4
5	19	19	$7.5 \cdot 10^{-7}$	4	68	112	$2.0 \cdot 10^{-7}$	2
6	40	40	$3.2 \cdot 10^{-6}$	4	447	641	$5.8 \cdot 10^{-8}$	4
7	614	614	29.52503	4	342	381	24.33194	4
8	34	34	1039.416	4	85	93	1039.416	4
9	11	11	−98.85603	4	11	11	−98.85603	4
10	370	517	−140.1000	4	235	250	−140.1000	4
11	323	363	1.077664	4	547	707	1.077659	2
12	229	229	82.29126	4	330	377	82.29126	4
13	13	15	$6.1 \cdot 10^{-10}$	4	15	17	$1.3 \cdot 10^{-13}$	4
14	3	3	$1.2 \cdot 10^{-6}$	4	3	3	$1.2 \cdot 10^{-6}$	4
15	129	129	1.924023	4	116	116	1.924023	4
16	39	39	−49.99978	4	123	147	−49.99978	4
17	29	30	−0.037998	4	45	57	−0.037999	4
18	49	50	−0.024580	4	94	94	−0.024581	4
19	115	115	59.40673	4	101	101	59.40673	4
20	117	117	−1.001340	4	359	441	−1.001340	4
21	75	75	2.138266	4	137	155	2.138263	4
22	1094	1097	1.000004	4	185	188	1.000000	4
Time:			2.32				4.88	

Table 11: Numerical results of PNEW with 100 variables.

	$m_\xi = 10$					$m_\xi = 103$				
No	Ni	Nf	Ng	f	Iterm	Ni	Nf	Ng	f	Iterm
1	438	439	44339	3.986624	4	438	439	44339	3.986624	4
2	227	233	23033	$1.6 \cdot 10^{-18}$	4	242	246	24546	$1.7 \cdot 10^{-23}$	4
3	36	37	3737	$1.1 \cdot 10^{-9}$	2	36	37	3737	$1.1 \cdot 10^{-9}$	2
4	38	42	3942	25.20613	4	38	42	3942	25.20613	4
5	19	20	2020	$2.8 \cdot 10^{-12}$	4	19	20	2020	$2.8 \cdot 10^{-12}$	4
6	26	27	2727	$3.0 \cdot 10^{-12}$	4	26	27	2727	$3.0 \cdot 10^{-12}$	4
7	10	11	1111	33.37543	4	10	11	1111	33.37543	4
8	15	16	1616	1039.416	4	15	16	1616	1039.416	4
9	9	10	1010	−98.85603	4	9	10	1010	−98.85603	4
10	68	88	6988	−133.1000	4	92	115	9415	−132.9000	4
11	53	54	5454	1.077659	4	69	70	7070	1.077659	4
12	11	12	1212	82.29126	4	11	12	1212	82.29126	4
13	13	14	1414	$8.1 \cdot 10^{-23}$	4	13	14	1414	$8.1 \cdot 10^{-23}$	4
14	4	5	505	$6.8 \cdot 10^{-25}$	4	4	5	505	$6.8 \cdot 10^{-25}$	4
15	4	5	505	1.924023	4	4	5	505	1.924023	4
16	10	11	1111	−49.99978	4	10	11	1111	−49.99978	4
17	4	5	505	−0.037999	4	4	5	505	−0.037999	4
18	2	3	303	−0.024581	4	2	3	303	−0.024581	4
19	2	3	303	59.40673	4	2	3	303	59.40673	4
20	8	9	909	−1.001340	4	8	9	909	−1.001340	4
21	9	10	1010	2.138263	4	9	10	1010	2.138263	4
22	129	140	13140	1.000000	4	148	157	15057	1.000000	4
Time:			14.98					37.62		

Table 12: Numerical results of PBUN with 100 variables.

	$m_\xi = 10$				$m_\xi = 103$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	1488	1489	$4.0 \cdot 10^{-5}$	4	1720	1722	$5.0 \cdot 10^{-5}$	4
2	1601	1602	$6.1 \cdot 10^{-7}$	2	1995	1999	$2.6 \cdot 10^{-8}$	2
3	164	165	$4.1 \cdot 10^{-6}$	2	75	76	$5.1 \cdot 10^{-5}$	4
4	65	67	25.20692	4	65	67	25.20692	4
5	22	23	$1.7 \cdot 10^{-7}$	4	22	23	$1.7 \cdot 10^{-7}$	4
6	23	24	$7.9 \cdot 10^{-8}$	4	23	24	$7.9 \cdot 10^{-8}$	4
7	24	27	33.37543	4	26	28	33.37543	4
8	45	46	1039.416	4	45	46	1039.416	4
9	11	12	−98.85603	4	11	12	−98.85603	4
10	163	164	−140.1000	4	166	167	−140.1000	4
11	358	362	1.078535	4	291	293	1.077677	4
12	221	222	82.29210	4	203	204	82.29136	4
13	11	12	$3.0 \cdot 10^{-12}$	4	11	12	$3.0 \cdot 10^{-12}$	4
14	21	26	$1.2 \cdot 10^{-6}$	2	21	26	$1.2 \cdot 10^{-6}$	2
15	277	280	1.924505	4	285	287	1.924444	4
16	56	57	−49.99978	4	51	52	−49.99978	4
17	323	325	−0.037998	2	183	185	−0.037995	4
18	170	173	−0.024576	4	177	180	−0.024558	4
19	232	234	59.40692	4	123	125	59.40683	4
20	254	256	−1.001321	4	199	201	−1.000899	4
21	188	190	2.139308	4	158	159	2.139909	4
22	20	35	17.97757	1	20	35	17.97757	1
Time:		0.93				4.10		

Table 13: Numerical results of PBNCGC with 100 variables.

	$m_\xi = 10$				$m_\xi = 103$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	884	889	$2.5 \cdot 10^{-6}$	0	2593	3365	$1.1 \cdot 10^{-6}$	0
2	1729	1731	$1.3 \cdot 10^{-6}$	0	1224	1225	$6.3 \cdot 10^{-7}$	0
3	162	163	$1.1 \cdot 10^{-5}$	0	152	153	$9.9 \cdot 10^{-6}$	0
4	188	189	25.20614	0	189	190	25.20614	0
5	15	16	$1.2 \cdot 10^{-6}$	0	15	16	$1.2 \cdot 10^{-6}$	0
6	20	21	$2.2 \cdot 10^{-6}$	0	21	22	$2.5 \cdot 10^{-7}$	0
7	19	20	33.37543	0	19	20	33.37543	0
8	79	80	1039.416	0	75	76	1039.416	0
9	13	14	−98.85603	0	13	14	−98.85603	0
10	340	341	−108.5000	0	313	314	−108.5000	0
11	197	438	1.077659	0	204	317	1.077659	0
12	442	443	82.29126	0	423	424	82.29126	0
13	15	16	$4.6 \cdot 10^{-8}$	0	15	16	$4.6 \cdot 10^{-8}$	0
14	248	1004	$1.0 \cdot 10^{-6}$	0	97	399	$1.2 \cdot 10^{-6}$	0
15	3851	3852	1.924023	0	735	736	1.924023	0
16	75	76	−49.99978	0	75	76	−49.99978	0
17	204	409	−0.037998	0	191	383	−0.037998	0
18	254	509	−0.024580	0	216	433	−0.024580	0
19	335	336	59.40673	0	342	343	59.40673	0
20	327	328	−1.001337	0	388	389	−1.001339	0
21	327	645	2.138266	0	305	599	2.138265	0
22	896	897	1.000001	0	590	591	1.000000	0
Time:		4.36				21.52		

Table 14: Numerical results of `LVMBM(7)` with 100 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	694	857	$9.8 \cdot 10^{-8}$	1	594	647	$2.3 \cdot 10^{-7}$	1
2	1127	1305	$9.4 \cdot 10^{-6}$	1	1197	1280	$5.4 \cdot 10^{-6}$	1
3	52	56	$8.2 \cdot 10^{-6}$	1	52	56	$8.3 \cdot 10^{-6}$	1
4	65	66	25.20613	1	73	75	25.20613	1
5	14	14	$9.5 \cdot 10^{-8}$	1	14	14	$9.5 \cdot 10^{-8}$	1
6	22	22	$6.5 \cdot 10^{-7}$	1	22	22	$6.5 \cdot 10^{-7}$	1
7	16	17	33.37543	1	16	17	33.37543	1
8	42	44	1039.416	1	42	44	1039.416	1
9	11	11	-98.85603	1	11	11	-98.85603	1
10	116	175	-140.1000	1	119	133	-139.7000	1
11	100	117	1.077787	1	97	105	1.077659	1
12	138	140	82.29126	1	131	132	82.29126	1
13	7	7	$4.5 \cdot 10^{-8}$	1	7	7	$4.5 \cdot 10^{-8}$	1
14	2	2	$1.2 \cdot 10^{-6}$	1	2	2	$1.2 \cdot 10^{-6}$	1
15	63	65	1.924024	1	65	67	1.924024	1
16	39	39	-49.99978	1	39	39	-49.99978	1
17	32	33	-0.037997	1	32	33	-0.037997	1
18	51	53	-0.024579	1	51	53	-0.024579	1
19	68	69	59.40673	1	68	69	59.40673	1
20	127	129	-1.001336	1	129	131	-1.001335	1
21	85	85	2.138267	1	85	85	2.138267	1
22	98	107	1.000000	1	101	108	1.000001	1
Time:			0.35				1.17	

Table 15: Numerical results of `LVMBM(3)` with 100 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	730	880	$3.4 \cdot 10^{-7}$	1	649	708	$4.1 \cdot 10^{-7}$	1
2	1120	1423	$5.3 \cdot 10^{-6}$	1	705	774	$9.8 \cdot 10^{-6}$	1
3	47	49	$1.2 \cdot 10^{-5}$	1	47	49	$1.3 \cdot 10^{-5}$	1
4	69	73	25.20613	1	59	61	25.20614	1
5	14	14	$9.5 \cdot 10^{-8}$	1	14	14	$9.5 \cdot 10^{-8}$	1
6	22	22	$6.5 \cdot 10^{-7}$	1	22	22	$6.5 \cdot 10^{-7}$	1
7	16	17	33.37543	1	16	17	33.37543	1
8	44	47	1039.416	1	43	45	1039.416	1
9	11	11	-98.85603	1	11	11	-98.85603	1
10	138	179	-118.3000	1	120	137	-125.5000	1
11	178	230	1.077674	1	255	291	1.077660	1
12	158	162	82.29126	1	144	150	82.29126	1
13	7	7	$4.5 \cdot 10^{-8}$	1	7	7	$4.5 \cdot 10^{-8}$	1
14	2	2	$1.2 \cdot 10^{-6}$	1	2	2	$1.2 \cdot 10^{-6}$	1
15	110	113	1.924026	1	87	90	1.924061	1
16	45	46	-49.99978	1	45	46	-49.99978	1
17	59	63	-0.037997	1	55	58	-0.037995	1
18	76	78	-0.024579	1	79	82	-0.024579	1
19	73	74	59.40674	1	73	74	59.40674	1
20	149	151	-1.001339	1	135	136	-1.001335	1
21	113	113	2.138269	1	113	113	2.138269	1
22	94	98	1.000002	1	100	104	1.000001	1
Time:			0.32				1.02	

Table 16: Numerical results of `L-BFGS` with 100 variables.

	$m_c = 3$				$m_c = 7$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	548	598	$5.1 \cdot 10^{-13}$	0	535	627	$2.3 \cdot 10^{-14}$	0
2	1045	1473	$4.8 \cdot 10^{-12}$	0	791	1151	$4.1 \cdot 10^{-12}$	0
3	197	214	$2.2 \cdot 10^{-12}$	0	150	163	$7.3 \cdot 10^{-12}$	0
4	222	233	25.20613	0	114	118	25.20613	0
5	22	23	$1.9 \cdot 10^{-12}$	0	21	22	$2.9 \cdot 10^{-12}$	0
6	35	37	$2.7 \cdot 10^{-12}$	0	35	37	$1.8 \cdot 10^{-12}$	0
7	27	31	33.37543	0	26	30	33.37543	0
8	65	70	1039.416	0	60	66	1039.416	0
9	13	16	-98.85603	0	13	16	-98.85603	0
10	352	375	-108.9000	0	251	258	-108.5000	0
11	354	392	1.077659	0	139	169	1.077659	0
12	238	255	82.29126	0	226	242	82.29126	0
13	11	12	$1.5 \cdot 10^{-14}$	0	11	12	$1.2 \cdot 10^{-14}$	0
14	6629	6942	$9.0 \cdot 10^{-9}$	0	2964	3038	$6.7 \cdot 10^{-10}$	0
15	–	–	–	$-1/3$	–	–	–	$-1/3$
16	74	80	-49.99978	0	68	73	-49.99978	0
17	156	166	-0.037999	0	121	127	-0.037999	0
18	262	273	-0.024581	0	159	166	-0.024581	0
19	214	228	59.40673	0	158	163	59.40673	0
20	309	326	-1.001340	0	240	246	-1.001340	0
21	310	320	2.138263	0	194	202	2.138263	0
22	249	257	1.000000	0	140	149	1.000000	0
Time:		0.93				0.65		

Table 17: Numerical results of PVAR with 1000 variables.

	$m_\xi = 2^*$				$m_\xi = 1003$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	14357	14374	$6.3 \cdot 10^{-9}$	4	5376	5425	$9.2 \cdot 10^{-8}$	4
2	2035	2036	53.58541	4	2812	2833	75.32483	4
3	99	101	$3.1 \cdot 10^{-5}$	4	2805	2884	$9.6 \cdot 10^{-7}$	4
4	8177	8667	269.5364	4	8460	8699	269.4996	4
5	18	18	$1.2 \cdot 10^{-6}$	4	28	40	$2.1 \cdot 10^{-7}$	4
6	308	308	$4.2 \cdot 10^{-5}$	4	261	282	$8.2 \cdot 10^{-8}$	4
7	5527	5724	188.7643	4	3235	3293	276.2918	4
8	116	121	761775.0	2	108	109	761775.0	2
9	28	28	316.4361	4	28	29	316.4361	4
10	4001	6137	−139.2500	4	2770	2925	−136.8500	4
11	2158	2354	10.77659	4	688	702	10.77659	4
12	1216	1249	982.2738	2	1073	1095	982.2736	2
13	22	23	$2.2 \cdot 10^{-11}$	4	17	20	$5.4 \cdot 10^{-12}$	4
14	3	3	$1.3 \cdot 10^{-9}$	4	3	3	$1.3 \cdot 10^{-9}$	4
15	1063	1063	1.924016	4	1026	1030	1.924016	4
16	229	242	−427.4045	2	1242	1268	−427.4045	4
17	341	531	−0.037992	4	506	708	−0.037992	4
18	454	907	−0.024574	4	454	907	−0.024574	4
19	1052	1070	59.59862	4	1034	1044	59.59862	4
20	4678	4687	−1.000135	4	3496	3525	−1.000135	4
21	676	1271	2.138670	4	1323	1324	2.138664	4
22	1162	1162	1.000121	4	1572	1574	1.000000	4
Time:		3886.62	(≈ 1.1h)			8437.67	(≈ 2.3h)	

* $m_\xi = 4$ in problems 1 and 2

Table 18: Numerical results of PNEW with 1000 variables.

	$m_\xi = 10$				
No	Ni	Nf	Ng	f	Iterm
1	4303	4305	4308305	3.986624	4
2	35	36	36036	3930.722	1
3	36	37	37037	$1.2 \cdot 10^{-9}$	2
4	57	58	58058	269.4995	4
5	21	22	22022	$1.1 \cdot 10^{-12}$	4
6	23	24	24024	$4.2 \cdot 10^{-12}$	4
7	10	11	11011	336.9372	4
8	8	15	9015	761775.0	4
9	26	30	27030	316.4361	4
10	139	140	140140	−82.93942	1
11	82	83	83083	10.77659	4
12	11	12	12012	982.2736	4
13	14	15	15015	$1.8 \cdot 10^{-42}$	4
14	4	5	5005	$1.2 \cdot 10^{-26}$	4
15	4	5	5005	1.924016	4
16	11	12	12012	−427.4045	4
17	4	5	5005	−0.037992	4
18	2	3	3003	−0.024574	4
19	2	6	3006	59.59862	4
20	11	12	12012	−1.000135	4
21	10	11	11011	2.138664	4
22	116	130	117130	11.83995	1
Time:			32192.21	(≈ 8.9h)	

Table 19: Numerical results of PBUN with 1000 variables.

	$m_\xi = 10$				$m_\xi = 1003$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	15293	15295	$1.9 \cdot 10^{-6}$	2	16059	16061	$4.5 \cdot 10^{-6}$	4
2	9655	9656	549.3333	4	9124	9125	269.2752	4
3	205	207	$2.6 \cdot 10^{-6}$	4	114	116	$6.0 \cdot 10^{-5}$	4
4	88	89	269.4998	4	88	89	269.4998	4
5	26	27	$1.7 \cdot 10^{-9}$	4	26	27	$1.7 \cdot 10^{-9}$	4
6	34	35	$2.0 \cdot 10^{-7}$	4	34	35	$2.0 \cdot 10^{-7}$	4
7	23	24	336.9372	4	23	24	336.9372	4
8	29	30	761775.0	2	29	30	761775.0	2
9	16	17	316.4361	4	16	17	316.4361	4
10	705	717	−133.3841	4	1005	1006	−130.6238	4
11	411	414	10.77798	4	405	408	10.77888	4
12	307	308	982.2737	4	310	311	982.2748	4
13	15	16	$1.2 \cdot 10^{-14}$	4	15	16	$1.2 \cdot 10^{-14}$	4
14	11	16	$1.3 \cdot 10^{-9}$	2	11	16	$1.3 \cdot 10^{-9}$	2
15	3607	3609	1.981668	4	4003	4007	1.983609	4
16	160	161	−427.4044	4	166	168	−427.4043	4
17	2728	2730	−0.033888	4	2383	2384	−0.036646	4
18	2494	2497	−0.024183	4	2434	2436	−0.023817	4
19	1850	1854	59.66069	4	2035	2038	59.65549	4
20	3054	3059	−0.977521	4	2388	2389	−0.989295	4
21	1333	1335	2.369362	4	1208	1210	2.309550	4
22	20	35	17.99977	1	20	35	17.99977	1
Time:		81.25	(≈ 1.4min)			6752.60	(≈ 1.9h)	

Table 20: Numerical results of PBNCGC with 1000 variables.

	$m_\xi = 10$				$m_\xi = 1003$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	4683	4699	$1.5 \cdot 10^{-6}$	0	8373	15216	$2.4 \cdot 10^{-7}$	0
2	2627	2628	$5.7 \cdot 10^{-7}$	0	4406	4407	7.147128	0
3	153	154	$8.7 \cdot 10^{-6}$	0	162	163	$6.7 \cdot 10^{-6}$	0
4	200	201	269.4995	0	168	169	269.4996	0
5	22	23	$2.0 \cdot 10^{-6}$	0	22	23	$2.0 \cdot 10^{-6}$	0
6	21	22	$8.8 \cdot 10^{-7}$	0	21	22	$8.8 \cdot 10^{-7}$	0
7	28	29	336.9372	0	30	31	336.9372	0
8	34	35	761775.0	0	34	35	761775.0	0
9	14	26	316.4361	0	14	26	316.4361	0
10	2737	2740	−130.6100	0	2318	2319	−130.6300	0
11	220	509	10.77659	0	255	676	10.77659	0
12	428	429	982.2736	0	385	386	982.2736	0
13	13	14	$4.1 \cdot 10^{-9}$	0	13	14	$4.1 \cdot 10^{-9}$	0
14	2	12	$1.3 \cdot 10^{-9}$	0	2	12	$1.3 \cdot 10^{-9}$	0
15	9769	19539	1.924021	0	5937	11875	1.924020	0
16	258	259	−427.4045	0	277	278	−427.4045	0
17	5929	11859	−0.037988	0	3494	6989	−0.037989	0
18	4529	13545	−0.024573	0	2442	7311	−0.024573	0
19	6362	6363	59.59865	0	5739	5746	59.59863	0
20	6081	6082	−1.000115	0	5475	5476	−1.000129	0
21	4469	13464	2.138666	0	5137	15394	2.138666	0
22	17891	18157	1.000010	0	13124	13125	1.000002	0
Time:		318.32	(≈ 5.3min)			36520.20	(≈ 10.1h)	

Table 21: Numerical results of `LVMBM(7)` with 1000 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	6605	8143	$2.7 \cdot 10^{-7}$	1	5865	6570	$3.0 \cdot 10^{-7}$	1
2	175	198	3.572791	1	214	227	7.147057	1
3	42	44	$1.8 \cdot 10^{-6}$	1	42	44	$1.8 \cdot 10^{-6}$	1
4	72	74	269.4996	1	71	72	269.4996	1
5	17	17	$6.8 \cdot 10^{-7}$	1	17	17	$6.8 \cdot 10^{-7}$	1
6	19	19	$3.5 \cdot 10^{-7}$	1	19	19	$3.5 \cdot 10^{-7}$	1
7	24	26	336.9372	1	23	24	336.9372	1
8	26	26	761775.0	1	26	26	761775.0	1
9	14	14	316.4361	1	14	14	316.4361	1
10	725	797	-134.6299	1	910	925	-128.3100	4
11	102	111	10.77659	1	88	92	10.77691	1
12	140	148	982.2736	4	142	150	982.2736	4
13	12	12	$5.4 \cdot 10^{-13}$	1	12	12	$5.4 \cdot 10^{-13}$	1
14	2	2	$1.3 \cdot 10^{-9}$	1	2	2	$1.3 \cdot 10^{-9}$	1
15	987	1010	1.924045	1	872	882	1.924095	1
16	119	121	-427.4045	4	124	126	-427.4045	1
17	327	332	-0.037964	1	491	497	-0.037961	1
18	458	468	-0.024564	1	433	438	-0.024540	1
19	646	663	59.59866	1	713	717	59.59873	1
20	970	989	-1.000104	1	1102	1110	-1.000116	1
21	992	1012	2.138693	1	837	849	2.138703	1
22	895	1015	1.000009	1	845	867	1.000037	1
Time:		15.38				892.16	($\approx$ 14.9min)	

Table 22: Numerical results of `LVMBM(3)` with 1000 variables.

	$m_\xi = 2$				$m_\xi = 13$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	6766	8152	$9.3 \cdot 10^{-8}$	1	6187	6722	$5.3 \cdot 10^{-8}$	1
2	228	270	3.572785	1	210	223	3.572777	1
3	52	56	$3.6 \cdot 10^{-6}$	1	53	58	$3.0 \cdot 10^{-6}$	1
4	68	73	269.4996	1	67	73	269.4996	1
5	16	16	$3.7 \cdot 10^{-7}$	1	16	16	$3.7 \cdot 10^{-7}$	1
6	19	19	$3.4 \cdot 10^{-7}$	1	19	19	$3.4 \cdot 10^{-7}$	1
7	23	24	336.9372	1	24	25	336.9372	1
8	25	26	761775.0	1	26	27	761775.0	1
9	14	14	316.4361	1	14	14	316.4361	1
10	784	815	-124.3699	4	922	931	-126.6500	1
11	101	119	10.77659	1	108	119	10.77665	1
12	155	164	982.2736	4	138	144	982.2736	4
13	12	12	$2.5 \cdot 10^{-15}$	1	12	12	$2.5 \cdot 10^{-15}$	1
14	2	2	$1.3 \cdot 10^{-9}$	1	2	2	$1.3 \cdot 10^{-9}$	1
15	1164	1193	1.924473	1	1433	1456	1.924616	1
16	128	130	-427.4045	4	128	133	-427.4045	1
17	348	350	-0.037953	1	392	395	-0.037970	1
18	716	741	-0.024534	1	423	428	-0.024448	1
19	801	824	59.59876	1	1045	1063	59.59877	1
20	1207	1234	-1.000052	1	1206	1221	-1.000076	1
21	1172	1190	2.138855	1	1083	1099	2.139081	1
22	932	1069	1.000018	1	1405	1426	1.000020	1
Time:		14.34				1078.94	($\approx$ 18.0min)	

Table 23: Numerical results of `L-BFGS` with 1000 variables.

	$m_c = 3$				$m_c = 7$			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	5036	5428	$1.2 \cdot 10^{-11}$	0	4998	5844	$1.3 \cdot 10^{-12}$	0
2	4754	5125	562.7389	0	3850	4085	228.9450	0
3	341	372	$4.9 \cdot 10^{-10}$	0	104	112	$7.5 \cdot 10^{-11}$	0
4	176	187	269.4995	0	107	113	269.4995	0
5	24	25	$2.9 \cdot 10^{-11}$	0	23	24	$3.2 \cdot 10^{-11}$	0
6	37	39	$1.0 \cdot 10^{-11}$	0	37	39	$1.2 \cdot 10^{-11}$	0
7	30	32	336.9372	0	30	32	336.9372	0
8	–	–	–	$-1/3$	–	–	–	$-1/3$
9	11	15	316.4361	0	11	15	316.4361	0
10	1923	2066	-134.4500	0	1238	1278	-134.4100	0
11	369	429	10.77659	0	116	149	199.0539	0
12	245	265	982.2736	0	230	246	982.2736	0
13	12	13	$1.0 \cdot 10^{-18}$	0	12	13	$1.7 \cdot 10^{-20}$	0
14	1	3	$1.3 \cdot 10^{-9}$	0	1	3	$1.3 \cdot 10^{-9}$	0
15	–	–	–	$-1/3$	–	–	–	$-1/3$
16	–	–	–	$-1/3$	–	–	–	$-1/3$
17	2725	2883	-0.037992	0	1180	1228	-0.037992	0
18	3935	4125	-0.024574	0	1934	1972	-0.024574	0
19	2715	2885	59.59862	0	1355	1395	59.59862	0
20	3948	4122	-1.000135	0	2304	2353	-1.000135	0
21	3679	3845	2.138664	0	2241	2282	2.138664	0
22	3814	3975	1.000000	0	1556	1626	1.000000	0
Time:		35.74				23.58		

Table 24: Numerical results of `LVMBM` and `L-BFGS` with 10000 variables.

	`LVMBM(7)`				`L-BFGS(7)`			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	40525	50000	3774.268	2	43682	50001	1265.081	–
2	39082	49990	583.3777	4	30483	32509	1520.488	0
3	57	61	$1.2 \cdot 10^{-5}$	1	136	143	$4.3 \cdot 10^{-14}$	0
4	120	122	2712.434	4	–	–	–	$-1/3$
5	38	38	$1.1 \cdot 10^{-7}$	1	25	33	$4.4 \cdot 10^{-10}$	0
6	25	25	$3.5 \cdot 10^{-7}$	1	36	39	$1.2 \cdot 10^{-10}$	0
7	35	40	3369.618	1	–	–	–	$-1/3$
8	23	24	$8.1 \cdot 10^{7}$	4	–	–	–	$-1/3$
9	49	49	3986.877	1	13	22	3986.877	0
10	3694	3827	-127.3240	1	17588	18004	-127.1890	0
11	99	129	107.7659	1	103	134	107.7659	0
12	122	133	9982.274	4	217	236	9982.274	0
13	15	15	$2.6 \cdot 10^{-8}$	1	13	14	$4.2 \cdot 10^{-15}$	0
14	39749	50000	$1.3 \cdot 10^{-12}$	2	1	3	$1.3 \cdot 10^{-12}$	0
15	5305	5415	1.936269	1	35765	50001	1.924252	–
16	307	315	-4159.932	4	–	–	–	$-1/3$
17	2604	2662	-0.037414	1	20568	21273	-0.037992	0
18	5311	5442	-0.024078	1	24373	25190	-0.024574	0
19	6129	6265	59.61931	1	20377	20806	59.61806	0
20	10366	10589	-0.999215	1	29826	31526	-1.000014	0
21	6836	7125	2.139914	1	17610	17907	2.138670	0
22	8694	9480	1.001152	1	–	–	–	$-1/3$
Time:		1707.23	($\approx$ 28.5min)			3119.81	($\approx$ 52.0min)	

Table 25: Numerical results of PVAR ($n = 500$ and $m_\xi = 2$).

	Default parameters				Selected parameters*			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	10000	10002	413.9563	12	6781	6800	$7.3 \cdot 10^{-8}$	4
2	10000	10002	252.0023	12	1216	1216	$1.2 \cdot 10^{-5}$	4
3	1846	1846	0.001133	4	79	81	$4.7 \cdot 10^{-7}$	4
4	18	21	272903.1	−10	437	438	133.7810	4
5	3158	3159	0.640665	4	19	19	$4.9 \cdot 10^{-7}$	4
6	10000	10002	32707.54	12	142	142	$1.9 \cdot 10^{-5}$	4
7	2321	2321	105.5881	4	2321	2321	105.5881	4
8	1001	1322	163899.9	2	128	132	163899.9	2
9	20	20	90.96722	4	19	19	90.96722	4
10	2039	3033	−140.7400	2	2021	3031	−144.8600	4
11	641	1169	256.8913	−10	981	1022	5.388299	4
12	3	3	2495.000	−10	1220	1244	482.2736	4
13	19	27	22.63183	−10	12	16	$7.5 \cdot 10^{-9}$	4
14	10000	10002	$9.5 \cdot 10^{-9}$	12	3	3	$1.0 \cdot 10^{-8}$	4
15	1390	1392	1.930750	−10	524	527	1.924016	4
16	1561	2152	−218.9106	2	112	128	−218.9106	4
17	170	212	−0.037992	4	170	212	−0.037992	4
18	1116	1149	−0.024574	4	228	454	−0.024574	4
19	610	614	59.57709	4	520	524	59.57709	4
20	10000	10572	16.65728	12	889	929	−1.000270	4
21	354	562	2.138652	4	396	475	2.138652	4
22	907	907	1.005574	4	657	657	1.000034	4
Time:		623.87	(≈ 10.4min)			189.30	(≈ 3.2min)	

* $m_\xi = 4$ in problems 1 and 2

Table 26: Numerical results of PNEW ($n = 500$ and $m_\xi = 10$).

	Default parameters					Selected parameters				
No	Ni	Nf	Ng	f	Iterm	Ni	Nf	Ng	f	Iterm
1	4430	4431	2219931	3.986624	4	2147	2148	1076148	3.986624	4
2	35	36	18036	1961.479	1	35	36	18036	1961.479	1
3	36	37	18537	$1.2 \cdot 10^{-9}$	2	36	37	18537	$1.2 \cdot 10^{-9}$	2
4	39	40	20040	133.7810	4	39	40	20040	133.7810	4
5	20	21	10521	$3.6 \cdot 10^{-12}$	4	20	21	10521	$3.6 \cdot 10^{-12}$	4
6	22	23	11523	$1.4 \cdot 10^{-11}$	4	22	23	11523	$1.4 \cdot 10^{-11}$	4
7	10	11	5511	168.2918	4	10	11	5511	168.2918	4
8	7	8	4008	163899.9	4	7	8	4008	163899.9	4
9	26	30	13530	90.96721	4	19	20	10020	90.96721	4
10	36	107	18607	−34.31313	1	152	163	76663	−135.9000	4
11	117	121	59121	5.388294	4	69	70	35070	5.388294	4
12	11	12	6012	482.2736	4	11	12	6012	482.2736	4
13	14	15	7515	$9.0 \cdot 10^{-43}$	4	14	15	7515	$9.0 \cdot 10^{-43}$	4
14	4	5	2505	$4.0 \cdot 10^{-26}$	4	4	5	2505	$4.0 \cdot 10^{-26}$	4
15	4	5	2505	1.924016	4	4	5	2505	1.924016	4
16	10	11	5511	−218.9106	4	10	11	5511	−218.9109	4
17	4	5	2505	−0.037992	4	4	5	2505	−0.037992	4
18	2	5	1505	−0.024574	4	2	5	1505	−0.024574	4
19	2	4	1504	59.57709	4	2	4	1504	59.57709	4
20	14	19	7519	−1.000270	4	9	11	5011	−1.000270	4
21	10	11	5511	2.138647	4	10	11	5511	2.138647	4
22	29	40	15040	17.51285	1	29	40	15040	17.51285	1
Time:			2819.27	(≈ 47.0min)				1715.84	(≈ 28.6min)	

Table 27: Numerical results of PBUN ($n = 500$ and $m_\xi = 503$).

	Default parameters				Selected parameters			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	8156	8161	$8.7 \cdot 10^{-6}$	2	8198	8199	$6.3 \cdot 10^{-7}$	2
2	20	21	795953.1	1	7766	7769	66.48174	4
3	20	21	127935.0	1	76	77	$2.0 \cdot 10^{-5}$	4
4	–	–	–	*	72	73	133.7815	4
5	27	29	$5.1 \cdot 10^{-10}$	2	28	29	$4.5 \cdot 10^{-10}$	4
6	20	21	32708.17	1	29	30	$8.0 \cdot 10^{-9}$	4
7	25	27	168.2918	2	20	21	168.2918	4
8	20	24	202960.6	1	24	25	163899.9	4
9	13	15	90.96721	4	13	15	90.96721	4
10	433	441	−128.8583	4	510	527	−136.6595	4
11	4	6	54336.85	−10	401	402	5.388375	4
12	3	5	734.3398	−10	258	260	482.2739	4
13	4	5	998.0000	−10	12	13	$2.8 \cdot 10^{-12}$	4
14	1	3	$1.0 \cdot 10^{-8}$	4	1	3	$1.0 \cdot 10^{-8}$	4
15	1606	1611	1.965370	4	2183	2186	1.930779	4
16	20	24	57997.03	1	118	121	−218.9104	4
17	1625	1626	−0.037266	4	1107	1108	−0.037757	4
18	894	896	−0.015347	4	1168	1170	−0.024315	4
19	1090	1094	59.60100	4	809	812	59.59290	4
20	1031	1034	−0.944770	4	1219	1224	−0.997502	4
21	1124	1128	2.226110	4	856	857	2.165969	4
22	20	35	17.99907	1	20	35	17.99907	1
Time:		–			344.58	(≈ 5.7min)		

* Program jammed.

Table 28: Numerical results of PBNCGC ($n = 500$ and $m_\xi = 503$).

	Default parameters				Selected parameters			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	–	–	–	1	4619	9185	$1.6 \cdot 10^{-7}$	0
2	6306	6307	7.147128	0	3507	3508	$3.7 \cdot 10^{-7}$	0
3	164	165	$7.6 \cdot 10^{-6}$	0	164	165	$7.6 \cdot 10^{-6}$	0
4	181	184	133.7810	0	115	118	133.7810	0
5	22	23	$1.8 \cdot 10^{-7}$	0	22	23	$1.8 \cdot 10^{-7}$	0
6	20	21	$2.9 \cdot 10^{-6}$	0	20	21	$2.9 \cdot 10^{-6}$	0
7	24	25	168.2918	0	24	25	168.2918	0
8	29	30	163899.9	0	29	30	163899.9	0
9	14	32	90.96721	0	13	23	90.96721	0
10	1117	1118	−133.0600	0	993	994	−135.2600	0
11	222	376	5.388294	0	198	360	5.388294	0
12	462	463	482.2736	0	421	422	482.2736	0
13	17	18	$1.9 \cdot 10^{-7}$	0	13	14	$4.8 \cdot 10^{-7}$	0
14	5	51	$1.0 \cdot 10^{-8}$	0	5	26	$1.0 \cdot 10^{-8}$	0
15	4771	4837	1.924021	0	2540	5081	1.924019	0
16	196	197	−218.9106	0	178	179	−218.9106	0
17	1136	3409	−0.037992	0	1226	2453	−0.037990	0
18	2189	6568	−0.024574	0	1119	2239	−0.024573	0
19	2360	2361	59.57709	0	2214	2215	59.57709	0
20	2319	2320	−1.000268	0	2144	2145	−1.000265	0
21	2436	7283	2.138648	0	1668	3329	2.138654	0
22	6261	6262	1.000001	0	5682	5683	1.000001	0
Time:		3781.70	(≈ 1.1h)			2289.42	(≈ 38.2min)	

Table 29: Numerical results of `LVMBM` ($n = 500$, $m_\xi = 2$ and $m_c = 7$).

	Default parameters				Selected parameters*			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	3270	3971	$1.2 \cdot 10^{-7}$	1	3270	3971	$1.2 \cdot 10^{-7}$	1
2	2552	3151	7.147143	1	2592	3270	$1.1 \cdot 10^{-5}$	1
3	67	73	$4.1 \cdot 10^{-7}$	1	42	43	$1.2 \cdot 10^{-6}$	1
4	80	81	133.7810	1	69	74	133.7810	1
5	18	18	$2.1 \cdot 10^{-7}$	1	15	15	$1.4 \cdot 10^{-7}$	1
6	26	26	$1.1 \cdot 10^{-6}$	1	21	21	$4.2 \cdot 10^{-7}$	1
7	20	20	168.2918	1	18	19	168.2918	1
8	22	23	163899.9	1	20	20	163899.9	1
9	26	26	90.96722	1	12	12	90.96722	1
10	443	455	−128.3400	1	397	438	−134.1400	1
11	114	131	5.388506	1	102	115	5.388294	1
12	160	167	482.2736	4	142	149	482.2736	4
13	16	16	$1.9 \cdot 10^{-10}$	1	10	10	$2.1 \cdot 10^{-10}$	1
14	2	2	$1.0 \cdot 10^{-8}$	1	2	2	$1.0 \cdot 10^{-8}$	1
15	443	458	1.924043	1	442	455	1.924050	1
16	92	92	−218.9106	1	86	88	−218.9106	1
17	195	197	−0.037986	1	195	197	−0.037986	1
18	237	242	−0.024558	1	237	242	−0.024558	1
19	477	481	59.57710	1	473	477	59.57709	1
20	604	608	−1.000263	1	604	608	−1.000263	1
21	340	349	2.138724	1	462	469	2.138670	1
22	418	447	1.000005	1	418	447	1.000005	1
Time:		5.04				5.02		

Table 30: Numerical results of `L-BFGS` ($n = 500$ and $m_c = 5$).

	Default parameters				Selected parameters			
No	Ni	Nf	f	Iterm	Ni	Nf	f	Iterm
1	2514	2849	$4.2 \cdot 10^{-13}$	0	2523	2884	$1.5 \cdot 10^{-12}$	0
2	2566	2779	158.6535	0	2527	2707	81.29840	0
3	164	181	$3.2 \cdot 10^{-10}$	0	164	181	$3.2 \cdot 10^{-10}$	0
4	164	172	133.7810	0	142	152	133.7810	0
5	22	23	$5.5 \cdot 10^{-12}$	0	22	23	$5.5 \cdot 10^{-12}$	0
6	36	39	$5.0 \cdot 10^{-12}$	0	36	39	$5.0 \cdot 10^{-12}$	0
7	28	30	168.2918	0	28	30	168.2918	0
8	–	–	–	−1/3	33	44	163899.9	0
9	11	15	90.96721	0	11	15	90.96721	0
10	693	711	−131.1000	0	692	709	−131.1000	0
11	213	259	5.388294	0	225	270	5.388294	0
12	243	266	482.2736	0	240	263	482.2736	0
13	12	13	$5.7 \cdot 10^{-15}$	0	12	13	$5.7 \cdot 10^{-15}$	0
14	92	94	$9.9 \cdot 10^{-9}$	0	92	94	$9.9 \cdot 10^{-9}$	0
15	–	–	–	−1/3	–	–	–	−1/3
16	–	–	–	−1/3	–	–	–	−1/3
17	817	845	−0.037992	0	674	696	−0.037992	0
18	1061	1090	−0.024574	0	913	944	−0.024574	0
19	823	855	59.57709	0	848	875	59.57709	0
20	1238	1281	−1.000270	0	1161	1201	−1.000270	0
21	1168	1200	2.138647	0	1097	1114	2.138647	0
22	723	741	1.000000	0	693	713	1.000000	0
Time:		6.29				6.19		

Table 31: Numerical results for problem 23 for $n = 100$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	384	384	384	0.597233	4	0.19
PVAR	50	293	293	293	0.597233	4	0.20
PVAR	100	289	289	289	0.597229	4	0.22
PNEW	10	588	711	59611	0.597325	1	6.14
PNEW	50	860	950	87050	0.597226	4	24.96
PNEW	100	366	391	37091	0.597226	4	18.97
PBUN	10	6869	7104	7104	0.597343	4	2.15
PBUN	50	2970	3002	3002	0.597322	4	9.08
PBUN	100	2799	2822	2822	0.597280	4	20.79
PBNCGC	10	6207	6208	6208	0.597233	0	3.45
PBNCGC	50	547	548	548	0.597235	0	2.63
PBNCGC	100	369	370	370	0.597235	0	4.64
LVMBM(3)	2	78	134	134	0.603491	1	0.01
LVMBM(3)	50	143	152	152	0.600164	1	0.03
LVMBM(3)	100	143	152	152	0.600164	1	0.04
LVMBM(7)	2	201	390	390	0.598741	1	0.02
LVMBM(7)	50	162	193	193	0.600360	1	0.04
LVMBM(7)	100	254	289	289	0.598478	1	0.10
LVMBM(15)	2	325	814	814	0.598306	1	0.06
LVMBM(15)	50	288	369	369	0.598136	1	0.10
LVMBM(15)	100	282	356	356	0.598138	1	0.14

Table 32: Numerical results for problem 23 for $n = 300$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	1235	1235	1235	2.251760	4	4.94
PVAR	50	928	928	928	2.251706	4	5.01
PVAR	100	918	918	918	2.251699	4	4.54
PNEW	10	654	737	197237	2.253521	1	103.54
PNEW	50	975	1081	293881	2.251711	1	536.12
PNEW	100	881	999	265599	2.251645	1	1125.10
PBUN	10	7381	7906	7906	2.255322	4	6.94
PBUN	50	11002	12136	12136	2.255831	4	76.04
PBUN	100	1776	1817	1817	2.257157	4	37.53
PBNCGC	10	11429	11430	11430	2.251683	0	21.55
PBNCGC	50	2234	2235	2235	2.251691	0	38.93
PBNCGC	100	1597	1598	1598	2.251695	0	77.92
LVMBM(3)	2	272	296	296	2.283775	1	0.04
LVMBM(3)	50	311	316	316	2.290962	1	0.22
LVMBM(3)	100	311	316	316	2.290962	1	0.35
LVMBM(7)	2	337	395	395	2.268969	1	0.08
LVMBM(7)	50	449	462	462	2.273679	1	0.35
LVMBM(7)	100	474	495	495	2.265205	1	0.76
LVMBM(15)	2	384	561	561	2.264918	1	0.14
LVMBM(15)	50	364	398	398	2.267038	1	0.32
LVMBM(15)	100	374	405	405	2.266850	1	0.50

Table 33: Numerical results for problem 23 for $n = 1000$.

Program	m_ξ	Ni	Nf	f	Iterm	Time
PVAR	2	3489	3489	8.019038	4	306.01
PVAR	50	3191	3191	7.973548	4	288.78
PVAR	100	3156	3156	7.973310	4	314.99
PBUN	10	20000	22406	8.020267	12	60.43
PBUN	50	20000	22946	8.021438	12	345.12
PBUN	100	20000	22875	8.013328	12	853.66
PBNCGC	10	20000	20001	7.957525	2	132.60
PBNCGC	50	20000	20001	7.957104	2	1397.29
PBNCGC	100	17383	17384	7.957054	0	3161.41
LVMBM(3)	2	870	873	8.119211	1	0.50
LVMBM(3)	50	1433	1437	8.063846	1	3.86
LVMBM(3)	100	1636	1645	8.058992	1	7.56
LVMBM(7)	2	1499	1566	8.064696	1	1.21
LVMBM(7)	50	1356	1362	8.071977	1	3.75
LVMBM(7)	100	1371	1377	8.071638	1	7.16
LVMBM(15)	2	1322	1489	8.075832	1	1.44
LVMBM(15)	50	1788	1806	8.039140	1	5.53
LVMBM(15)	100	1520	1538	8.056927	1	8.67

Table 34: Numerical results for problem 23 for $n = 3000$.

Program	m_ξ	Ni	Nf	f	Iterm	Time
LVMBM(3)	2	2939	2940	25.60902	1	5.42
LVMBM(3)	50	3933	3933	25.52001	1	34.52
LVMBM(3)	100	3933	3933	25.52001	1	64.08
LVMBM(7)	2	3940	3973	25.46932	1	9.48
LVMBM(7)	50	2036	2037	25.77998	1	19.08
LVMBM(7)	100	2036	2037	25.77998	1	34.04
LVMBM(15)	2	3844	3878	25.48648	1	13.49
LVMBM(15)	50	4021	4030	25.44966	1	41.58
LVMBM(15)	100	4021	4030	25.44966	1	85.24

Table 35: Numerical results for problem 23 for $n = 10000$.

Program	m_ξ	Ni	Nf	f	Iterm	Time
LVMBM(3)	2	14742	14743	86.83975	1	95.74
LVMBM(3)	50	13454	13454	87.00658	1	528.43
LVMBM(7)	2	12069	12070	86.99000	1	108.01
LVMBM(7)	50	13235	13235	86.93343	1	641.56
LVMBM(15)	2	11984	11994	86.98348	1	162.04
LVMBM(15)	50	11628	11630	87.01912	1	685.39

Table 36: Numerical results for problem 24 for $n = 100$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	370	370	370	0.797506	4	0.18
PVAR	50	253	253	253	0.797510	4	0.16
PVAR	100	266	266	266	0.797504	4	0.21
PNEW	10	638	736	64638	0.797516	1	6.93
PNEW	50	853	972	86372	0.797501	4	25.97
PNEW	100	182	189	18489	0.797501	4	7.67
PBUN	10	6472	6609	6609	0.797581	4	2.08
PBUN	50	2859	2865	2865	0.797577	4	10.01
PBUN	100	2620	2630	2630	0.797542	4	20.55
PBNCGC	10	5900	5901	5901	0.797509	0	3.29
PBNCGC	50	467	468	468	0.797510	0	2.27
PBNCGC	100	300	301	301	0.797509	0	3.70
LVMBM(3)	2	136	214	214	0.805387	1	0.01
LVMBM(3)	50	140	147	147	0.800699	1	0.03
LVMBM(3)	100	145	154	154	0.800702	1	0.04
LVMBM(7)	2	266	584	584	0.798614	1	0.03
LVMBM(7)	50	220	265	265	0.798696	1	0.07
LVMBM(7)	100	240	281	281	0.798596	1	0.10
LVMBM(15)	2	157	341	341	0.799959	1	0.03
LVMBM(15)	50	222	281	281	0.798410	1	0.08
LVMBM(15)	100	238	301	301	0.798451	1	0.11

Table 37: Numerical results for problem 24 for $n = 300$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	1143	1143	1143	2.758809	4	4.40
PVAR	50	886	886	886	2.758071	4	3.90
PVAR	100	901	901	901	2.758055	4	4.44
PNEW	10	541	619	163219	2.767790	1	81.84
PNEW	50	1228	1380	370080	2.758019	1	554.50
PNEW	100	1215	1332	366132	2.757974	4	958.10
PBUN	10	11112	11966	11966	2.761180	4	10.08
PBUN	50	4344	4575	4575	2.765381	4	31.81
PBUN	100	4045	4174	4174	2.760637	4	85.15
PBNCGC	10	10773	10774	10774	2.758036	0	19.82
PBNCGC	50	2466	2467	2467	2.758053	0	40.77
PBNCGC	100	1388	1389	1389	2.758047	0	55.07
LVMBM(3)	2	287	320	320	2.816503	1	0.05
LVMBM(3)	50	294	299	299	2.789543	1	0.21
LVMBM(3)	100	294	299	299	2.789543	1	0.42
LVMBM(7)	2	392	503	503	2.774816	1	0.09
LVMBM(7)	50	378	391	391	2.775504	1	0.34
LVMBM(7)	100	378	391	391	2.775504	1	0.48
LVMBM(15)	2	399	576	576	2.775665	1	0.15
LVMBM(15)	50	420	490	490	2.775337	1	0.40
LVMBM(15)	100	401	450	450	2.775795	1	0.56

Table 38: Numerical results for problem 24 for $n = 1000$.

Program	m_ξ	Ni	Nf	f	Iterm	Time
PVAR	2	3281	3281	9.863656	4	285.28
PVAR	50	3120	3120	9.780578	4	276.41
PVAR	100	2932	2932	9.802215	4	264.10
PBUN	10	20000	22376	9.868510	12	60.28
PBUN	50	20000	22782	9.875106	12	347.34
PBUN	100	20000	22653	9.883567	12	800.27
PBNCGC	10	20000	20001	9.762187	2	133.75
PBNCGC	50	20000	20001	9.761775	2	1424.21
PBNCGC	100	16777	16778	9.761733	0	3270.80
LVMBM(3)	2	850	864	10.01449	1	0.49
LVMBM(3)	50	1094	1098	9.976292	1	2.99
LVMBM(3)	100	1094	1098	9.976292	1	5.28
LVMBM(7)	2	1685	1768	9.889116	1	1.26
LVMBM(7)	50	1610	1635	9.904412	1	4.43
LVMBM(7)	100	2050	2059	9.877420	1	9.87
LVMBM(15)	2	1663	1828	9.886469	1	1.78
LVMBM(15)	50	1655	1685	9.880411	1	5.20
LVMBM(15)	100	1711	1742	9.877058	1	8.76

Table 39: Numerical results for problem 24 for $n = 3000$.

Program	m_ξ	Ni	Nf	f	Iterm	Time
LVMBM(3)	2	2577	2586	32.12470	1	4.76
LVMBM(3)	50	2271	2277	32.17679	1	20.16
LVMBM(3)	100	2280	2284	32.17654	1	36.92
LVMBM(7)	2	3962	3963	31.77740	1	9.48
LVMBM(7)	50	5072	5073	31.64788	1	47.57
LVMBM(7)	100	5072	5073	31.64788	1	88.38
LVMBM(15)	2	5879	5933	31.57238	1	20.48
LVMBM(15)	50	4032	4044	31.75792	1	42.56
LVMBM(15)	100	4032	4044	31.75792	1	74.67

Table 40: Numerical results for problem 24 for $n = 10000$.

Program	m_ξ	Ni	Nf	f	Iterm	Time
LVMBM(3)	2	19248	19249	108.3458	1	129.79
LVMBM(3)	50	17929	17929	108.6977	1	745.69
LVMBM(7)	2	20361	20399	108.1726	1	180.70
LVMBM(7)	50	23229	23229	107.9648	1	1124.01
LVMBM(15)	2	20935	21012	108.1005	1	270.24
LVMBM(15)	50	22932	22940	107.8543	1	1345.16

Table 41: Numerical results for problem 25 for $n = 100$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	849	978	978	$2.73 \cdot 10^{-5}$	4	0.66
PVAR	50	649	649	649	$2.29 \cdot 10^{-5}$	4	0.64
PVAR	100	607	624	624	$3.19 \cdot 10^{-5}$	4	0.74
PNEW	10	933	1032	94432	$1.12 \cdot 10^{-3}$	1	36.11
PNEW	50	927	1084	93884	$3.06 \cdot 10^{-4}$	2	56.68
PNEW	100	447	539	45339	$4.46 \cdot 10^{-4}$	2	41.25
PBUN	10	16884	18565	18565	$1.98 \cdot 10^{-2}$	1	10.65
PBUN	50	8087	8515	8515	$1.51 \cdot 10^{-2}$	2	30.83
PBUN	100	4938	5072	5072	$3.72 \cdot 10^{-2}$	4	45.14
PBNCGC	10	20000	20038	20038	$2.19 \cdot 10^{-2}$	2	18.43
PBNCGC	50	20000	20039	20039	$2.16 \cdot 10^{-3}$	2	218.15
PBNCGC	100	854	891	891	$7.85 \cdot 10^{-6}$	0	26.30
LVMBM(3)	2	286	584	584	4.241187	1	0.19
LVMBM(3)	50	322	335	335	0.812795	1	0.18
LVMBM(3)	100	322	335	335	0.812795	1	0.22
LVMBM(7)	2	609	1270	1270	$3.14 \cdot 10^{-2}$	1	0.42
LVMBM(7)	50	344	383	383	0.144926	1	0.21
LVMBM(7)	100	531	594	594	0.147723	1	0.44
LVMBM(15)	2	506	1217	1217	$2.79 \cdot 10^{-4}$	1	0.44
LVMBM(15)	50	952	1216	1216	$1.24 \cdot 10^{-2}$	1	0.70
LVMBM(15)	100	901	1132	1132	$2.17 \cdot 10^{-4}$	1	0.84

Table 42: Numerical results for problem 25 for $n = 300$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	1837	1837	1837	1.843808	4	12.69
PVAR	50	1810	1810	1810	$2.83 \cdot 10^{-4}$	4	12.35
PVAR	100	1798	1800	1800	$4.55 \cdot 10^{-4}$	4	13.26
PNEW	10	572	586	172486	242.6153	1	519.75
PNEW	50	2000	2026	602326	262.4942	12	2519.98
PNEW	100	463	464	139664	829.7253	1	860.69
PBUN	10	20000	22310	22310	$4.23 \cdot 10^{-2}$	12	75.71
PBUN	50	14278	15327	15327	$2.88 \cdot 10^{-2}$	1	181.91
PBUN	100	6572	7053	7053	$5.63 \cdot 10^{-2}$	2	195.79
PBNCGC	10	20000	21800	21800	0.129068	2	94.24
PBNCGC	50	20000	22283	22283	$1.61 \cdot 10^{-2}$	2	671.56
PBNCGC	100	20000	22798	22798	$6.11 \cdot 10^{-3}$	2	2396.59
LVMBM(3)	2	417	546	546	4.192283	1	1.45
LVMBM(3)	50	208	240	240	284.4290	1	0.76
LVMBM(3)	100	209	244	244	284.4290	1	0.82
LVMBM(7)	2	1215	1735	1735	0.122461	1	4.67
LVMBM(7)	50	817	884	884	3.219914	1	2.89
LVMBM(7)	100	922	978	978	3.126862	1	3.73
LVMBM(15)	2	717	1334	1334	$3.41 \cdot 10^{-2}$	1	3.64
LVMBM(15)	50	823	930	930	0.561701	1	3.11
LVMBM(15)	100	962	1068	1068	0.650828	1	4.12

Table 43: Numerical results for problem 26 for $n = 100$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	416	832	832	$1.36 \cdot 10^{-5}$	4	0.20
PVAR	50	398	763	763	$9.51 \cdot 10^{-6}$	4	0.28
PVAR	100	415	746	746	$1.09 \cdot 10^{-5}$	4	0.38
PNEW	10	59	91	5991	0.774453	−10	0.56
PNEW	50	56	94	5694	0.772433	−10	0.82
PNEW	100	56	94	5694	0.772433	−10	0.78
PBUN	10	11	21	21	0.746257	2	0.003
PBUN	50	11	21	21	0.746257	2	0.003
PBUN	100	11	21	21	0.746257	2	0.003
PBNCGC	10	1782	2054	2054	$9.40 \cdot 10^{-6}$	0	0.54
PBNCGC	50	1279	1329	1329	$1.00 \cdot 10^{-5}$	0	1.93
PBNCGC	100	1238	1358	1358	$9.85 \cdot 10^{-6}$	0	4.29
LVMBM(3)	2	9981	50001	50001	0.746257	2	1.67
LVMBM(3)	50	30	104	104	0.746257	4	0.01
LVMBM(3)	100	30	104	104	0.746257	4	0.01
LVMBM(7)	2	46	350	350	0.746257	4	0.01
LVMBM(7)	50	26	176	176	0.746257	4	0.01
LVMBM(7)	100	26	176	176	0.746257	4	0.01
LVMBM(15)	2	237	940	940	0.731147	1	0.07
LVMBM(15)	50	42	350	350	0.746257	4	0.02
LVMBM(15)	100	42	350	350	0.746257	4	0.02

Table 44: Numerical results for problem 26 for $n = 300$.

Program	m_ξ	Ni	Nf	Ng	f	Iterm	Time
PVAR	2	20000	1384	1384	$3.50 \cdot 10^{-3}$	12	21.86
PVAR	50	756	1865	1865	$1.15 \cdot 10^{-4}$	4	3.63
PVAR	100	669	1443	1443	$2.53 \cdot 10^{-4}$	4	3.33
PNEW	10	246	461	74261	0.764749	−10	38.19
PNEW	50	294	489	88989	0.771073	4	104.37
PNEW	100	280	468	84768	0.771481	4	149.51
PBUN	10	16	20	20	0.742528	2	0.01
PBUN	50	16	20	20	0.742528	2	0.01
PBUN	100	16	20	20	0.742528	2	0.01
PBNCGC	10	2810	6774	6774	$9.98 \cdot 10^{-5}$	0	3.38
PBNCGC	50	1781	4835	4835	$9.86 \cdot 10^{-5}$	0	16.65
PBNCGC	100	1517	3622	3622	$9.03 \cdot 10^{-5}$	0	28.35
LVMBM(3)	2	9994	50000	50000	0.743683	2	4.88
LVMBM(3)	50	30	86	86	0.743344	4	0.01
LVMBM(3)	100	30	86	86	0.743344	4	0.01
LVMBM(7)	2	52	352	352	0.743683	4	0.04
LVMBM(7)	50	41	135	135	0.743344	4	0.03
LVMBM(7)	100	41	135	135	0.743344	4	0.03
LVMBM(15)	2	65	620	620	0.743683	4	0.07
LVMBM(15)	50	35	206	206	0.743344	4	0.04
LVMBM(15)	100	35	206	206	0.743344	4	0.03

B Default Parameters

Table 45: Default parameters of program PVAR.

Param.	Value	Significance
MEX	1	Variable specifying the version of variable metric bundle method. MEX= 0: convex version is used, MEX= 1: nonconvex version of method.
MOS	1	Distance measure exponent ω.
MTESX	20	Maximum number of iterations with changes of the vector $\mathbf{x}$ smaller than TOLX.
MTESF	2	Maximum number of iterations with changes of function values smaller than TOLF.
MIT	10000	Maximum number of iterations.
MFV	20000	Maximum number of function calls.
TOLX	$1.0 \cdot 10^{-16}$	Tolerance for the change of the vector $\mathbf{x}$.
TOLF	$1.0 \cdot 10^{-8}$	Tolerance for the change of the function value.
TOLB	$-1.0 \cdot 10^{-60}$	Minimum acceptable function value.
TOLG	$1.0 \cdot 10^{-6}$	Final accuracy parameter ε.
ETA	0.250	Distance measure parameter γ.
XMAX	1000.0	Maximum step size.
NA	$n+3$	Maximum number of stored subgradients m_ξ.

Table 46: Default parameters of program PNEW.

Param.	Value	Significance
MOS	1	Distance measure exponent ω.
MES	2	Variable specifying the interpolation method selection in a line search. MES= 1: bisection, MES= 2: two-point quadratic interpolation, MES= 3: three-point quadratic interpolation, MES= 4: three-point cubic interpolation.
MTESX	20	Maximum number of iterations with changes of the vector $\mathbf{x}$ smaller than TOLX.
MTESF	2	Maximum number of iterations with changes of function values smaller than TOLF.
MIT	10000	Maximum number of iterations.
MFV	20000	Maximum number of function calls.
TOLX	$1.0 \cdot 10^{-16}$	Tolerance for the change of the vector $\mathbf{x}$.
TOLF	$1.0 \cdot 10^{-8}$	Tolerance for the change of the function value.
TOLB	$-1.0 \cdot 10^{-60}$	Minimum acceptable function value.
TOLG	$1.0 \cdot 10^{-6}$	Final accuracy parameter ε.
TOLD	$1.0 \cdot 10^{-4}$	Restart tolerance.
TOLS	0.010	Line search parameter ε_L.
TOLP	0.50	Line search parameter ε_R.
ETA	0.250	Distance measure parameter γ.
XMAX	1000.0	Maximum step size.
NA	$n+3$	Maximum number of stored subgradients m_ξ.

Table 47: Default parameters of program PBUN.

Param.	Value	Significance

MOT	3	Variable specifying the weight updating method.
		MOT= 1: quadratic interpolation,
		MOT= 2: local minimization,
		MOT= 3: quasi-Newton condition.

MES	2	Variable specifying the interpolation method selection in a line search.
		MES= 1: bisection,
		MES= 2: two-point quadratic interpolation.

MTESX	20	Maximum number of iterations with changes of the vector **x** smaller than TOLX.
MTESF	2	Maximum number of iterations with changes of function values smaller than TOLF.
MIT	10000	Maximum number of iterations.
MFV	20000	Maximum number of function calls.
TOLX	$1.0 \cdot 10^{-16}$	Tolerance for the change of the vector **x**.
TOLF	$1.0 \cdot 10^{-8}$	Tolerance for the change of the function value.
TOLB	$-1.0 \cdot 10^{-60}$	Minimum acceptable function value
TOLG	$1.0 \cdot 10^{-6}$	Final accuracy parameter ε.
TOLS	0.010	Line search parameter ε_L.
TOLP	0.50	Line search parameter ε_R.
ETA	0.50	Distance measure parameter γ.
XMAX	1000.0	Maximum step size.
NA	$n+3$	Maximum number of stored subgradients m_ξ.

Table 48: Default parameters of program PBNCGC.

Param.	Value	Significance
RL	0.010	Line search parameter ε_L.
LMAX	10	Maximum number of function calls in line search.
GAM	0.250	Distance measure parameter γ.
EPS	$1.0 \cdot 10^{-6}$	Final accuracy parameter ε.
JMAX	$n+3$	Maximum number of stored subgradients m_ξ.
NITER	10000	Maximum number of iterations.
NFASG	20000	Maximum number of function calls.
BL	$-1.0 \cdot 10^{60}$	Lower bounds of **x**.
BU	$1.0 \cdot 10^{60}$	Upper bounds of **x**.

Table 49: Default parameters of program LVMBM.

Param.	Value	Significance
MOS	2	Distance measure exponent ω.
MIT	10000	Maximum number of iterations.
MFV	20000	Maximum number of function calls.
MTESF	2	Maximum number of iterations with changes of function values smaller than TOLF.
TOLF	$1.0 \cdot 10^{-8}$	Tolerance for the change of the function value.
TOLB	$-1.0 \cdot 10^{-60}$	Minimum acceptable function value
TOLG	$1.0 \cdot 10^{-6}$	Final accuracy parameter ε.
ETA	0.25	Distance measure parameter γ.
EPSL	0.010	Line search parameter ε_L.
XMAX	2.0	Maximum step size.
NA	2	Maximum number of stored subgradients m_ξ.
MC	7	Maximum number of stored corrections m_c.

Table 50: Default parameters of program L-BFGS.

Param.	Value	Significance
DIAGCO	.FALSE.	LOGICAL variable specifying the diagonal matrix $D_k^{(0)}$. DIAGCO = .TRUE.: $D_k^{(0)}$ is provided by user. DIAGCO =.FALSE.: A default value I is used.
DIAG	I	Initial diagonal matrix $D_k^{(0)}$.
GTOL	0.9	Parameter controlling the accuracy of line search.
FTOL	$1.0 \cdot 10^{-4}$	Parameter controlling the accuracy of line search.
STPMIN	$1.0 \cdot 10^{-20}$	Lower bound for the step in line search.
STPMAX	$1.0 \cdot 10^{20}$	Upper bound for the step in line search.
MIT	10000	Maximum number of iterations.
MFV	20000	Maximum number of function calls.
EPS	$1.0 \cdot 10^{-6}$	Final accuracy parameter ε.
M	5	Maximum number of stored corrections m_c.

C Parameters Used in Experiments

With smooth problems 1 – 22 the maximum number of iterations was set to 20000 with $n \leq 1000$ and to 50000 with $n = 10000$. In both cases the maximum number of function evaluations was set to 50000. The bundle methods `PVAR`, `PNEW`, `PBUN`, `PBNCGC` and `LVMBM` were tested with different sizes of bundles (m_ξ) and the limited memory methods `L-BFGS` and `LVMBM` with different numbers of stored corrections (m_c). For bundle-Newton method `PNEW` and for both proximal bundle methods `PBUN` and `PBNCGC` the used sizes of bundles were 10 and $n+3$ and for variable metric bundle methods `PVAR` and `LVMBM` the used sizes of bundles were 2 and $n+3$. For the limited memory programs `L-BFGS` and `LVMBM` the maximum number of stored corrections were first set to 3 and then to 7.

In our experiments, the following values of parameters were used:

- The stopping criterion ε was set to 10^{-6} for all the problems.
- With programs `PVAR`, `PNEW`, `PBUN` and `LVMBM` the maximum number of iterations with changes of function values smaller than 10^{-8} (parameter `MTESF`) was set to 10 for all the problems.
- The used values of distance measure parameter γ (`ETA` in the programs `PVAR`, `PNEW`, `PBUN` and `LVMBM` and `GAM` in the program `PBNCGC`) are given in Tables 51, 53, 55, 57 and 59.
- The values of the maximum step size `XMAX` used with the programs `PVAR`, `PNEW`, `PBUN` and `LVMBM` are given in Tables 52, 54, 56 and 60.
- The values of the line search parameter `RL` used with the program `PBNCGC` are given in Table 58.
- With the program `L-BFGS` the parameter `GTOL` that controls the accuracy of the line search was set to 1.0 for all the problems.
- With the program `L-BFGS` the only parameter changed during the tests was `FTOL`. Also this parameter controls the accuracy of the line search. The values used are given in Table 61.

Otherwise, the experiments were run with the default parameters of programs (see Appendix B).

Table 51: Parameter `ETA` of Program `PVAR`.

	$n = 10$		$n = 100$		$n = 1000$		$n = 500$
No.	$m_\xi = 2$	$m_\xi = 13$	$m_\xi = 2$	$m_\xi = 103$	$m_\xi = 2$	$m_\xi = 1003$	$m_\xi = 2$
1	0.20	0.20	0.01	0.01	1.00	0.25	0.01
2	0.40	0.40	0.10	0.10	0.10	0.01	0.10
3	0.40	0.40	0.40	0.25	0.40	0.10	0.40
4	0.40	0.40	0.40	0.10	0.25	0.25	0.40
5	0.40	0.40	0.40	0.50	0.20	0.20	0.20
6	0.40	0.40	0.40	0.40	0.25	0.01	0.40
7	0.40	0.40	0.40	0.40	0.50	0.10	0.25
8	0.40	0.40	0.40	0.10	1.00	0.50	0.40
9	0.40	0.40	0.01	0.01	0.40	0.01	0.40
10	0.40	0.40	0.10	0.10	0.50	0.15	0.20
11	0.01	0.01	0.10	0.10	0.10	0.01	0.01
12	0.40	0.40	0.25	0.40	1.00	0.25	0.40
13	0.40	0.40	0.25	0.25	0.40	0.50	1.00
14	0.40	0.40	0.001	0.001	$1.0 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$
15	0.40	0.40	0.001	0.001	0.001	0.25	0.20
16	0.40	0.40	0.40	0.50	0.20	0.25	0.25
17	0.20	0.20	0.40	0.10	0.20	0.25	0.25
18	0.40	0.40	1.50	0.01	1.50	1.50	1.00
19	0.40	0.40	0.01	0.01	1.50	0.50	0.40
20	0.40	0.40	0.40	0.40	0.01	0.01	0.50
21	0.40	0.40	0.10	0.01	0.50	0.001	0.10
22	0.40	0.40	0.01	0.01	0.40	0.001	0.40

Table 52: Parameter `XMAX` of Program `PVAR`.

	$n = 10$		$n = 100$		$n = 1000$		$n = 500$
No.	$m_\xi = 2$	$m_\xi = 13$	$m_\xi = 2$	$m_\xi = 103$	$m_\xi = 2$	$m_\xi = 1003$	$m_\xi = 2$
1	2.50	2.50	8.00	8.00	5.00	2.10	3.20
2	1.00	1.00	3.10	2.50	4.50	10.0	2.30
3	2.50	2.50	10.0	10.0	5.00	2.10	5.00
4	2.00	2.00	1.00	1.00	10.0	1.30	1.50
5	2.10	2.10	2.10	10.0	10.0	10.0	10.0
6	2.10	2.10	10.0	10.0	2.10	1.10	1.00
7	1000	1000	1000	1000	100	100	1000
8	1.50	1.50	5.00	5.00	5.00	5.00	10.0
9	5.00	5.00	100	1000	10.0	100	10.0
10	1.00	1.00	100	1000	100	100	100
11	2.10	2.10	1.10	1.10	1.50	2.10	1.10
12	1.50	1.50	2.10	1.10	1.00	2.10	2.10
13	2.10	2.10	2.00	5.00	2.50	10.0	10.0
14	2.10	2.10	1000	1000	1000	1000	1000
15	1000	1000	2.00	10.0	2.10	2.10	1.10
16	2.10	2.10	2.00	5.00	2.10	100	2.10
17	2.10	2.10	1000	1000	1000	2.10	1000
18	1000	1000	5.00	10.0	2.10	1.10	1000
19	2.10	2.10	2.10	2.10	5.00	10.0	1.10
20	1.00	1.00	1.00	1.00	1.10	1.10	3.20
21	1.50	1.50	1000	1000	10.0	1000	1.00
22	1.00	1.00	1.10	1.10	1.00	1000	1.00

Table 53: Parameter `ETA` of Program `PNEW`.

	$n = 10$		$n = 100$		$n = 1000$	$n = 500$
No.	$m_\xi = 10$	$m_\xi = 13$	$m_\xi = 10$	$m_\xi = 103$	$m_\xi = 10$	$m_\xi = 10$
1	1.50	1.50	1.50	1.50	1.50	1.50
2	0.50	0.50	1.00	1.00	0.25	0.25
3	0.25	0.25	0.25	0.25	0.25	0.25
4	0.25	0.25	0.10	0.10	0.50	0.25
5	0.25	0.25	0.25	0.25	0.25	0.25
6	0.25	0.25	0.25	0.25	0.25	0.25
7	0.25	0.25	0.25	0.25	0.25	0.25
8	0.50	0.50	1.00	1.00	0.25	0.25
9	0.25	0.25	0.10	0.10	0.25	0.50
10	0.25	0.25	1.00	1.00	0.25	0.25
11	0.25	0.25	0.50	0.50	0.50	0.50
12	0.25	0.25	0.25	0.25	0.25	0.25
13	1.00	1.00	0.25	0.25	0.25	0.25
14	0.25	0.25	0.25	0.25	0.25	0.25
15	0.10	0.10	0.25	0.25	0.25	0.25
16	0.25	0.25	1.00	1.00	0.25	0.25
17	0.25	0.25	0.25	0.25	0.25	0.25
18	0.25	0.25	0.25	0.25	0.25	0.25
19	0.25	0.25	0.25	0.25	0.25	0.25
20	0.25	0.25	0.25	0.25	0.50	0.50
21	0.25	0.25	0.25	0.25	0.25	0.25
22	0.50	0.50	0.10	0.10	1.00	0.25

Table 54: Parameter `XMAX` of Program `PNEW`.

	$n = 10$		$n = 100$		$n = 1000$	$n = 500$
No.	$m_\xi = 10$	$m_\xi = 13$	$m_\xi = 10$	$m_\xi = 103$	$m_\xi = 10$	$m_\xi = 10$
1	1000	1000	1000	1000	10.0	10.0
2	1000	1000	1000	1000	1000	1000
3	1000	1000	1000	1000	1000	1000
4	1000	1000	5.00	5.00	2.10	1000
5	1000	1000	1000	1000	1000	1000
6	1000	1000	1000	1000	1000	1000
7	1000	1000	1000	1000	1000	1000
8	5.00	5.00	5.00	5.00	1000	1000
9	2.10	2.10	5.00	5.00	1000	10.0
10	2.10	2.10	10.0	10.0	1.10	1.10
11	1000	1000	2.10	2.10	2.10	2.10
12	1000	1000	1000	1000	1000	1000
13	1000	1000	1000	1000	1000	1000
14	1000	1000	1000	1000	1000	1000
15	1000	1000	1000	1000	1000	1000
16	2.10	2.10	10.0	10.0	1000	1000
17	1000	1000	1000	1000	1000	1000
18	1000	1000	1000	1000	1000	1000
19	1000	1000	1000	1000	1000	1000
20	2.10	2.10	2.10	2.10	10.0	10.0
21	1000	1000	1000	1000	1000	1000
22	1000	1000	5.00	5.00	10.0	1000

Table 55: Parameter ETA of Program PBUN.

	$n = 10$		$n = 100$		$n = 1000$		$n = 500$
No.	$m_\xi = 10$	$m_\xi = 13$	$m_\xi = 10$	$m_\xi = 103$	$m_\xi = 10$	$m_\xi = 1003$	$m_\xi = 503$
1	0.20	0.20	0.01	1.00	0.50	0.50	0.50
2	0.50	0.50	0.30	0.25	0.50	0.10	0.01
3	0.30	0.30	0.20	0.20	0.00	0.00	0.10
4	0.50	0.50	0.50	0.50	0.50	0.50	0.40
5	0.50	0.50	0.50	0.50	0.50	0.50	0.40
6	0.50	0.50	0.50	0.50	0.25	0.25	0.50
7	0.50	0.50	0.10	0.50	0.25	0.25	0.50
8	0.50	0.50	0.50	0.50	0.50	0.50	0.50
9	0.00	0.00	0.50	0.50	0.00	0.00	0.50
10	0.50	0.50	0.50	0.50	0.15	0.50	1.10
11	0.30	0.01	0.40	0.40	1.00	1.00	0.20
12	0.50	0.50	0.50	0.50	0.50	0.50	0.50
13	0.50	0.50	0.50	0.50	0.50	0.50	0.40
14	1.00	1.00	0.50	0.50	0.50	0.50	0.50
15	0.50	0.50	0.10	0.10	0.25	0.50	0.40
16	0.50	0.50	0.50	0.50	0.50	0.50	0.50
17	1.00	1.00	0.30	0.30	0.20	0.01	0.01
18	1.00	1.00	0.10	0.10	0.00	0.00	0.01
19	0.50	0.50	0.40	0.40	0.25	0.10	0.01
20	0.25	0.25	0.10	0.10	0.25	0.01	0.10
21	0.00	0.00	0.10	0.10	0.25	0.05	0.00
22	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 56: Parameter XMAX of Program PBUN.

	$n = 10$		$n = 100$		$n = 1000$		$n = 500$
No.	$m_\xi = 10$	$m_\xi = 13$	$m_\xi = 10$	$m_\xi = 103$	$m_\xi = 10$	$m_\xi = 1003$	$m_\xi = 503$
1	2.10	2.10	2.10	5.00	10.0	10.0	3.20
2	4.00	4.00	3.20	4.00	4.50	3.50	2.10
3	1.50	1.50	3.50	3.50	10.0	5.00	3.20
4	2.10	2.10	4.00	4.00	10.0	10.0	3.20
5	1000	1000	4.00	4.00	2.10	2.10	3.20
6	1.10	1.10	3.20	3.20	2.10	2.10	3.20
7	2.10	2.10	2.10	1000	2.10	2.10	2.10
8	5.00	5.00	4.00	4.00	5.00	5.00	3.20
9	1.50	1.50	1000	1000	10.0	10.0	1000
10	1.50	1.50	10.0	10.0	100.	2.10	1000
11	2.10	2.10	10.0	10.0	2.10	2.10	2.10
12	2.10	2.10	2.10	2.10	5.00	5.00	2.10
13	5.00	5.00	3.50	3.50	5.00	5.00	10.0
14	1000	1000	1000	1000	1000	1000	1000
15	1.10	1.10	2.10	2.10	1000	1000	10.0
16	1.50	1.50	4.00	4.00	3.50	5.00	2.10
17	1000	1000	1000	1000	1000	1000	10.0
18	1000	1000	1000	1000	1000	1000	10.0
19	2.10	2.10	4.00	4.00	10.0	100.	1000
20	1000	1000	4.00	4.00	2.10	1.10	10.0
21	1000	1000	1000	1000	2.10	100.	1000
22	1000	1000	1000	1000	1000	1000	1000

Table 57: Parameter GAM of Program PBNCGC.

	$n = 10$		$n = 100$		$n = 1000$		$n = 500$
No.	$m_\xi = 10$	$m_\xi = 13$	$m_\xi = 10$	$m_\xi = 103$	$m_\xi = 10$	$m_\xi = 1003$	$m_\xi = 503$
1	0.25	0.40	0.25	0.40	0.25	0.40	0.45
2	0.20	0.25	0.40	0.40	0.25	0.10	0.45
3	0.25	0.20	0.40	0.25	0.25	0.25	0.25
4	0.00	0.00	0.25	0.20	0.25	0.25	0.40
5	0.25	0.25	0.25	0.25	0.25	0.25	0.25
6	0.25	0.25	0.40	0.25	0.25	0.25	0.25
7	0.00	0.00	0.25	0.40	0.25	0.40	0.25
8	0.25	0.20	0.25	0.25	0.25	0.25	0.25
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.25	0.25	0.25	0.20	0.30
11	0.25	0.25	0.40	0.40	0.20	0.25	0.40
12	0.25	0.25	0.25	0.25	0.40	0.40	0.40
13	0.00	0.00	0.40	0.40	0.25	0.40	0.40
14	0.00	0.00	0.20	0.20	0.40	0.40	0.00
15	0.40	0.40	0.00	0.00	0.20	0.20	0.20
16	0.00	0.00	0.25	0.25	0.25	0.40	0.25
17	0.40	0.40	0.20	0.20	0.20	0.20	0.10
18	0.00	0.00	0.40	0.40	0.20	0.20	0.10
19	0.25	0.25	0.25	0.25	0.25	0.25	0.40
20	0.00	0.00	0.25	0.25	0.20	0.25	0.10
21	0.20	0.20	0.00	0.00	0.00	0.00	0.01
22	0.00	0.00	0.25	0.20	0.00	0.00	0.20

Table 58: Parameter RL of Program PBNCGC.

	$n = 10$		$n = 100$		$n = 1000$		$n = 500$
No.	$m_\xi = 10$	$m_\xi = 13$	$m_\xi = 10$	$m_\xi = 103$	$m_\xi = 10$	$m_\xi = 1003$	$m_\xi = 503$
1	0.010	0.300	0.010	0.300	0.010	0.400	0.450
2	0.400	0.010	0.050	0.050	0.0001	0.010	0.001
3	0.010	0.200	0.100	0.010	0.010	0.010	0.010
4	0.001	0.001	0.010	0.100	0.010	0.010	0.200
5	0.010	0.010	0.010	0.010	0.010	0.010	0.010
6	0.010	0.010	0.100	0.010	0.010	0.010	0.010
7	0.400	0.400	0.010	0.010	0.010	0.400	0.010
8	0.010	0.010	0.010	0.010	0.010	0.010	0.010
9	0.001	0.001	0.010	0.010	0.400	0.400	0.400
10	0.100	0.100	0.010	0.010	0.010	0.100	0.100
11	0.010	0.010	0.001	0.001	0.100	0.010	0.100
12	0.010	0.010	0.010	0.010	0.001	0.001	0.001
13	0.100	0.100	0.300	0.300	0.010	0.200	0.200
14	0.100	0.100	0.300	0.300	0.400	0.400	0.400
15	0.200	0.200	0.010	0.010	0.450	0.450	0.450
16	0.200	0.200	0.010	0.010	0.010	0.200	0.100
17	0.200	0.200	0.400	0.400	0.400	0.400	0.450
18	0.010	0.010	0.400	0.400	0.200	0.200	0.450
19	0.010	0.010	0.010	0.010	0.010	0.010	0.010
20	0.400	0.400	0.010	0.010	0.001	0.010	0.001
21	0.010	0.010	0.400	0.400	0.0001	0.0001	0.450
22	0.010	0.010	0.010	0.0001	0.0001	0.0001	0.010

Table 59: Parameter `ETA` of Program `LVMBM`.

	$n = 10$		$n = 100$		$n = 1000$		$n = 10000$	$n = 500$
No.	$m_\xi = 2$	$m_\xi = 13$	$m_\xi = 2$	$m_\xi = 103$	$m_\xi = 2$	$m_\xi = 1003$	$m_\xi = 2$	$m_\xi = 2$
1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
2	0.25	0.50	0.50	0.75	0.25	0.25	0.25	0.85
3	0.25	0.25	0.05	0.05	0.25	0.25	0.50	0.25
4	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
5	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
6	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
7	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
8	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
9	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
10	0.25	0.25	0.80	0.80	0.20	0.50	0.50	0.95
11	0.25	0.25	0.25	0.25	0.60	0.25	0.25	0.25
12	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
13	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
14	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
15	0.25	0.25	0.25	0.25	0.25	0.50	0.65	1.00
16	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
17	0.25	0.25	0.25	0.25	0.65	0.50	0.65	0.25
18	0.25	0.25	0.25	0.25	0.65	0.65	1.00	0.25
19	0.25	0.25	0.25	0.25	0.95	0.25	0.25	0.25
20	0.25	0.50	0.25	0.25	0.25	0.25	0.25	0.25
21	0.25	0.25	0.25	0.25	0.15	0.50	0.25	0.65
22	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

Table 60: Parameter `XMAX` of Program `LVMBM`.

	$n = 10$		$n = 100$		$n = 1000$		$n = 10000$	$n = 500$
No.	$m_\xi = 2$	$m_\xi = 13$	$m_\xi = 2$	$m_\xi = 103$	$m_\xi = 2$	$m_\xi = 1003$	$m_\xi = 2$	$m_\xi = 2$
1	2.00	2.00	2.00	10.0	2.00	10.0	2.00	2.00
2	10.0	2.00	2.00	2.00	50.0	50.0	2.00	2.00
3	10.0	2.00	10.0	10.0	10.0	10.0	10.0	10.0
4	2.00	2.00	2.00	2.00	2.00	2.00	2.00	10.0
5	2.00	2.00	2.00	2.00	10.0	10.0	2.00	10.0
6	2.00	2.00	10.0	10.0	10.0	10.0	10.0	10.0
7	2.00	2.00	2.00	2.00	2.00	10.0	10.0	10.0
8	10.0	10.0	2.00	2.00	2.00	2.00	2.00	10.0
9	2.00	2.00	10.0	10.0	50.0	50.0	10.0	10.0
10	2.00	2.00	50.0	50.0	10.0	2.00	10.0	10.0
11	1000	1000	10.0	10.0	2.00	2.00	1000	50.0
12	2.00	10.0	2.00	2.00	10.0	10.0	1000	10.0
13	2.00	2.00	10.0	10.0	10.0	10.0	10.0	10.0
14	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
15	2.00	2.00	2.00	2.00	10.0	2.00	2.00	2.00
16	2.00	2.00	10.0	10.0	10.0	10.0	10.0	10.0
17	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
18	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
19	10.0	10.0	10.0	10.0	50.0	2.00	1000	10.0
20	2.00	2.00	2.00	2.00	10.0	10.0	10.0	2.00
21	2.00	2.00	2.00	2.00	2.00	2.00	2.00	10.0
22	1000	10.0	2.00	2.00	1000	10.0	10.0	2.00

Table 61: Parameter FTOL of Program L-BFGS.

No	$n = 10$	$n = 100$	$n = 1000$	$n = 10000$	$n = 500$
1	0.050	$1.0 \cdot 10^{-4}$	0.100	$1.0 \cdot 10^{-4}$	0.100
2	$1.0 \cdot 10^{-4}$	0.500	0.010	0.001	0.010
3	0.500	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
4	0.500	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.100
5	0.500	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.500	$1.0 \cdot 10^{-4}$
6	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
7	$1.0 \cdot 10^{-4}$	0.100	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
8	0.050	0.010	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
9	0.050	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
10	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.100	0.010	$1.0 \cdot 10^{-4}$
11	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.050	$1.0 \cdot 10^{-4}$	0.100
12	$1.0 \cdot 10^{-4}$	0.050	0.010	0.100	0.100
13	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
14	0.500	0.001	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
15	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.500	$1.0 \cdot 10^{-4}$
16	$1.0 \cdot 10^{-4}$	0.100	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
17	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.100	0.100	0.100
18	$1.0 \cdot 10^{-4}$	0.100	$1.0 \cdot 10^{-4}$	0.100	0.001
19	$1.0 \cdot 10^{-4}$	0.100	0.100	0.010	0.010
20	$1.0 \cdot 10^{-4}$	0.010	0.001	0.100	0.100
21	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$	0.050	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$
22	$1.0 \cdot 10^{-4}$	0.100	0.100	$1.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-4}$

D Termination Parameters

In the `UFO` programs (`PVAR`, `PNEW` and `PBUN`) the value of the termination parameter `ITERM` has the following meanings:

`ITERM` $= 1$: $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq$ `TOLX` in `MTESX` subsequent iterations.

`ITERM` $= 2$: $|f_{k+1} - f_k| \leq$ `TOLF` in `MTESF` subsequent iterations.

`ITERM` $= 3$: $f_{k+1} <$ `TOLB`.

`ITERM` $= 4$: The problem has been solved with the desired accuracy.

`ITERM` $= 11$: Number of function evaluations is greater than `MFV`.

`ITERM` $= 12$: Number of iterations is greater than `MIT`.

`ITERM` < 0: Failure in the method:

> `ITERM` $= -6$: The required precision was not achieved.
>
> `ITERM` $= -10$: Two consecutive restarts are required.
>
> `ITERM` $= -12$: The quadratic programming subroutine failed.

In the program `PBNCGC` the value of the termination parameter `IERR` has the following meanings:

`IERR` $= 0$: The problem has been solved with the desired accuracy.

`IERR` $= 1$: Number of function evaluations is greater than `NFASG`.

`IERR` $= 2$: Number of iterations is greater than `NITER`.

`IERR` $= 3$: Invalid input parameters.

`IERR` $= 4$: Not enough working space.

`IERR` $= 5$: Failure in the quadratic problem.

`IERR` $= 6$: The starting point is not feasible.

`IERR` $= 7$: Failure in attaining the demanded accuracy.

In the program LVMBM the value of the termination parameter ITERM has the following meanings:

ITERM $= 1$: The problem has been solved with the desired accuracy.

ITERM $= 2$: Number of function evaluations is greater than MFV.

ITERM $= 3$: Number of iterations is greater than MIT.

ITERM $= 4$: $|f_{k+1} - f_k| \leq$ TOLF in MTESF subsequent iterations.

ITERM $= 5$: $f_{k+1} <$ TOLB.

ITERM < 0: Failure in the method:

- ITERM $= -1$: Two consecutive restarts are required.
- ITERM $= -2$: TMAX $<$ TMIN in two subsequent iterations.
- ITERM $= -3$: Error in updating the limited memory matrices.
- ITERM $= -4$: Failure in attaining the demanded accuracy.

In the program `L-BFGS` the value of the termination parameter `IFLAG` has the following meanings:

`IFLAG` $= 0$**:** The problem has been solved without detecting errors.

`IFLAG` < 0**:** Failure in the method:

- `IFLAG` $= -1$**:** The line search routine `MCSRCH` has failed. The parameter `INFO` provides more detailed information:
  - `INFO` = **0:** Improper input parameters.
  - `INFO` = **2:** Relative width of the interval of uncertainty is at most `XTOL`.
  - `INFO` = **3:** More than 20 function evaluations were required at the present iteration.
  - `INFO` = **4:** The step is too small.
  - `INFO` = **5:** The step is too large.
  - `INFO` = **6:** Rounding errors prevent further progress. There may not be a step satisfying the sufficient decrease and curvature conditions. Tolerances may be too small.
- `IFLAG` $= -2$**:** The ith diagonal element of the diagonal approximation of the inverse of the Hessian matrix, given in `DIAG`, is not positive.
- `IFLAG` $= -3$**:** Improper input parameters for `L-BFGS` (`N` or `M` are not positive).

References

A. Auslender. Numerical methods for nondifferentiable convex optimization. *Mathematical Programming Study*, 30:102–126, 1987.

C. G. Broyden. The convergence of a class of double-rank minimization algorithms, Part I — General considerations, Part II — The new algorithm. *Journal of the Institute of Mathematics and Its Applications*, 6: 76–90, 222–231, 1970.

A. G. Buckley and A. LeNir. QN-like variable storage conjugate gradients. *Mathematical Programming*, 27:155–175, 1983.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal of Scientific Computing*, 16(5):1190–1208, 1995.

R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

R. H. Byrd, J. Nocedal, and Y. Yuan. Global convergence of a class of quasi-Newton methods on convex problems. *SIAM Journal of Numerical Analysis*, 24(3):1171–1189, 1987.

F. H. Clarke. *Optimization and Nonsmooth Analysis.* Wiley-Interscience, New York, 1983.

W. C. Davidon. Variable metric method for minimization. Technical Report ANL-5990 (Rev.), Argonne National Laboratory, Research and Development, 1959.

H. Fayez Khalfan, R. H. Byrd, and R. B. Schnabel. A theoretical and experimental study of the symmetric rank-one update. *SIAM Journal of Optimization*, 3(1):1–24, 1993.

R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.

R. Fletcher. *Practical Methods of Optimization.* John Wiley and Sons, Chichester, second edition, 1987.

R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Computer Journal*, 6:163–168, 1963.

J.-C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, 45:407–435, 1989.

D. Goldfarb. A family of variable metric methods derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

A. Griewank and P. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:119–137, 1982.

A. Grothey. *Decomposition Methods for Nonlinear Nonconvex Optimization Problems*. PhD thesis, University of Edinburgh, 2001.

J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II*. Springer-Verlag, Berlin, 1993.

H. Y. Huang. Unified approach to quadratically convergent algorithms for function minimization. *Journal of Optimization Theory and Applications*, 5:405–423, 1970.

J. E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

K. C. Kiwiel. *Methods of Descent for Nondifferentiable Optimization*. Lecture Notes in Mathematics 1133. Springer-Verlag, Berlin, 1985.

K. C. Kiwiel. A method for solving certain quadratic programming problems arising in nonsmooth optimization. *IMA Journal of Numerical Analysis*, 6:137–152, 1986.

K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46:105–122, 1990.

K. C. Kiwiel. Approximations in decomposition of large-scale convex programs via a nondifferentiable optimization method. In Ü. Jaaksoo and V. I. Utkin, editors, *Proceedings of the 11th Triennial IFAC World Congress, Tallin, Estonia, 1990*, volume 1, pages 161–166. Pergamon Press, Oxford, England, 1991.

T. G. Kolda, D. P. O'Leary, and L. Nazareth. BFGS with update skipping and varying memory. *SIAM Journal of Optimization*, 8(4):1060–1083, 1998.

T. Kärkkäinen, K. Majava, and M. M. Mäkelä. Comparison of formulations and solution methods for image restoration problems. Technical Report B 14/2000, Department of Mathematical Information Technology, University of Jyväskylä, 2000.

T. Kärkkäinen, K. Majava, and M. M. Mäkelä. Comparison of formulations and solution methods for image restoration problems. *Inverse Problems*, 17(6):1977–1995, 2001.

M. Lalee, J. Nocedal, and P. Todd. On the implementation of an algorithm for large-scale equality constrained optimization. *SIAM Journal of Optimization*, 8(3):682–706, 1998.

C. Lemaréchal. An extension of Davidon methods to nondifferentiable problems. In M. L. Balinski and P. Wolfe, editors, *Nondifferentiable Optimization, Mathematical Programming Study 3*, pages 95–109. 1975.

C. Lemaréchal. Combining Kelley's and conjugate gradient methods. In *Abstracts of IX International Symposium on Mathematical Programming*, Budapest, Hungary, 1976.

C. Lemaréchal. Nonsmooth optimization and descent methods. Technical Report 78/4, IIASA, Laxemburg, Austria, 1978.

C. Lemaréchal. Numerical experiments in nonsmooth optimization. In E. A. Nurminski, editor, *Proceedings of the IIASA workshop on Progress in Nondifferentiable Optimization*, pages 61–84, Laxemburg, Austria, 1982.

C. Lemaréchal. Nondifferentiable optimization. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, pages 529–572. North-Holland, Amsterdam, 1989.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

L. Lukšan. Dual method for solving a special problem of quadratic programming as a subproblem at linearly constrained nonlinear minmax approximation. *Kybernetika*, 20:445–457, 1984.

L. Lukšan. Computational experience with improved variable metric methods for unconstrained minimization. *Kybernetika*, 26:415–431, 1990.

L. Lukšan and E. Spedicato. Variable metric methods for unconstrained optimization and nonlinear least squares. *Journal of Computational and Applied Mathematics*, 124:61–95, 2000.

L. Lukšan, M. Tůma, M. Šiška, J. Vlček, and N. Ramešová. UFO 2000: Interactive system for universal functional optimization. Technical Report 826, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2000.

L. Lukšan and J. Vlček. A bundle-Newton method for nonsmooth unconstrained minimization. *Mathematical Programming*, 83:373–391, 1998.

L. Lukšan and J. Vlček. Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory and Applications*, 102:593–613, 1999a.

L. Lukšan and J. Vlček. Sparse and partially separable test problems for unconstrained and equality constrained optimization. Technical Report 767, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1999b.

L. Lukšan and J. Vlček. Introduction to nonsmooth analysis. Theory and algorithms. Technical Report DMSIA 1/2000, University of Bergamo, 2000a.

L. Lukšan and J. Vlček. NDA: Algorithms for nondifferentiable optimization. Technical Report 797, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2000b.

L. Lukšan and J. Vlček. Variable metric methods for nonsmooth optimization. Technical Report 837, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2001.

K. Miettinen, M. M. Mäkelä, and T. Männikkö. Optimal control of continuous casting by nondifferentiable multiobjective optimization. *Computational Optimization and Applications*, 11:177–194, 1998.

M. M. Mäkelä. Issues of implementing a Fortran subroutine package NSOLIB for nonsmooth optimization. Technical Report 5/1993, Department of Mathematics, Laboratory of Scientific Computing, University of Jyväskylä, 1993.

M. M. Mäkelä. Methods and algorithms for nonsmooth optimization. Reports on Applied Mathematics and Computing 2, Department of Mathematics, University of Jyväskylä, 1998.

M. M. Mäkelä. Survey of bundle methods for nonsmooth optimization. *Optimization Methods and Software*, 17(1):1–29, 2002.

M. M. Mäkelä, M. Miettinen, L. Lukšan, and J. Vlček. Comparing nonsmooth nonconvex bundle methods in solving hemivariational inequalities. *Journal of Global Optimization*, 14:117–135, 1999.

M. M. Mäkelä and P. Neittaanmäki. *Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control.* World Scientific Publishing Co., Singapore, 1992.

J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

J. Nocedal. Theory of algorithms for unconstrained optimization. *Acta Numerica*, 1:199–242, 1992.

J. Nocedal. Large scale unconstrained optimization. In A. Watson and I. Duff, editors, *The State of the Art in Numerical Analysis*, pages 311–338. Oxford University Press, 1997.

R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Optimal Control and Optimization*, 14:877–898, 1976.

H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM Journal of Optimization*, 2:121–152, 1992.

D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24:647–657, 1970.

N. Z. Shor. *Minimization Methods for Non-Differentiable Functions.* Springer-Verlag, Berlin, 1985.

P. L. Toint. On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation*, 31(140):954–961, 1977.

J. Vlček. Bundle algorithms for nonsmooth unconstrained optimization. Technical Report 608, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1995.

J. Vlček and L. Lukšan. Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. Technical Report B 8/1999, Department of Mathematical Information Technology, University of Jyväskylä, 1999.