UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Quasi-Newton methods for nonsmooth Riemannian optimization

Dissertation presented by
Florentin GOYENS

for obtaining the Master's degree in
Mathematical Engineering

Supervisor(s)
Pierre-Antoine ABSIL , Wen HUANG

Reader(s)
Paul VAN DOOREN

Academic year 2015-2016

Acknowledgements

I would first like to thank my thesis advisors, Pr. Pierre-Antoine Absil and Dr. Wen Huang for their availability throughout the semester, and the interest they were able to give me for Riemannian optimization. I am also grateful to my reader Pr. Paul Van Dooren for accepting to be part of the jury and taking the time to read this Master thesis. Finally, I would also thank my friends and familly for their support and making these 5 years at university such a tremendous experience.

Contents

Introduction

The field of optimization deals with the process of minimizing the values of a function over its domain. Applications are countless, going from the control of systems to computer vision problems . In the past decades, a notable interest has grown for optimization problems defined on Riemannian manifold. It has become a highly active field of research, as Riemannian optimization offers efficient alternative formulations to many problems in engineering and science. Some of these problems are known to have nonsmooth cost functions. For nonsmooth optimization problem, people usually resorts to specifically designed algorithm. Nonsmooth optimization is still an area under development. Recent work by Lewis and Hoverton [LO13], shows that under some conditions, the quasi-Newton methods do very well in practice on nonsmooth problems. The concept of this master thesis is to study quasi-Newton methods on Riemannian manifolds for nonsmooth problems. Which is a very recent idea.

We first establish some preliminary notions in section 1.1. Where well known quasi-Newton methods for smooth unconstrained optimization in $\mathbb{R}^n$ are described. Follows section 1.2, a description of the work done in [LO13], for nonsmooth quasi-Newton methods in $\mathbb{R}^n$. We conclude the background and state of the art chapter with a minimal introduction to Riemannian optimization, section 1.3.

The second chapter describes the algorithms that were developed in this project. Section 2.1 shows how a nonsmooth quasi-Newton method is adapted to the Riemannian settings. Two majors challenges appear. A line search is defined based on the Wolfe conditions. We also address the question of a reliable stopping criterion for nonsmooth functions. Some variants of the algorithms are explored: a limited-memory version for the updates of the Hessian approximation; as well as a subgradient approach to generate the search directions.

The third chapter defines the applications that we have shown interest in. The first in section 3.1, is a sparse approximation of the principal component of a matrix, namely sparse PCA. The second application comes from computer vision. As the goal is to find the minimal volume box enclosing a set of points (section 3.2). This problem is labelled orthogonal bounding box or OBB.

In the final chapter, we present results of our experiments. We study the behaviour of each method and compare their performances. We also bring insight on the nature of the test problems. As a conclusion, we synthesize the main properties of each algorithm and discuss whether our methods are competitive for the problems considered.

Chapter 1

Background and state of the art

We recall some typical notions in optimization that are a foundation for the methods used in this master thesis. We then explain a state of the art method in nonsmooth Euclidean optimization (section 1.2). Finally, we give a basic explanation of the concepts behind optimization on Riemannian manifolds.

1.1 Smooth Euclidean BFGS and LBFGS methods

In Euclidean optimization we deal with the general problem of minimizing a real valued function f over the Euclidean space $\mathbb{R}^n$, namely $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. We write

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x).$$

A popular class of methods to solve this general problem are the line-search methods. They begin with an initial point $x_0 \in \mathbb{R}^n$ and produce a sequence of iterates $x_0, x_1, x_2 \ldots$ that we hope will converge towards a minimum of the problem. The method is defined by a search direction $p_k \in \mathbb{R}^n$ and a step size $\alpha_k \in \mathbb{R}$, for each step k. The next iterate is defined as $x_{k+1} = x_k + \alpha_k p_k$.
A famous example is the Newton method defined as

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k).$$

This method has known quadratic convergence properties. Quadratic convergence of a converging sequence in $\mathbb{R}^n$ is defined in definition 1.1. In practice, this means that when close enough to the limit, the convergence is extremely fast.

Definition 1.1 ***(Quadratic convergence)***
Let $(x_k)_{k\in\mathbb{N}}$ be a sequence in $\mathbb{R}^n$. The sequence is said to convergence quadratically to a minimizer x_* if there exists $c \geq 0$ such that for all k sufficiently large

$$\|x_{k+1} - x_*\| \leq c\|x_k - x_*\|^2. \tag{1.1}$$

There are unfortunately important drawbacks. The function must be twice differentiable and have an invertible Hessian for the method to be well defined. Finding the direction $p \in \mathbb{R}^n$ at each step requires to solve the linear system $\nabla^2 f(x)p = -\nabla f(x)$ which has a $\mathcal{O}(n^3)$ time complexity in general. One may also store the Hessian with $\mathcal{O}(n^2)$ entries. It is also possible to only use the action of the Hessian to solve the system. This makes the method impractical for problems of large dimension. There are also robustness issues as there is no guaranty to converge if the initial iterate x_0 is not close enough to the minimizer.

This is why we turn to the class of *quasi-Newton* methods. They have always received a lot of attention as they try to keep some of the good properties of the Newton method but solve the issues that we just mentioned. The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is a generalization of the one dimensional secant method. An approximation of the Hessian is updated at each iteration, $B_k \approx \nabla^2 f(x_k)$. A popular choice is to take $B_0 = Id$ as an initial approximation and then B_k is chosen such that $B_k\Big(x_{k+1} - x_k\Big) = \Big(\nabla f(x_{k+1}) - \nabla f(x_k)\Big)$. The BFGS update formula is one of the many possible choices that enforces the secant equation. The update formula is presented in algorithm 1.

Algorithm 1 Smooth Euclidean BFGS algorithm

```
1: Input:  $x_0,\ k = 0,\ \varepsilon > 0$
2: while $\|\nabla f(x_k)\| > \varepsilon$ do
3:     Obtain $p_k$ from the linear system $B_k p_k = -\nabla f_k$
4:     Perform line search to find step size $\alpha_k$ and set $x_{k+1} = x_k + \alpha_k p_k$
5:     Set $s_k = \alpha_k p_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
6:     Update the Hessian approximation
           $$B_{k+1} = B_k + \frac{y_k y_k^T}{s_k^T y_k} - \frac{B_k s_k s_k^T B_k}{s_k B_k s_k}. \quad (1.2)$$
7: $k \leftarrow k + 1$
```

At step 4 a line search is performed. This is a one dimensional minimization along direction p_k. Line search methods will be discussed later. An important remark must be made for implementation purposes. It is an approximation of the inverse Hessian that is stored, $H_k \approx \nabla^2 f(x_k)^{-1}$. There is an equivalent update formula for the inverse

$$H_{k+1} = H_k + (s_k^T y_k + y_k^T H_k y_k)\frac{s_k s_k^T}{(s_k^T y_k)^2} - \frac{(H_k y_k s_k^T + s_k y_k^T H_k)}{(s_k^T y_k)}. \tag{1.3}$$

This allows to compute the direction at step 3 as $p_k = -H_k \nabla f_k$. Which is a matrix-vector product performed in $O(n^2)$ instead of the resolution of a linear system in $O(n^3)$. The number of operations for the scheme is of the order of $\mathcal{O}(n^2)$ at each step. For very large problems this can still be expensive. That is when the limited memory version of the algorithm can be considered. Instead of storing the Hessian approximation and update it at each step, we only store the m previous updates for a value $m \ll n$. Algorithm 2 describes the limited memory version.

Method	Newton	BFGS	LBFGS
Cost per iteration	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(m * n)$

The BFGS method does not have a clear convergence analysis. It was shown that if f is convex and twice continuously differentiable, and the sublevel set $\{x : f(x) \leq f(x_0)\}$ is bounded, then the sequence of function values generated by the BFGS method with an inexact Armijo–Wolfe line search converges to the minimal value of f [Pow76]. In the nonconvex case, some pathological counterexamples to convergence have been found. But it is widely accepted that the method works well in practice for both the convex and nonconvex case [LO13]. BFGS is known to converge superlinearly under some conditions [NW00]. Superlinear convergence is defined in definition 1.2. This type of convergence may not be good as the quadratic convergence of the Newton method. But it is still very fast and the trade-off that BFGS offers for more robustness is what makes it one of the most widespread methods in optimization.

Algorithm 2 Smooth Euclidean LBFGS algorithm [AS10]

```
1: input : $x_0$, $k = 0$, $\varepsilon$
2: while $\|\nabla f(x_k)\| > \varepsilon$ do
3:     Obtain $p_k$ as follows
4:     for $i = k-1, k-2, \dots, k-m$ do
5:         $\xi_i \leftarrow \rho_i s_i^T q$;
6:         $q \leftarrow q - \xi_i y_i$;
7:     $r \leftarrow H_k^0 q$;
8:     for $i = k-m, k-m, +1, \dots, k-1$ do
9:         $\omega \leftarrow \rho_i y_i^T$;
10:        $r \leftarrow r + s_i(\xi_i - \omega)$;
11:    Set $p_k = -r$
12:    Perform line search to find step size $\alpha_k$ and set $x_{k+1} = x_k + \alpha_k p_k$
13:    Set $s_k = \alpha_k p_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
14:    $k \leftarrow k+1$
```

Definition 1.2 ***(Superlinear convergence)***
Let $(x_k)_{k\in\mathbb{N}}$ be a sequence in $\mathbb{R}^n$. The sequence is said to convergence superlinearly to a minimizer x_* if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = 0. \tag{1.4}$$

In definition 1.3, we define the linear convergence. This type of convergence is considered slow. The same number of iterations is required to gain every digit of accuracy. This is a characteristic of the gradient method.

Definition 1.3 ***(Linear convergence)***
Let $(x_k)_{k\in\mathbb{N}}$ be a sequence in $\mathbb{R}^n$. The sequence is said to convergence linearly to x_* if there exists $\mu \in]0,1[$ such that

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = \mu \tag{1.5}$$

If as $k \to \infty$, $\mu \to 1$ this is called a sublinear rate. Whereas if $\mu \to 0$, we go back to the definition of superlinear convergence.

We have described quasi-Newton methods for smooth optimization in $\mathbb{R}^n$. Their efficiency in this context has been known for quite some time. The next section looks at their application to nonsmooth cost functions. A much more recent area of interest.

1.2 Nonsmooth Euclidean BFGS

The BFGS method of algorithm 1 is defined for smooth functions in $\mathbb{R}^n$. The user must be able to provide the gradient of the cost function at any point. The behaviour of quasi-Newton methods on nonsmooth problems had received very little attention in the past. Recent work has shown that the BFGS method can also be very efficient in the nonsmooth case [LO13]. In 2013, Lewis and Overton experiment that, under some assumptions, the iterations of the BFGS method converge to a minimizer of f. We go trough a few definitions to explain the context of their work.

Definition 1.4 ***(Locally Lipschitz functions)***
A function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is said to be locally Lipschitz when for every $x \in \mathbb{R}^n$, there exists a neighbourhood U of x such that f is Lipschitz continuous on U. There $\exists L > 0$ such that for every $x, y \in U$, $\| f(x) - f(y) \| \leq L \| x - y \|$.

Lewis and Overton based their analysis on locally Lipschitz functions. Local Lipschitz continuity implies that the set of nondifferentiable points has measure zero [LO13]. In other words, there exists $D \subset \mathbb{R}^n$, with D dense in $\mathbb{R}^n$ such that f is smooth on D.
The optimality condition must also be revised for nonsmooth functions. Indeed the usual first order $\nabla f(x) = 0$ does not work. The Clarke subdifferential was introduced by F.H. Clarke and R.T. Rockafellar in the early 1980s as a generalization of the subdifferential to nonconvex functions [BKM14]. A Clarke stationary point is such that the subdifferential at this point contains zero.

Definition 1.5 ***(Clarke subdifferential [Hua13])***
The generalized directional derivative of a locally Lipschitz continuous function $f : \mathbb{R}^n \to \mathbb{R}$ evaluated at x in the direction v is given by

$$f^o(x; v) = \limsup_{y \to x} \frac{f(y + \lambda v) - f(y)}{\lambda}, \tag{1.6}$$

and the Clarke subdifferential of f at x is the set

$$\partial f(x) = \{\eta \in \mathbb{R}^n : f^o(x; v) \geq \langle v, \eta \rangle \textit{ for all } v \textit{ in } \mathbb{R}^n\}. \tag{1.7}$$

A Clarke stationary point x^* is such that $0 \in \partial f(x^*)$.

This is the standard optimality condition for nonsmooth functions. In the case of locally Lipschitz functions, the Clarke subdifferential at x, written $\partial f(x)$, is simply the convex hull of the limits of gradients of f evaluated at sequences converging to x [GH15]. An element of $\partial f(x)$ is called a subgradient of f at x.

Proposition 1.1 ***(based on [BKM14])***
Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz on $\mathbb{R}^n$. Then

1. *There exists $D \subset \mathbb{R}^n$, with D dense in $\mathbb{R}^n$ such that f is smooth on D;*

2. *The Clarke subdifferential of f at $x \in \mathbb{R}^n$ is the convex hull of the limits of gradients of f evaluated at sequences converging to x*

$$\partial f(x) = conv\Big\{ \lim_{i \to \infty} \nabla f(x_i) : \{x_i\} \subset D, x_i \to x \Big\}.$$

A key point is that, since locally Lipschitz functions are differentiable almost everywhere, as long as the method is initialized with a random x_0 and an inexact line search is used, it is very unlikely that the sequence of iterations will reach a point where the objective is not differentiable. Under the assumption that such a point is never encountered, the BFGS method is well defined for nonsmooth functions. Lewis and Overton bring two modifications to the BFGS algorithm to account for the nonsmoothness of the function and guarantee that the method will be well defined with probability one. They give an inexact line search and a new stopping criterion.

1.2.1 The nonsmooth line search

For nonsmooth optimization, the conditions for the line search must be chosen carefully. Let $h(\alpha) = f(x + \alpha p) - f(x)$ be the line search objective function, where $p \in \mathbb{R}^n$ is the search direction and $\alpha \in \mathbb{R}$ is the step size. Define $s = \limsup_{\alpha \to 0} \dfrac{h(\alpha)}{\alpha}$, the initial slope. We make the assumption that $s < 0$ which means that p is a descent direction. Some terminology about the line search conditions is fixed in definition 1.6.

Definition 1.6 ***(Wolfe conditions)***
For parameters $0 < c_1 < c_2 < 1$, we define the following line search conditions

$$\begin{array}{rll} A(\alpha): & h(\alpha) < c_1 s \alpha & \textit{(Armijo condition)} \\ W(\alpha): & h \textit{ is differentiable at } \alpha \textit{ with } h'(\alpha) > c_2 s & \textit{(Weak curvature condition)} \\ S(\alpha): & h \textit{ is differentiable at } \alpha \textit{ with } |h'(\alpha)| < c_2|s| & \textit{(Strong curvature condition)} \end{array} \tag{1.8}$$

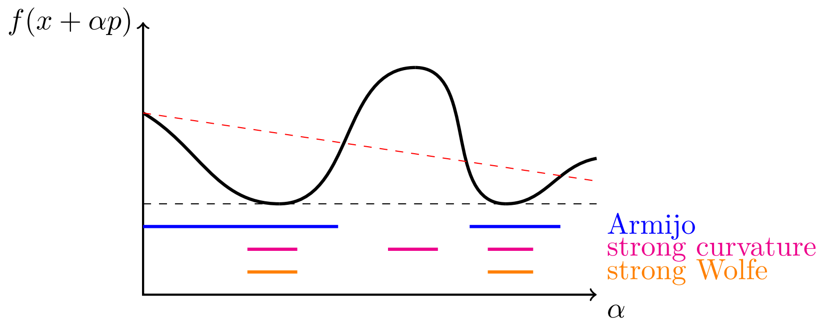
If some step α satisfies conditions $A(\alpha)$ and $W(\alpha)$, one says that it satisfies the weak Wolf conditions. Whereas $A(\alpha)$ and $S(\alpha)$ give the strong Wolfe conditions.

The strong Wolfe conditions are often preferred in smooth optimization. However, a rather simple example illustrates why the weak conditions must be used for nonsmooth optimization. Figure 1.1 shows the strong Wolfe condition for a smooth and a nonsmooth one dimensional minimization. The segment in blue shows the points that satisfy the sufficient decrease condition $A(\alpha)$. The purple points satisfy the condition $S(\alpha)$. And the orange points are admissible for the strong Wolfe conditions. As we can see, this works perfectly in the smooth case but does not give any admissible step length in the nonsmooth case because the derivative is never small enough in absolute value near the minimizers. In the case of the weak conditions $W(\alpha)$, any positive slope is accepted and this allows to have admissible intervals for the step length, even though the slope never gets close to zero, see figure 1.2.

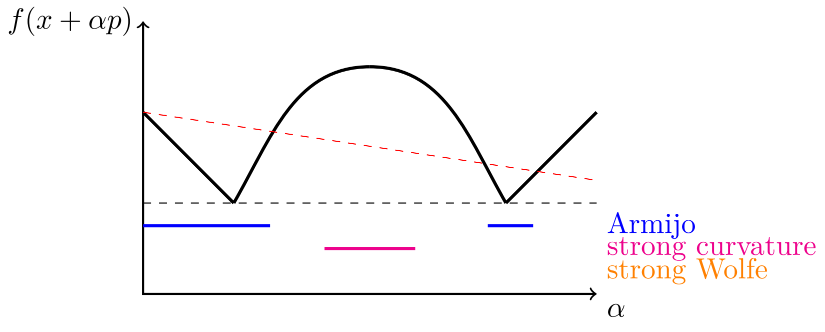
To get a step that satisfies the weak Wolfe conditions we introduce algorithm 3. This is a backtracking line search procedure. In [LO13], the existence of a step satisfying the weak Wolfe conditions is established for locally Lipschitz functions with $s < 0$. They also prove that when algorithm 3 terminates, it returns a weak Wolfe step.

Algorithm 3 Nonsmooth line search: Backtracking for weak Wolfe condition [LO13]

```
1: **Input:** $x_k$, $k \geq 0$
2: $\gamma = 0$
3: $\beta = +\infty$
4: $\alpha_k = 1$
5: **while** 1 **do**
6:     **if** $A(\alpha_k)$ fails **then** $\beta = \alpha_k$
7:     **else if** $W(\alpha_k)$ fails **then** $\gamma = \alpha_k$
8:     **else** break
9:     **if** $\beta < +\infty$ **then** $\alpha_k = (\gamma + \beta)/2$
10:    **else** $\alpha_k = 2\gamma$
11: **Output:** *Next iterate* $x_{k+1} = x_k + \alpha_k p_k$
```

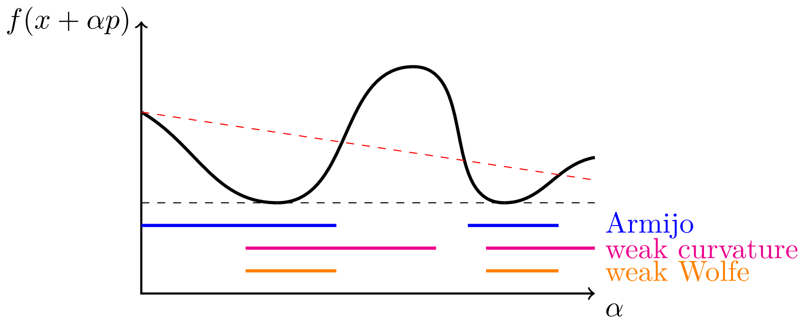


(a) Smooth $\varphi(\alpha)$

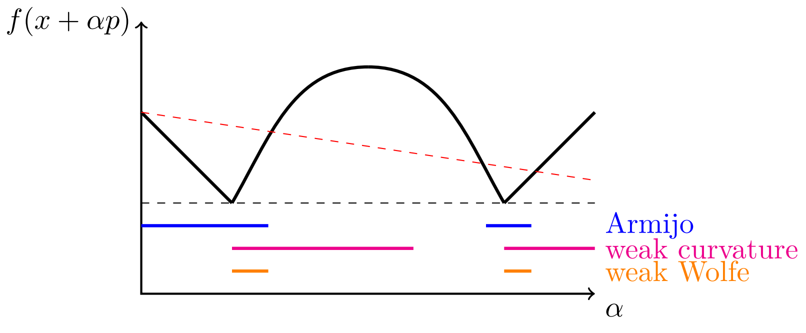


(b) Nonsmooth $\varphi(\alpha)$

FIGURE 1.1: Strong Wolfe conditions for smooth and nonsmooth line search.



(a) Smooth $\varphi(\alpha)$



(b) Nonsmooth $\varphi(\alpha)$

FIGURE 1.2: Weak Wolfe conditions for smooth and nonsmooth line search.

This backtracking procedure has a probability zero to find the exact minimizer in the search direction. One can understand why it is important for the minimization to be inexact. In the case of nonsmooth optimization, the exact minimum in the search direction will more likely be a point where there is no differential. Reaching such a point would cause the algorithm to break down. In other words, if we were able to perform an exact minimization for the line search, we should not use it. Because using an inexact line search is precisely what allows the BFGS method to stay well defined.

1.2.2 The nonsmooth stopping criterion

It is clear that a usual criterion such as the norm of the gradient cannot be used successfully. A (local) minimum will likely be a point where there is no gradient at all. We therefore introduce the modified nonsmooth criterion from [LO13]. Set an integer $J \geq n+1$ and two small constants ε_x and ε_d. We use the ℓ_2 norm, written $\|.\|$ in this section. As long as following iterates satisfy $||x_{k+1} - x_k|| < \varepsilon_x$, we save the gradient at each point and store up to J gradients. Once for some k we have $||x_{k+1} - x_k|| > \varepsilon_x$, then all previously stored gradients are freed from memory. This collection of gradients at points near x_k is stored in a set G_k. At each iteration we compute the smallest element in the convex hull of the gradients stored, call it $d_k \in \mathbb{R}^n$

$$d_k = \text{argmin}\{\|d\|^2 : d \in \text{conv}G_k\}.$$

If $\|d\| < \varepsilon_d$, then the stopping criterion is met and the algorithm stops. Indeed, the previous process is an approximated condition of Clarke optimality. If we are close enough to a Clarke stationary point, the convex hull of enough nearby points should contain the null vector. The parameter J must be large enough to have enough gradients to reconstruct the zero vector and not too large because computing the convex hull is expensive. In practice it is expected that J must be at least larger than the dimension n.

The stopping criterion requires to solve a convex quadratic optimization problem. Let $G_k = \{g_1, g_2, \dots g_{j_k}\}$ with $j_k < J$ be the stored gradients. The minimum norm problem can be expressed as follows

$$\begin{array}{ll}\underset{x,a_i}{\text{minimize}} & ||x||^2 \\ \text{subject to} & \\ & x = a_1 g_1 + a_2 g_2 + \cdots + a_{j_k} g_{j_k} \\ & \displaystyle\sum_{i=1}^{j_k} a_i = 1 \\ & a_i \geq 0 \ \forall i = 1, 2, \dots, j_k \end{array} \tag{1.9}$$

It is a convex quadratic problem of dimension $n + j_k$. The resolution of this problem will be discussed in section 2.1.3. Algorithm 4 is a formal description of the nonsmooth stopping criterion based on [LO13].

In our opinion, this algorithm 4 is cleaver because it allows not to compute a full minimum norm problem of size J at each iteration. This would be the case if we were to evaluate the gradient at J nearby points and compute the convex hull at each step. This is the classical way to approximate the subdifferential. The algorithm waits for the sequence of iterations to gather in the same neighbourhood before starting to store several gradients. That is why it seems important to take the parameter ε_x small enough. On the other hand, this approach is a bit sensitive. When we are close enough to a minimizer, it might be necessary to continue the iteration process to recreate the entire subdifferential and form the null vector. This will take some time depending on how well the iterations bring information about the subdifferential around the minimizer.

Algorithm 4 Nonsmooth stopping criterion [LO13]

```
1: Input:  $J \in \mathbb{N}$; $\varepsilon_x$, $\varepsilon_d > 0$
2: At iteration $k$
3: if $\|x_k - x_{k-1}\| > \varepsilon_x$ then $j_k = 1, G_k = \{\mathrm{grad} f_k\}$
4: if $\|x_k - x_{k-1}\| \leq \varepsilon_x$, and $j_{k-1} < J$ then $j_k = j_k + 1, G_k = \{\mathrm{grad} f_{k-j_k+1}, \ldots, \mathrm{grad} f_k\}$
5: if $\|x_k - x_{k-1}\| \leq \varepsilon_x$, and $j_{k-1} = J$ then $j_k = J, G_k = \{\mathrm{grad} f_{k-J+1}, \ldots, \mathrm{grad} f_k\}$
6: $d_k = \mathrm{argmin}\{\|d\|^2 : d \in \mathrm{conv} G_k\}$
7: if $\|d_k\| < \varepsilon_d$ then STOP
8: else Compute $x_{k+1}$
```

1.2.3 Performances of nonsmooth BFGS

We shall first observe what the quasi-Newton methods do in smooth optimization. This will serve as evidence for the claims of superlinear convergence made earlier. We take the smooth function

$$f_{smooth} : \mathbb{R}^2 \longrightarrow \mathbb{R} : (x_1, x_2) \longmapsto (1 - x_1)^2 + \left(x_2 - x_1^2\right)^2. \tag{1.10}$$

The global minimizer is $x^* = (1, 1)$ and the optimal function value $f^* = 0$. We ran the BFGS and Newton method with the following settings:

x_0	$(-1, -1)$
Stopping criterion:	$\|\nabla f(x)\| < 10^{-12}$

The results are on figure 1.3. The distance to x^* is represented in log scale. For the Newton method, the quadratic convergence is clearly observed. As each iteration doubles the precision. Between iterations 8 and 9, the distance goes from 10^{-15} to 10^{-30}. Whereas for BFGS we observe superlinear convergence. This is still very fast and much faster than linear convergence. Similar results are obtained whether we look at the error of the function values or the distance to the minimizer x^*.

For our next example, we finally apply BFGS to a nonsmooth problem. We choose the cost function

$$f_1 : \mathbb{R}^2 \longrightarrow \mathbb{R} : (x_1, x_2) \longmapsto \|x\|_1 = |x_1| + |x_2|.$$

This is the ℓ_1 norm function in $\mathbb{R}^2$. This function was chosen because it can be visualized and has kink lines on the axis along which the function is nondifferentiable. And we are looking to see how this will affect the convergence of the iterations. It has a unique global minimizer in $x^* = (0, 0)$. The following parameters were used:

x_0	Initialized randomly
Stopping criterion:	Algorithm 4 with $J = 10$, $\varepsilon_x = 10^{-6}$, $\varepsilon_d = 10^{-12}$.

We see on figure 1.4 that the convergence to x^* is only linear. Has the distance to optimality (in log scale) is a straight line of slope -1. We make the important observation that the nonsmooth character of the cost function has reduced the speed of convergence from superlinear to linear. This confirms the observations in [LO13] for nonsmooth quasi-Newton methods.

To the author's knowledge, there has not been many effort spent on nonsmooth limited memory BFGS. A first promising step is found in [AS10]. They experiment that LBFGS

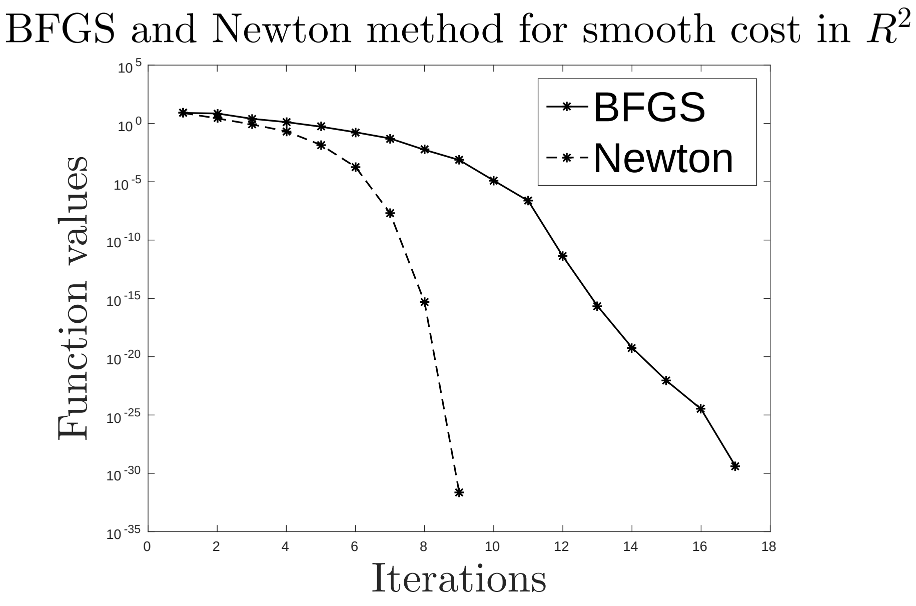


FIGURE 1.3: Quadratic and superlinear convergence of the BFGS and Newton methods for smooth cost f_{smooth}.

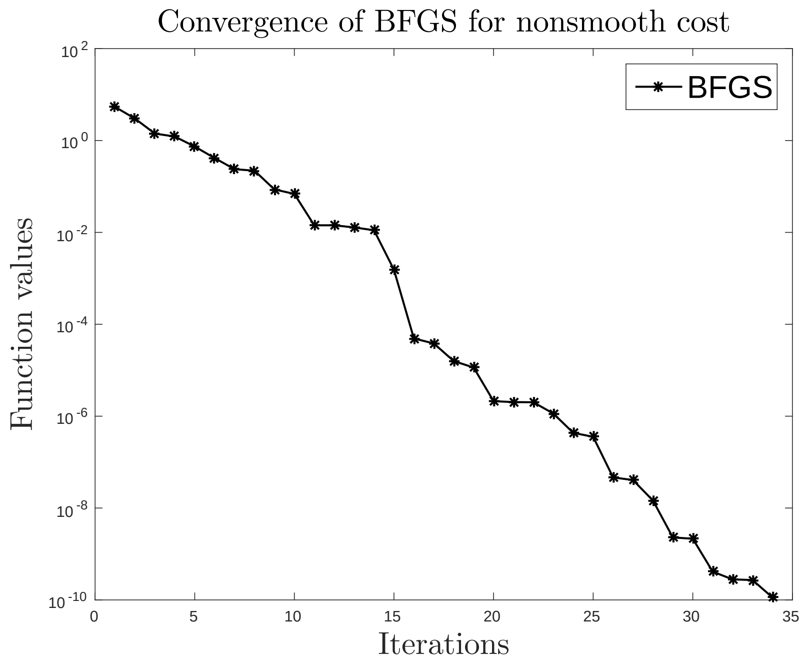


FIGURE 1.4: Linear convergence of the distance to the minimizer in log scale for f_1.

competes with other nonsmooth methods to minimize some nonsmooth functions in $\mathbb{R}^n$. LBFGS seems mostly designed for large scale problems.

This section was the last one dedicated to optimization in $\mathbb{R}^n$. We have looked at the common themes when quasi-Newton methods are applied to nonsmooth optimization. We have seen the difficulties that come with a nonsmooth cost. In the next section we introduce the notion of a Riemannian manifold. The rest of this work will focus on minimizing nonsmooth functions defined on such spaces.

1.3 Riemannian optimization

In the past years, a large number of applications have appeared where the cost function of the optimization problem is defined over a Riemannian manifold. We will describe the properties of manifolds and explain the main concepts in Riemannian optimization. For a more complete reference on the subject, see [AMS08].

1.3.1 Riemannian manifolds

A manifold is a set $\mathcal{M}$ that is locally homeomorphic to the Euclidean space. This informally means that locally there is always a map from a subset of $\mathcal{M}$ to an open subset of $\mathbb{R}^n$. At each point $x \in \mathcal{M}$, the tangent space $T_x\mathcal{M}$ is associated with an inner product $\langle \cdot, \cdot \rangle_x$. The inner product is a bilinear, symmetric and positive-definite form. It induces a norm for every $\xi \in T_x\mathcal{M}$,

$$\|\xi\|_x := \sqrt{\langle \xi, \xi \rangle_x}.$$

This norm is essential to define the notion of length for a search direction. When the tangent spaces are endowed with a smooth inner product, the manifold is said to be a Riemannian manifold. The inner product is called a Riemannian metric and the subscript is often omitted when no confusion is possible, $\langle \xi, \eta \rangle_x = \langle \xi, \eta \rangle$.

Let $\mathcal{M}$ be a Riemannian manifold. Riemannian optimization consists in minimizing a real valued function on a manifold, $f : \mathcal{M} \longrightarrow \mathbb{R}$. For most applications, including the ones covered in this text, the manifolds are embedded submanifolds of $\mathbb{R}^{n\times m}$. This allows a natural matrix representation of the points on the manifold.

In this section we describe some specific concepts related to Riemannian optimization. These concepts can be envisioned in a more general framework of differential geometry. The goal here is to introduce them in a straightforward way so that they can be applied to optimization algorithms. But first, we look at a couple example of manifold that we will encounter further in our test problems.

Example 1.1 ***(The unit sphere)*** *The unit sphere $\mathbb{S}^{n-1} = \{x \in \mathbb{R}^n : \|x\|_2 = 1\}$ is a nice and simple example of a manifold. It is embedded in the Euclidean space $\mathbb{R}^n$. This assures that we have a matrix representation for each point on the manifold. The unit sphere is naturally equipped with the inner product of the embedding space $\mathbb{R}^n$: $\langle u, v \rangle = u^T v \ \forall u, v \in T_x\mathbb{S}^{n-1}$. For a point $x \in \mathbb{S}^{n-1}$, the tangent space is expressed as $T_x\mathbb{S}^{n-1} = \{v \in \mathbb{R}^n : x^T v = 0\}$.*

Example 1.2 ***(The orthogonal group)*** *Another manifold that we will encounter is the orthogonal group $O_n = \{Q \in \mathbb{R}^{n\times n} : Q^T Q = I_n\}$. The manifold is endowed with the Euclidean inner product $\langle Q_1, Q_2 \rangle = trace(Q_1^T Q_2)$. This inner product induces the Frobenius norm. In the plane, $O_2 \subset \mathbb{R}^{2\times 2}$ only has one dimension as it is a rotation depending on one angle. The manifold $O_3 \subset \mathbb{R}^{3\times 3}$ has dimension 3. In general, the manifold has dimension $\frac{1}{2}n(n-1)$. Let $U \in O_n$, then the tangent space $T_U O_n = \{U\Omega : \Omega^T = -\Omega\} = US_{skew}(n)$, where $S_{skew}(n)$ denotes the set of all skew-symmetric $n \times n$ matrices [AMS08]. The tangent space has dimension $\frac{n(n-1)}{2}$, the degrees of freedom in a $n \times n$ skew-symmetric matrix.*

The following concepts of retraction and vector transport have matured quite a lot over the past decades. They are still a very active topic of research.

1.3.2 Retraction

In $\mathbb{R}^n$ it is straightforward to travel from a point in a given direction. On a manifold one must ensure that the next iterate still belongs to the manifold. This is done via the concept of retraction, see [AMS08] for a complete reference. At a given point $x \in \mathcal{M}$, the retraction is a mapping from the tangent space $T_x\mathcal{M}$ to the manifold.

Definition 1.7 ***(Retraction [AMS08])***
A retraction on a manifold $\mathcal{M}$ is a smooth mapping R from the tangent bundle $T\mathcal{M}$ onto $\mathcal{M}$ with the following properties. Let R_x denote the restriction of R to $T_x\mathcal{M}$.
(i) $R_x(0_x) = x$, where 0_x denotes the zero element of $T_x\mathcal{M}$.
(ii) With the canonical identification $T_{0_x}T_x\mathcal{M} \simeq T_x\mathcal{M}, R_x$ satisfies

$$DR_x(0_x) = id_{T_x\mathcal{M}},$$

where $id_{T_x\mathcal{M}}$ denotes the identity mapping on $T_x\mathcal{M}$.

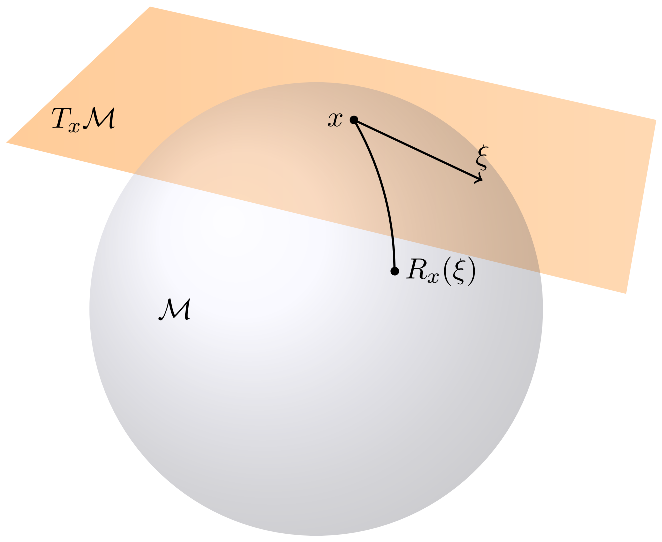


FIGURE 1.5: Retraction as illustrated in [AMS08]

Figure 1.5 illustrates the concept of retraction. When the projection of a point on the manifold is available, it gives a simple way to compute a retraction. Indeed for $\xi \in T_x\mathcal{M}$ one has a *retraction by projection* with

$$R_x(\xi) = P_{\mathcal{M}}(x + \xi).$$

For the unit sphere $\mathbb{S}^{n-1}$ we have a projection and therefore

$$R_x(\xi) = P_{\mathbb{S}^{n-1}}(x + \xi) = \frac{x + \xi}{||x + \xi||}.$$

For the orthogonal group, we recall the QR factorization of a matrix $A \in \mathbb{R}_*^{n \times n}$ as $A = QR$ with $Q \in \mathcal{O}_n$ and R upper triangular with positive diagonal entries. Then let qf denote the mapping that sends a matrix to the Q factor of its QR decomposition, $qf(A) = Q$. For $X \in \mathcal{O}_n$ and $X\Omega \in T_X\mathcal{O}_n$,

$$R_X(X\Omega) = qf(X + X\Omega)$$

is denoted as the *qf retraction* [AMS08].

For each manifold comes a handful of different retractions. One main aspect of Riemannian optimization is to compute retractions efficiently.

1.3.3 Vector transport

In Euclidean optimization, the gradients of the function at different points all belong to the same space $\mathbb{R}^n$. On a non linear manifold, the gradient belongs to the tangent space. The tangent spaces at two points are different linear spaces. As we have seen in algorithm 4 (the nonsmooth stopping criterion), we will have to compute the convex hull of gradients from different points. Therefore the gradients will lie in different tangent space. This does not make much sense unless we have a way to *transport* all the gradients to a unique tangent space. The notion of vector transport allows that. Note that this is also crucial to use a vector transport when computing differences of gradients in some algorithms such as BFGS methods. It is then also required to transport the gradients in the same tangent space before their subtractions has any meaning. Vector transports are defined in 1.8.

Definition 1.8 ***(Vector transport [AMS08])***
A vector transport on a manifold $\mathcal{M}$ is a smooth mapping

$$\mathcal{T} : T\mathcal{M} \oplus T\mathcal{M} \to T\mathcal{M} : (\eta_x, \xi_x) \mapsto \mathcal{T}_{\eta_x}(\xi_x) \in T\mathcal{M}$$

with the following properties for all $x \in \mathcal{M}$:
(i) (Associated retraction) There exists a retraction R, called the retraction associated with $\mathcal{T}$, such that the following diagram commutes

$$\begin{tikzcd}
(\eta_x, \xi_x) \arrow[r, "\mathcal{T}"] \arrow[d] & \mathcal{T}_{\eta_x}(\xi_x) \arrow[d, "\pi"] \\
\eta_x \arrow[r, "R"'] & \pi\Big(\mathcal{T}_{\eta_x}(\xi_x)\Big)
\end{tikzcd}$$

where $\pi\Big(\mathcal{T}_{\eta_x}(\xi_x)\Big)$ denotes the foot of the tangent vector $\mathcal{T}_{\eta_x}(\xi_x)$.
(ii) (Consistency) $\mathcal{T}_{0_x}\xi_x = \xi_x$ for all $\xi_x \in T_x\mathcal{M}$;
(iii) (Linearity) $\mathcal{T}_{\eta_x}(a\xi_x + b\zeta_x) = a\mathcal{T}_{\eta_x}(\xi_x) + b\mathcal{T}_{\eta_x}(\zeta_x)$.

As for retractions, there are many possible vector transport for each manifold. Vector transport also depend on the representation chosen for the tangent space. Indeed, in an embedded submanifold, an element of the tangent space can be stored as a vector from the embedding space. It can also be stored as its coordinates in a basis of the tangent space. This is presented in [HAG16] as the *intrinsic approach.* The formulation of the vector transport will change accordingly.

The *parallel transport* is a tool in differential geometry. It is defined in [AMS08]. Historically this was the first attempt to transport vectors from different tangent spaces. Unfortunately, in general, it is difficult to compute. Which is why vector transports were defined as a generalization of parallel transport, allowing for transportations that are easier to compute. Nevertheless, there exists a closed expression for the parallel transport on the sphere.

Example 1.3 ***(Parallel transport on the sphere, [AMS08], eq. (8.4))***
Let $t \mapsto x(t)$ be a geodesic on $\mathbb{S}^{n-1}$. Let u denote $\dfrac{1}{\|\dot{x}(0)\|}\dot{x}(0)$. The parallel translation of the vector $\xi(0) \in T_{x(0)}\mathbb{S}^{n-1}$ along the geodesic is given by

$$\xi(t) = -x(0)\sin(\|\dot{x}(0)\|t)u^T\xi(0) + u\cos\Big(\|\dot{x}(0)\|t\Big)u^T\xi(0) + (I - uu^T)\xi(0).$$

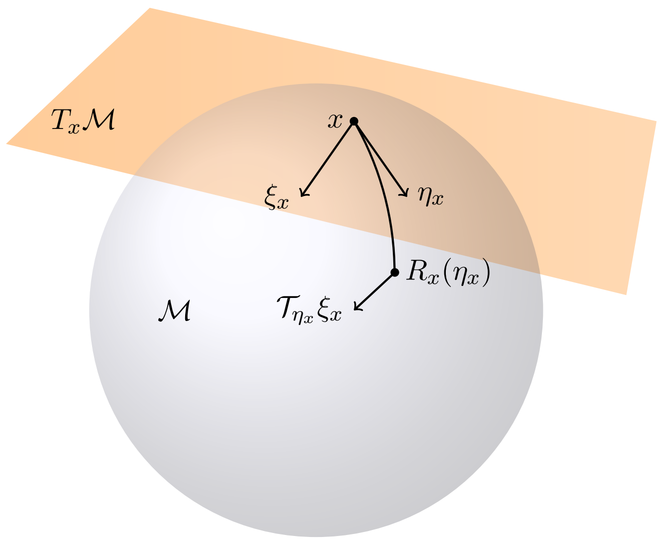


FIGURE 1.6: Vector transport based on [AMS08]

The parallel translation gives a vector transport on the sphere.

To compute a vector transport on the orthogonal group, we shall use the recently defined notion of vector transport by parallelization, [WH14] section 2.3.1.

For the manifold O_n as a submanifold of $\mathbb{R}^{n\times n}$, the dimension of the manifold is $d = n(n-1)/2$. An open subset $\mathcal{U}$ of O_n is called *parallelizable* if it admits a smooth field of tangent bases, i.e., a smooth function $B : \mathcal{U} \to \mathbb{R}^{n^2\times d} : z \mapsto B_z$ where B_z is a basis of $T_z\mathcal{M}$. Given such basis, the vector transport by parallelization from $T_x\mathcal{M}$ to $T_y\mathcal{M}$ is defined as

$$\mathcal{T} = B_y B_x^b$$

where $v^\flat$ represents the flat of v, i.e., $v^\flat : T_x\mathcal{M} \to \mathbb{R} : w \to \langle v, w\rangle$. For $\xi \in T_x\mathcal{M}$, both ξ and $\mathcal{T}_{x\to y}(\xi)$ have the same coordinates in their respectives bases, B_x and B_y. So if an intrinsic representation is used and the tangent vector is stored with its coordinates in a basis of the tangent space, the vector transport by parallelization is the identity. This is a great advantage of the intrinsic approach. It removes the computational cost of vector transport from the algorithms.

It has the disadvantage that a basis of the tangent space must be available at every point on the manifold. And it must be possible to go from a vector to its coordinates and vice-versa. Therefore, there are some manifold for which the intrinsic approach seems to be more suited. Mostly, the two following conditions are good indications: the dimension of the manifold d is notably smaller than n and there is a straightforward way to use basis coordinates on each tangent space.

Chapter 2

Algorithms

In this third chapter, we leave the background and preliminary notions behind to dive into the description of the algorithms that were studied. Starting with the core method: Riemannian BFGS (RBFGS). There will follow a limited memory version and some variants. We also describe another keystone algorithm in the field, the Riemannian gradient sampling (RGS), that will be used for comparison.
Let us state the formal context of these algorithms. Let $\mathcal{M}$ be a Riemannian manifold and the function $f : \mathcal{M} \longrightarrow \mathbb{R}$ locally Lipschitz on $\mathcal{M}$. We are looking for a local minimizer of f, also know as a Clarke stationary point. The function is locally Lipschitz which implies the following

Hypothesis 2.1 *The exists a subset $D \subset \mathcal{M}$, dense in $\mathcal{M}$. Such that f is smooth on D.*

2.1 Nonsmooth Riemannian BFGS

The Riemanian adaptation of BFGS follows from the Euclidean version of Lewis and Hoverton in section 1.2. This section is mainly about adapting the inexact line search (algorithm 3) and the stopping criterion (algorithm 4) to the Riemannian setting.
As for the Euclidean case, the objective f is locally Lipschitz and differentiable almost everywhere on the manifold. We assume that the iterations are well defined, even though in theory, it could break down if we reach a nondifferentiable point.
It is important to realize that the method presented is rather new and very few attention has been dedicated to a Riemannian nonsmooth BFGS so far. Even though [LO13] gives insight that BFGS works well for nonsmooth functions, there is no guarantee this will transpose to Riemannian optimization. Another state of the art result comes from [HGA15] where convergence of a smooth Riemannian BFGS method is proven. They require a twice differentiable cost, a convexity assumption and some assumptions for the retraction and vector transport. These assumptions are not met for the nonsmooth applications that we are interested in.

2.1.1 Non smooth Line Search

This is an adaptation of section 1.2.1. The line search function $h(t) = f(R_x(tp)) - f(x)$ is now defined with p in the tangent space $T_x\mathcal{M}$ and R_x is a retraction mapping at x. The step size is $t \in \mathbb{R}$. For our line search criterion, define $s = \limsup_{t \to 0} \dfrac{h(t)}{t}$ and we make the assumption that $s < 0$ which means that we have a descent direction. The conditions based on [Hua13] are the following. For $0 < c_1 < c_2 < 1$,

$$\begin{aligned} A(t) :&\quad h(t) < c_1 st \\ W(t) :&\quad \text{h is differentiable at t with } h'(t) > c_2 s \end{aligned} \tag{2.1}$$

The line search procedure used works as in algorithm 5. This will give an inexact min-

Algorithm 5 Non smooth line search [Hua13]

```
1:  Input: $x_k$, $k \geq 0$
2:  $\gamma = 0$
3:  $\beta = +\infty$
4:  $\alpha_k = 1$ or any heuristic for the initial step length
5:  while 1 do
6:      if $A(\alpha_k)$ fails then $\beta = \alpha_k$
7:      else if $W(\alpha_k)$ fails then $\gamma = \alpha_k$
8:      else break
9:      if $\beta < +\infty$ then $\alpha_k = (\gamma + \beta)/2$
10:     else $\alpha_k = 2\gamma$
11: Output: Next iterate $x_{k+1} = R_{x_k}(\alpha_k p_k)$
```

imization. As we know this is important to avoid reaching a point of non differentiability. Given a retraction, there are not many degrees of freedom to adapt the Euclidean line search. One uses the retraction to travel along the manifold in a given direction.

2.1.2 Non smooth stopping criterion

We are now looking at ways to adapt the nonsmooth stopping criterion from Lewis and Overton of section 1.2.2 to Riemannian manifolds.
We can still store a collection of gradients at the iterates reached. But it only makes sense to compute the convex hull of vectors defined on the same tangent space. So, at each step, the vectors must all be transported to the current tangent space before computing the convex hull. Let $\mathcal{T}_{R^{-1}_{x_i}(x_j)}$ be the inverse vector transport, associated to the retraction R, from the tangent space at x_i to x_j. And then note $\text{grad} f_i^{(j)} = \mathcal{T}_{R^{-1}_{x_i}(x_j)} \text{grad} f(x_i)$, the gradient at x_i transported to the tangent space at x_j. We follow the convention that the upper index always represents the tangent space in which the vector has been transported.

The Riemannian version of the algorithm 4 is the following

Algorithm 6 Nonsmooth stopping criterion [Hua13]

```
1: Input: $J$, $\varepsilon_x$, $\varepsilon_d$
2: Iteration $k$
3: if $\text{dist}(x_k, x_{k-1}) > \varepsilon_x$ then $j_k = 1, G_k = \{\text{grad} f_k\}$
4: if $\text{dist}(x_k, x_{k-1}) \leq \varepsilon_x$, and $j_{k-1} < J$ then $j_k = 1, G_k = \{\text{grad } f^{(k)}_{k-j_k+1}, \dots, \text{grad} f^{(k)}_k\}$
5: if $\text{dist}(x_k, x_{k-1}) \leq \varepsilon_x$, and $j_{k-1} = J$ then $j_k = J, G_k = \{\text{grad } f^{(k)}_{k-J+1}, \dots, \text{ grad } f^{(k)}_k\}$
   Where $\text{grad} f_i^{(j)}$ is the gradient at $x_i$ transported in $T_{x_j}\mathcal{M}$.
6: $d_k = \text{argmin}\{\|d\|^2_{x_k} : d \in \text{conv} G_k\}$
7: if $\|d_k\|_{x_k} < \varepsilon_d$ then STOP
8: else Compute $x_{k+1}$
```

This approach requires to transport the gradients stored in G_{k-1} to the tangent space of the new point x_k at each step. This is an expensive operation because we might perform up to $J-1$ vector transport at each step. In addition, finding the smallest vector in the convex hull is a convex quadratic problem of dimension $n + j_k$.

The appropriate number of gradients to store, J, will be discussed in more details during the experiments. However, we have mentioned previously that it has to be larger than n. So we are bound to solve a quadratic problem of about $\mathcal{O}(n)$ variables at each iteration to check the stopping criterion of algorithm 6. In the following section we analyse how expensive it is to solve such problem. We can already say that this will cause problems for medium and large scale problems. But we are not ready to give up just yet on having an optimality certificate for large scale problems. We have to use other approximations as stopping criterion. For a tolerance $\varepsilon > 0$, some efficient alternatives are

Measure of displacement	$\operatorname{dist}(x_{k+1}, x_k) < \varepsilon$
Relative function variation	$\dfrac{\lvert f(x_{k+1}) - f(x_k)\rvert}{\lvert f(x_k)\rvert + 1} < \varepsilon$

Other variations of the previous criteria exist. Note that using $\dfrac{\lvert f(x_{k+1}) - f(x_k)\rvert}{\lvert f(x_k)\rvert} < \varepsilon$ for the relative function variation is not very wise. If the function values tend towards zero at the minimum, this quotient will not be stable. The proposed criterion works well in practice. We will informally class these criteria as *approximate stopping criterion.* In the sense that when such a criterion is reached, there is no guaranty that we are near a Clarke stationary point.

Once such an approximated criterion is triggered, we can compute the subdifferential if a certificate of optimality is needed. Note that are never interested in the entire set ∂f. The subdifferential is usually approached by a finite number of nearby gradients. If the smallest vector in the convex hull is close enough to zero, we have indeed reached a Clarke stationary point. To approach the subdifferential, the notion of ε- subdifferential has been introduced [GH15][BKM14]. The idea is to approach the subdifferential given some fixed tolerance $\varepsilon > 0$.

Definition 2.1 ***(ε-subdifferential based on [BKM14])***
Let $f : \mathcal{M} \to \mathbb{R}$ be a locally Lipschitz function on a Riemannian manifold $\mathcal{M}$ with vector transport $\mathcal{T}$, and let $\varepsilon > 0$. Then the ε-subdifferential of f at x is te set

$$\partial_\varepsilon f(x) = conv\{\mathcal{T}_{x\leftarrow y}(\partial f(y)) : y \in clB(x, \varepsilon)\}.$$

We are going to look at two ways to compute the ε- subdifferential. The first approach is problem related. For some cost functions, there is an analytical procedure to compute the subdifferential. This usually happens when the nonsmooth part of the objective is written as a maximum of different smooth functions. Then, the subdifferential is the convex hull of the gradients of the smooth functions that are active in the *max* function. When several functions are active in the *max*, there is not a unique gradient. Since we do not expect to reach such points of nondifferentiability, we use a tolerance ε. Any function that is active in the max up to will be considered active and its gradient will enter the ε-subdifferential.

Another approach is a gradient sampling test based on the definition of $\partial_\varepsilon f$. The procedure is detailled in algorithm 7. It consists in evaluating gradients at neighbouring points in a ball of radius ε. The gradients are then transported to the current tangent space and the convex hull is an ε-subdifferential. The main advantage is that this approach is easy to formulate and can always be used. It does not require to write an analytic formulation of the subdifferential which is difficult for some problems. This approach requires to sample $J \geq n + 1$ gradients. It is not so easy to find a good value for J, this will be discussed in the experiments.

Algorithm 7 Stopping criterion based on gradient sampling for ε − subdifferential

```
1: Input: $J$, $\varepsilon$, $\varepsilon_x$,$\varepsilon_d$,$x_k$
2: if $\|x_k - x_{k-1}\| \leq \varepsilon_x$ then
3:     Choose $J$ points $\{x_k^i\}_{i=1}^m$ independently and uniformly in $B(x_k, \varepsilon)$.
4:     Set
           $$\partial_\varepsilon f \doteq \text{conv}\{\text{grad}f(x_k), \mathcal{T}_{x_k \leftarrow x_k^1}\Big(\text{grad}f(x_k^1)\Big), \dots, \mathcal{T}_{x_k \leftarrow x_k^m}\Big(\text{grad}f(x_k^m)\Big)\}$$
       and compute
           $$w_k = \text{argmin}\{\|w\|^2 : w \in G_k\}.$$
5:     if $\|w\| < \varepsilon$ then STOP
6: else Compute $x_{k+1}$
```

To conclude this section, here is the full statement of the nonsmooth RBFGS in algorithm 8, based on [Hua13][M.S13]. AS this is an adaptation of the work in [LO13] to Riemannian manifolds, there are no guarantees that the methods generates descent directions or converges to a local Clarke stationary point.

Algorithm 8 Nonsmooth RBFGS based on [Hua13][M.S13]

```
1: Input: A Riemannian manifold $\mathcal{M}$ with metric $\langle \cdot, \cdot \rangle$, a vector transport $\mathcal{T}$ with associated retraction $R$, a real valued cost function $f : \mathcal{M} \to \mathbb{R}$, an initial iterate $x_0 \in \mathcal{M} \cap D$; an initial approximation of the Hessian $B_0$. Set $k = 0$, $G_0 = \varnothing$ an empty set.
2: while Stopping criterion not met do
3:     Solve $B_k p_k = -\text{grad}f(x_k)$ for $p_k \in T_{x_k}\mathcal{M}$.
4:     Obtain the step length $\alpha$ trough the inexact line search in algorithm 5 to satisfy the Wolfe conditions.
5:     Set
           $$\begin{aligned} s_k &= \mathcal{T}_{\alpha p_k}(\alpha p_k) \\ y_k &= \text{grad}f(x_{k+1}) - \mathcal{T}_{\alpha p_k}\Big(\text{grad}f(x_k)\Big) \\ B_{k+1}\eta &= B_k^{(k+1)}\eta + \frac{< y_k, \eta >}{< y_k, s_k >} y_k - \frac{< s_k, B_k^{(k+1)}\eta >}{< s_k, B_k^{(k+1)} s_k >} B_k^{(k+1)} s_k \\ \text{with } B_k^{(k+1)} &= \mathcal{T}_{\alpha p_k} \circ B_k \circ \mathcal{T}_{\alpha p_k}^{-1}, \text{ for all } \eta \in T_{x_{k+1}}\mathcal{M}. \end{aligned} \quad (2.2)$$
       where $B_{k+1} : T_{x_{k+1}}\mathcal{M} \to T_{x_{k+1}}\mathcal{M}$ is a liner operator on the tangent space at $x_{k+1}$.
6:     Update $G_k$ according to algorithm 6 and compute the smallest norm vector in convex hull of $G_k$ to check the stopping criterion.
7:     Set $k \leftarrow k + 1$.
```

2.1.3 Smallest norm in convex hull problem

As we have seen, the stopping criterion algorithm requires to compute the element with the smallest norm in the convex hull of a finite set of gradients. This appears in the nonsmooth stopping criterion but also in other minimization algorithms that are discussed later. The goal of this section is to highlight the fact that solving a problem such as (1.9) is very expensive and has a great dependence on the dimension.

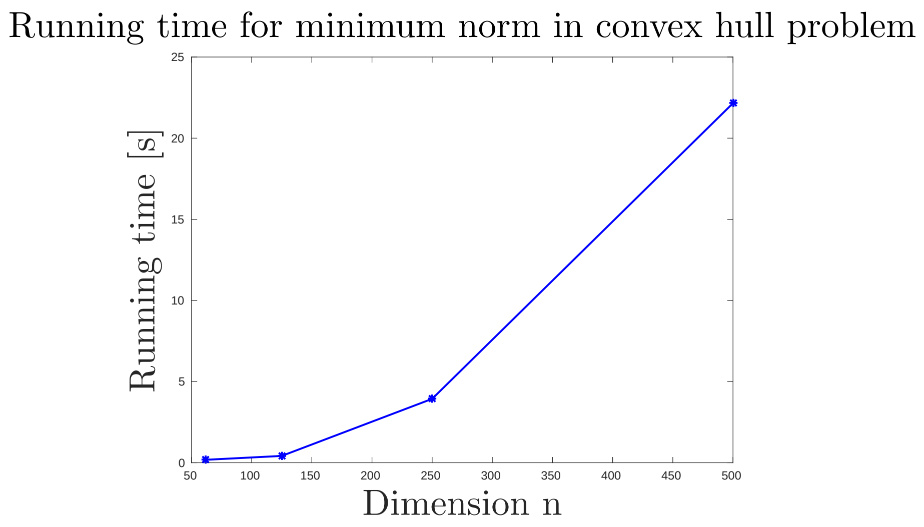


FIGURE 2.1: Running time for the minimum norm problem in the convex hull with interior point methods.

For a problem of size n, we computed the smallest vector in the convex hull of n vectors in $\mathbb{R}^n$. The process was repeated 10 times for each dimension to take the average running time. Figure 2.1, shows this average running time for increasing dimensions up to $n = 500$. We used the Matlab `quadprog` function which uses interior point methods. It already takes about 20 seconds to solve the problem in dimension 500. Even though there might be libraries better optimized for this problem, the point is that the computation time behaves as $\mathcal{O}(n^2)$. And this gives us indication that we will not be able to afford to use the nonsmooth stopping criterion for very large dimensions.

2.2 Nonsmooth limited memory Riemannian BFGS

In Euclidean optimization, the limited memory version of BFGS, namely LBFGS, is popular due to its efficiency, especially for large dimensions problems[AS10]. With this in mind, it only make sense to consider a limited memory version in the Riemannian framework. We shall denote LRBFGS the adaptation of the previously described RBFGS, with the addition of a memory parameter. As in the Euclidean setting, the idea of LRBFGS is to only store the recent updates for the Hessian approximation. Then, only the m most recent s_k and y_k are transported to the new tangent space rather than the entire matrix H_k. The LRBFGS algorithm based on [HGA15] is listed as algorithm 9.

Algorithm 9 Nonsmooth Limited memory Riemannian BFGS [HGA15]

```
1: Input: A Riemannian manifold $\mathcal{M}$ with metric $< \cdot, \cdot >$, a vector transport $\mathcal{T}$ with associated retraction $R$, a real valued cost function $f : \mathcal{M} \to \mathbb{R}$, an initial iterate $x_0 \in \mathcal{M} \cap D$, the memory parameter $m$ a positive integer;
2: Set $k = 0$, $\gamma_0 = 1$.
3: while Stopping criterion is not met do
4:     $H_k^0 = \gamma_k \mathrm{id}$. Obtain $\eta_k \in T_{x_k}\mathcal{M}$ by the following procedure:
5:     $q \leftarrow \mathrm{grad} f(x_k)$
6:     for $i = k-1, k-2, \dots, k-m$ do
7:         $\xi_i \leftarrow \rho_i \langle s_i^{(k)}, q \rangle$;
8:         $q \leftarrow q - \xi_i y_i^{(k)}$;
9:     $r \leftarrow H_k^0 q$;
10:    for $i = k-m, k-m, +1, \dots, k-1$ do
11:        $\omega \leftarrow \rho_i \langle y_i^{(k)}, r \rangle$;
12:        $r \leftarrow r + s_i^{(k)}(\xi_i - \omega)$;
13:    Set $p_k = -r$
14:    Obtain the step length $\alpha$ trough the inexact line search in algorithm 5 to satisfy the Wolfe conditions.
15:    Set $x_{k+1} = R_{x_k}(\alpha_k p_k)$;
16:    Define
```

$$\begin{aligned} s_k^{(k+1)} &= \mathcal{T}_{\alpha_k p_k}(\alpha_k p_k) \\ y_k^{(k+1)} &= \mathrm{grad} f(x_{k+1})/\beta_k - \mathcal{T}_{\alpha_k p_k}\Big(\mathrm{grad} f(x_k)\Big) \\ \rho_k &= \frac{1}{< s_k^{(k+1)}, y_k^{(k+1)} >} \\ \gamma_{k+1} &= \frac{< s_k^{(k+1)}, y_k^{(k+1)} >}{\|y_k^{(k+1)}\|^2} \\ \beta_k &= \frac{\|\alpha_k p_k\|}{\|\mathcal{T}_{\alpha_k p_k}(\alpha_k p_k)\|} \end{aligned} \tag{2.3}$$

```
17:    Add $s_k^{(k+1)}, y_k^{(k+1)}$ and $\rho_k$ into storage and if $k > m$ then discard vector pair $\{s_{k-l-1}^{(k)}, y_{k-l-1}^{(k)}\}$ and scalar $\rho_{k-l-1}$ from storage;
18:    Transport the previously stored information $s_{k-m}^{(k)}, s_{k-m+1}^{(k)}, \dots, s_{k-1}^{(k)}$ and $y_{k-m}^{(k)}, y_{k-m+1}^{(k)}, \dots, y_{k-1}^{(k)}$ from $T_{x_k}\mathcal{M}$ to $T_{x_{k+1}}\mathcal{M}$ by $\mathcal{T}_{\alpha_k p_k}$. Which gives $s_{k-m}^{(k+1)}, s_{k-m+1}^{(k+1)}, \dots, s_{k-1}^{(k+1)}$ and $y_{k-m}^{(k+1)}, y_{k-m+1}^{(k+1)}, \dots, y_{k-1}^{(k+1)}$;
19:    Update $G_k$ according to algorithm 6 and compute the smallest norm vector in convex hull of $G_k$ to check the stopping criterion.
20:    Set $k \leftarrow k + 1$.
```

2.3 Subgradient Riemanian methods

So far we have worked under the assumption that the methods would never encounter a point where the function is nondifferentiable. And we assumed that the gradient can be computed everywhere. This might yield satisfying results in practice. However, it seems to give a very difficult road to any convergence results [LO13]. More recent work was initiated by S. Hosseini to tackle this kind of problems with an entirely nonsmooth approach. Moreover, it approximates the subdifferential to compute a subgradient. And then uses it to define a quasi-Newton scheme. What we call the *subgradient RBFGS* algorithm is defined in [AHHY16].

2.3.1 Subgradient RBFGS

We shall give the main ideas behind this method. We begin with the definition of a *subgradient-oriented sequence.*

Definition 2.2 ***(Subgradient-oriented descent sequence [AHHY16])***
A sequence $\{p_k\}$ of descent directions is called subgradient-oriented if there exist a sequence of subgradients $\{g_k\}$ and a sequence of symmetric linear maps $\{P_k : T_{x_k}\mathcal{M} \to T_{x_k}\mathcal{M}\}$ satisfying

$$0 < \lambda \leq \lambda_{min}(P_k) \leq \lambda_{max}(P_k) \leq \Lambda < \infty, \tag{2.4}$$

for $0 < \lambda < \Lambda < \infty$ and all $k \in \mathbb{N}$, such that $p_k = -P_k g_k$.

Note that $\lambda_{min}(P_k)$ and $\lambda_{max}(P_k)$ denote respectively the smallest and largest eigenvalues of P_k. We move on with the idea that many smooth optimization algorithm have a search direction of the form $p_k = -P_k \text{grad} f(x_k)$, where P_k is a symmetric positive definite matrix. So it makes sense to use an approximation of $\partial f(x_k)$ to define the subgradients g_k from Definition 2.2. And then produce a subgradient-oriented sequence. To compute the subgradients g_k, we make use of the ε-subdifferential, defined in definition 2.1. At each step k, a set W_k will store nearby gradients with the goal to approximate $\partial_\varepsilon f(x)$ by $\text{conv} W_k$. Let P a symmetric positive definite matrix. The associated norm is defined as $\|\xi\|_P = \langle \xi, P\xi \rangle^{1/2}$. Let

$$g_k := \text{argmin}_{v \in \text{conv} W_k} \|v\|_P. \tag{2.5}$$

If the condition,

$$f\Big(R_x(\frac{\varepsilon p_k}{\|p_k\|})\Big) - f(x) \leq \frac{-c\varepsilon \|g_k\|_P^2}{\|p_k\|},\ c \in]0,1[\tag{2.6}$$

holds, then it is said that $\text{conv}\{W_k\}$ is a good enough approximation of $\partial_\varepsilon f(x)$. Otherwise, another vector must be added to W_k and [AHHY16] shows how this can be done. It is important to notice that if a nondifferentiable point is reached, the algorithm continues by sampling another point and this does not prevent to approximate the subdifferential and compute a subgradient. Whereas the RBFGS algorithm would break down if we reached a nondifferentiable point.

This allows to prove the following result. The subgradient-oriented directions are descent directions. It is important to realize that this comes as a great advancement since there are no similar results for the RBFGS algorithm.

Theorem 2.1 ***[AHHY16]*** *Assume that $f : \mathcal{M} \to \mathbb{R}$ is a locally Lipschitz function on a Riemannian manifold $\mathcal{M}$ and*

$$g_k = argmin_{\xi \in \partial_\varepsilon f(x)} \|\xi\|_{P_k}.$$

Then $p_k = -P_k g_k$ is a descent direction.

A line search is then defined in [AHHY16]. It is based on the same weak Wolfe condition as equation (2.1). It is proved that the line search algorithm finds an acceptable step length in a finite number of trials. The minimization scheme consists in using the BFGS update formula for the matrices P_k. Convergence of the iterative process to a Clarke stationary point can then be proved. We see a few complications to apply this convergence result to test problems. It seems difficult to guarantee the bound on the spectre of the operators P_k in definition 2.2.

2.3.2 Subgradient LRBFGS

This method is the limited-memory variant of the previous section. It consists in updating the matrices P_k with the LRBFGS update formula.

Chapter 3

Test problems

In this third chapter we describe the two test problems that were used to evaluate the performances of the algorithms. Both problems, the orthogonal bounding box (OBB) and the sparse principal component analysis (PCA), have real nonsmooth cost defined on a Riemannian manifold. We explain the questions of interest that arise in the applications, how the situation is expressed as a Riemannian optimization problem and what makes the objective partly smooth. Finally we try to see if an expression is available to approximate the subdifferential.

3.1 Sparse principal component analysis

The sparse principal component analysis problem is present in many field of science where high dimensional data sets are encountered. Consider a matrix $A \in \mathbb{R}^{N\times n}$, where each of the n columns represents a variable of interest, and each row is an sample from a population of size N. One assumes that each column of A has zero mean. The covariance matrix is then defined as $\Sigma = A^TA \in \mathbb{R}^{n\times n}$. In applications, we would like to reduce the dimension of the data but still be able to explain the variance in the data as much as possible . To find the principal components of Σ, one usually maximizes the Rayleigh quotient $\dfrac{x^T\Sigma x}{x^Tx}$, $x \neq 0$. This is equivalent to a minimization over the unit sphere:

$$\min_{x\in\mathbb{S}^{n-1}} -x^T\Sigma x. \tag{3.1}$$

This gives the eigenvector with the largest eigenvalue. In dimension reduction applications, we look for an approximation of the eigenvector that is sparse to some extent. Applications of the sparse principal component analysis (PCA) appear in many fields such as biology. Where this helps detect the few variables (or genes) that accounts for most of the variance in the data of a population [JNRS08].

To get a sparse approximation of the principal component, we introduce a penalty in the cost function. The ideal penalty would be the cardinality of the vector, the number of non zero elements. This is also know as the ℓ_0 norm of x, written $\|x\|_0$.

$$\min_{x\in\mathbb{S}^{n-1}} -x^T\Sigma x + \mu\|x\|_0. \tag{3.2}$$

The tuning coefficient $\mu > 0$ gives more or less importance to the penalty. This is for the user to decide whether he cares more about data analysis or sparsity of the solution.

The issue with this formulation is that the objective function would not even be continuous, let alone differentiable. So we turn to a less forcing penalty that is the ℓ_1 norm, $\|x\|_1 = \sum_{i=1}^n |x|_i$. This penalty reduces the cardinality of the principal component. Our final formulation for the Sparse PCA problem is then

$$\min_{x\in\mathbb{S}^{n-1}} f(x) = -x^TAx + \mu\|x\|_1 \tag{3.3}$$

Note that this formulation only allows to solve for the first principal component. Some block formulations cover the case of several principal components. We will focus on the simple case in this text. The Euclidean gradient of the objective is $\mathrm{grad}\tilde{f}(x) = -2Ax + \mu\ \mathrm{sign}(x)$. The Riemannian gradient is the projection on the tangent plane $T_x\mathbb{S}^{n-1}$,

$$\mathrm{grad}f(x) = (I - xx^T)\mathrm{grad}\tilde{f}(x) = (I - xx^T)(-2Ax + \mu\ \mathrm{sign}(x)).$$

This cost function is partly smooth. The problems arise along the axis where the ℓ_1 norm is not differentiable. We can rewrite the nonsmooth part to find an expression for the subdifferential. First we recall a results about the subdifferential. Let $f_1, \dots, f_k$ be smooth functions, and $f = \max\limits_{i=1\dots k} f_i$. Then

$$\partial f(x) = \mathrm{conv}\{\mathrm{grad}f_i(x) | f_i(x) = f(x)\}.$$

This is a standard result, the subdifferential of a max function is the convex hull of the gradients of all the functions f_i that are active in the max. For the purpose of illustration let us take $n = 2$. Then

$$\begin{aligned}\|x\|_1 &= |x_1| + |x_2| \\ &= \max\{x_1, -x_1\} + \max\{x_2, -x_2\} \\ &= \max\{\underbrace{x_1 + x_2}_{f_1}, \underbrace{x_1 - x_2}_{f_2}, \underbrace{-x_1 + x_2}_{f_3}, \underbrace{-x_1 - x_2}_{f_4}\}\end{aligned}$$

At $x = (1, 0)$, $\|x\|_1 = 1$ and there are two out of the four functions that are active in the max, f_1 and f_2. The subdifferential of f would be

$$\partial f(x) = (I - xx^T)(-2Ax + \mu\ \mathrm{conv}\{\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\-1\end{pmatrix}\}).$$

In practice, the ε-subdifferential is computed with a tolerance. If a function in the max is equal to $\|x\|_1$ up to some fixed $\varepsilon > 0$, the function is said to be active. This expression for the subdifferential is expected to work very well and provide a good check for optimality. It is described in algorithm 10. The only real issue here is the number of active functions. If there are N variables $x_i = 0$, 2^N subgradients enter the convex hull. We found that this quickly becomes large since the sparse PCA problems is usually considered in very large dimensions.

Algorithm 10 ε-subdifferential stopping criterion for the sparse PCA problem

```
1: Input: $x_k$, $\varepsilon_x$, $\varepsilon_f$, $\varepsilon$
2: if $\mathrm{dist}(x_k, x_{k-1}) \le \varepsilon_x$ then
3:     Let $G = \emptyset$ an empty set and $s_i$ for $i = 1, \dots, 2^n$ be the $2^n$ vectors filled with 1 and
       $-1$ for all possible sign combinations.
4:     for $i = 1 : 2^n$ do
5:         if $|s_i^T x_k - \|x_k\|_1| < \varepsilon_f$ then
6:             $G = G \cup \{s_i\}$
7:     Set the $\varepsilon$- subdifferential at $x_k$, $\partial_\varepsilon f(x_k) = (I_n - x_k x_k^T)(-2Ax_k + \mu\ \mathrm{Conv}\ G)$
8:     Find
           $w = \mathrm{argmin}\{\|d\|_{x_k}^2 : d \in \partial_\varepsilon f(x_k)$
9:     if $\|w\|_{x_k} < \varepsilon$ then STOP
10: else Compute $x_{k+1}$
```

The search space for this problem is the sphere $\mathbb{S}^{n-1}$. The sphere is a Riemannian manifold as detailed in section 1.3. In our experiments, we used the projection as a retraction and the parallel vector transport as in Example 1.3. We did not use the concept on intrinsic representation defined in [WH14].

3.2 Orthogonal bounding box problem

The orthogonal bounding box problem (OBB) comes as follows. Given a set of N points belonging to $\mathbb{R}^n$, the goal is to find the oriented parallelepiped box of minimum volume that contains all the points. This situation arises in several practical applications and mainly intersection problems. Deciding whether two objects intersect is a problem present in many applications. Which must often be answered as fast as possible. It is cheaper to detect intersections between bounding volumes than going through the struggle of computing the convex hull. The box is often chosen as a bounding volume because it makes the intersection test very simple [BA10].
For each possible orientation, there is a bounding box and a corresponding volume. We have to find the orientation that gives the smallest bounding box. The case $n = 2$ of the plane or $n = 3$ the ambient space are the most natural to visualize. In two dimensions, the variable is simply the angle that makes the rectangular box rotate. For each orientation between 0 and 90 degrees, there is an area for the bounding box. We look for the angle that minimizes the area of the box. In three dimensions, there are 3 degrees of freedom for the orientation of the box.
The idea is to formulate the problem as an optimization over the orthogonal group presented in section 1.3

$$O_n = \{R \in \mathbb{R}^{n\times n} : R^T R = I\}.$$

Let $A \in \mathbb{R}^{n\times N}$ be the N data points. For any candidate $R \in O_n$, we define $f_A(R)$ as the volume of the box oriented according to the rotation R. This defines a mapping $f_A : O_n \to \mathbb{R}$. The function is defined on the manifold O_n and has real values. So the minimization is well defined as a Riemannian optimization problem. The metric comes from the embedding space $\mathbb{R}^{n\times n}$. For clarity the subscript A will be omitted when no ambiguity is possible.

The next step is to compute the volume of the bounding box, $f(R)$. We apply the rotation R to the set of points A. Define $A' = RA$, then the volume $f(R)$ is the volume of the axis-aligned bounding box (AABB) of the points A'. This is naturally the product of the widths along each axis. This is written

$$f(R) = V_{AABB}(RA) = (a'_{1,\max} - a'_{1,\min})(a'_{2,\max} - a'_{2,\min}) \dots (a'_{n,\max} - a'_{n,\min})$$

where $a'_{i,\max}$ is the largest element in the i^{th} row of A', $a'_{i,\min}$ being the smallest.

The figure 3.1, shows the graph of the cost function for an example with 5 points in the plane. The bounding box is shown for a rotation of 51 degrees. This is the optimal box as we can see that it minimizes the cost. We notice that the cost is nonsmooth.

The next step is to find a formula for the gradient. Let $\tilde{f}$ define the extension of f to the embedding space $\mathbb{R}^{n\times n}$. We first compute the Euclidean gradient,

$$\mathrm{grad}\tilde{f}(R) = TA^T \in \mathbb{R}^{n\times n}, \tag{3.4}$$

where $T \in \mathbb{R}^{d\times N}$ and

$$i\text{-th row of } T = \begin{cases} \dfrac{\omega}{(a'_{i,\max} - a'_{i,\min})}, & \text{the column of } a'_{i,\max}; \\ -\dfrac{\omega}{(a'_{i,\max} - a'_{i,\min})}, & \text{the column of } a'_{i,\min}; \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

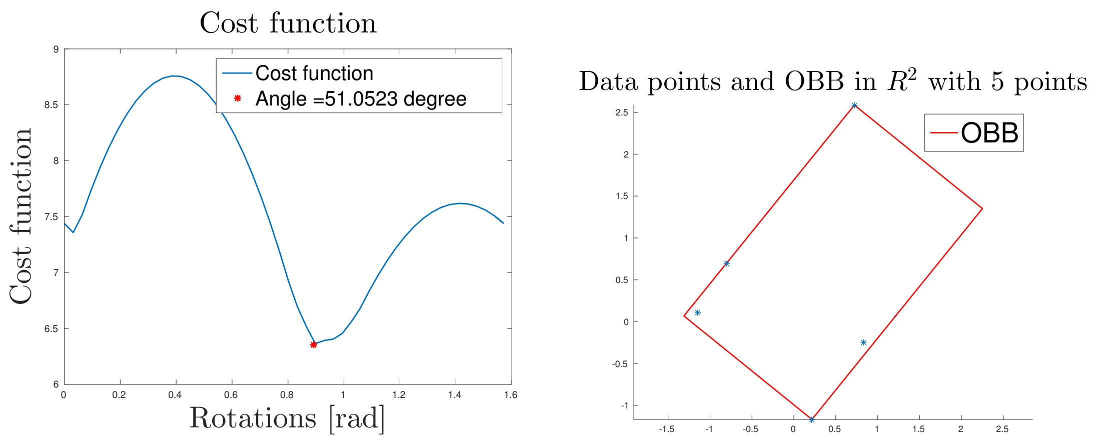


FIGURE 3.1: Illustration of the optimal bounding box problem in 2 dimensions

for $i = 1, \ldots, n$ and $\omega = f(R) = (a'_{1,\max} - a'_{1,\min})(a'_{2,\max} - a'_{2,\min}) \ldots (a'_{n,\max} - a'_{n,\min})$.

The Riemannian gradient is obtained as the projection of the Euclidean gradient on the tangent plane $T_R O_n$,

$$\operatorname{grad} f(R) = P_R(\operatorname{grad} \tilde{f}(R)) = \operatorname{grad} \tilde{f}(R) - R\operatorname{sym}(R^T \operatorname{grad} \tilde{f}(R)) = TA^T - R\operatorname{sym}(R^T TA^T),$$

where $\operatorname{sym}(M) = \frac{M^T+M}{2}$ is the symmetric part of a matrix M.

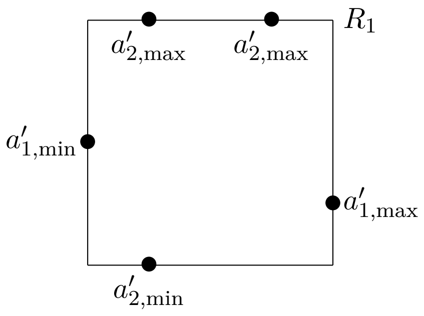
This gives an explicit formula for the gradient of the cost function. Unfortunately as we saw on figure 3.1, the function is nonsmooth over the manifold. The gradient is well defined only when the $a'_{i,\max}$ and $a'_{i,\min}$ are all unique, ie. there is only one data point on each border of the box. On figure 3.2(a), we see that the orientation given by the rotation R_1 leads to two points achieving $a'_{2,\max}$ on the upper end of the box. This gives two possible versions of the matrix T of formula 3.5. It should seem clear that the function is nonsmooth over a set of measure zero. Indeed, only a few rotations yield a problem to define the gradient. In figure 3.2(b), a rotation R_2 that is close to R_1 is a smooth point in the domain of the cost function f. Because the box was tilted a bit and it is enough to remove one of the points from the upper end of the box.

We can now try to find a representation of the subdifferential at nondifferentiable points. Simply put, for partly smooth functions the approached subdifferential is the convex hull of the gradients at neighbouring points. So we have to find the gradients at neighbouring rotations and take the convex hull. In our example of figure 3.2, we have

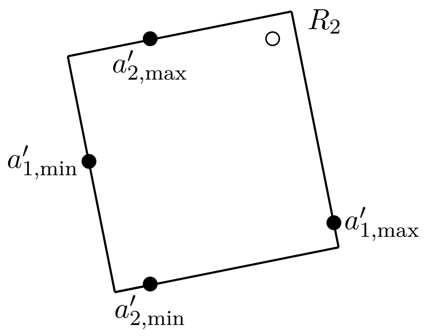
$$\partial f(R_1) = \operatorname{conv}\{\operatorname{grad} f(R_2), \operatorname{grad} f(R_3)\}.$$

The subdifferential is well approximated by this convex hull. To generalize, we have to take all the possible combinations that form neighbouring gradients. For example, if one edge of the box has two points on it and another has three, there will be six gradients in the convex hull. In three dimensions, the number of vectors in $\partial f(x)$ is still small when the data points are randomly located. Because the odds of having several points lined up is small. As the dimension increases, the number of possible subgradients seems to become very large.

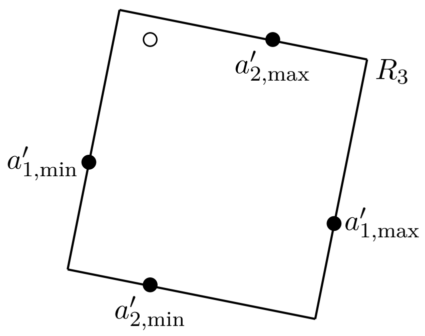
For our experiments on the orthogonal group as a manifold, we used the qf-retraction and vector transport by parallelization as described in section 1.3. We used an intrinsic representation for the tangent vectors, so that the vector transport is the identity mapping.



(a) f is nonsmooth at R_1



(b) f is smooth at R_2



(c) f is smooth at R_3

FIGURE 3.2: The objective function for the OBB problem is partly smooth

Chapter 4

Experiments

This chapter concerns all the analysis of the methods when used to solve the test problems. The goals are, at first, to see if the methods solve the problems accurately which is not always guaranteed by the theory. Then we shift our concerns to the efficiency of the methods. We try to highlight the operations that are computationally expensive and spot where improvements can be made. Each algorithm is first analysed separately. At the end of the chapter we make a summary by showing which Riemannian nonsmooth algorithm is best suited for each test problem.

4.1 Riemannian optimization library

The experiments in this chapter were conducted with the Riemannian Optimization Library available at `http://www.math.fsu.edu/~whuang2/Indices/index_ROPTLIB.html`. This is a C++ library written by Wen Huang. It features state of the art algorithms for Riemannian optimization, as well as many manifolds.

4.2 RBFGS

We first look at the RBFGS method (algorithm 8). In [LO13] we read *"The success of quasi-Newton methods when f is sufficiently smooth with nonsingular Hessian at a minimizer is in large part because inexact line searches quickly find an acceptable step: eventually the method always accepts the unit step and converges superlinearly. The behavior of these methods with an inexact line search on nonsmooth functions is complex: it is far from clear whether the direction will be well scaled."*. This is a very precise analysis that will guide our own experiments. Indeed, we should hope that in the best cases, the directions given by RBFGS will allow a constant step size. This reduces the time spent in the line search and therefore the number of cost function, gradient and retraction evaluations. Converging towards a constant step size is also related to the (superlinear) speed of convergence for quasi-Newton methods in the smooth case [NW06]. Unfortunately the authors of [LO13] have not found a clear answer to this question of scaled directions for nonsmooth Euclidean optimization. So we do not know if a better behaviour is to be expected in the Riemannian setting.

We first look at the results when RBFGS is used for the OBB problem and then for sparse PCA. For the OBB problem we will analyse the line search particularly. And the stopping criterion for the sparse PCA problem.

4.2.1 Orthogonal bounding box

Since no convergence results are yet available for nonsmooth RBFGS (algorithm 8), we shall first see if the method solves the problem correctly and how the method behaves on the problem. When the data points are in $\mathbb{R}^2$, it is possible to draw the bounding box and the entire cost function. The cost function only depends on one angle to define the rotations in $\mathbb{R}^2$. We describe the input parameters used for the first tests. The goal at this stage is not to optimize the process but to simply develop our understanding.

The initial iterate in O_n was obtained as the Q factor of a random matrix in $\mathbb{R}^{n\times n}$. The initial Hessian approximate used is always the identity. The qf retractions and vector transport by parallelization were used, as defined in section 1.3. The positions of the data points were generated randomly. The settings for the coming tests are summarized in the pseudo code below:

```
% Input parameter for RBFGS Algorithm
n = dimension of embedding space
N = number of points
retraction = qf
vector transport = parallelization
epsilon_d = 1e-4
epsilon_x = 1e-6
J = n(n-1)/2 +1
A = rand(n,N)
x_0 = qf (rand(n,n))
B_0 = Id
```

On figure 4.1 we start gently with only 5 points. The cost function is represented for values of the angle on the interval $[0, \frac{\pi}{2}]$ in radian. Due to the nature of the problem, the cost is periodic with period $\frac{\pi}{2}$ and $f(0) = f(\frac{\pi}{2})$. For $N = 5$, the cost has a global nonsmooth minimizer. The point reached by the solver is in red. The corresponding box is in red on the right. It corresponds to a rotation of 42, 23 degrees. This is the angle that the box forms with the x-axis on the right. We see that the global minimizer is the solution found by the solver. Let us now take a look a figures 4.2 and 4.3 where we used $N = 20$ and $N = 500$ points respectively.

For 20 points, the global minimizer is again reached. The cost function seems to have more local extrema. For the case with 500 points, there are even more local extrema. The solver has reached a local minimizer which is this time not global. It should come as no surprise because the RBFGS method is a local method. We can take good notice that the OBB problem is highly nonconvex. And the more points there are, the more local minima appear in the function. For the three examples show, it is clear that all the solutions obtained are points of nondifferentiability because the corresponding bounding box always has an edge with two or more points on it. If the number of point N keeps increasing (up to 10^6 points or more), shapes for the cost function similar to figure 4.3 are observed.

We conclude that the RBFGS method seems to solve this problem accurately. We have never encounter issues as long as the initial iterate x_0 is initialized randomly. The inexact line search coupled with a retraction seems to avoid nonsmooth points. The method does not break down in practice, similarly to the Euclidean observations of [LO13].

Let us try dimensions n greater than 2. We can no longer plot the cost function on its domain. In table 4.2 we have tried dimensions from 2 to 8 and displayed the operations performed before the stopping criterion was reached (algorithm 6). We used a tolerance of

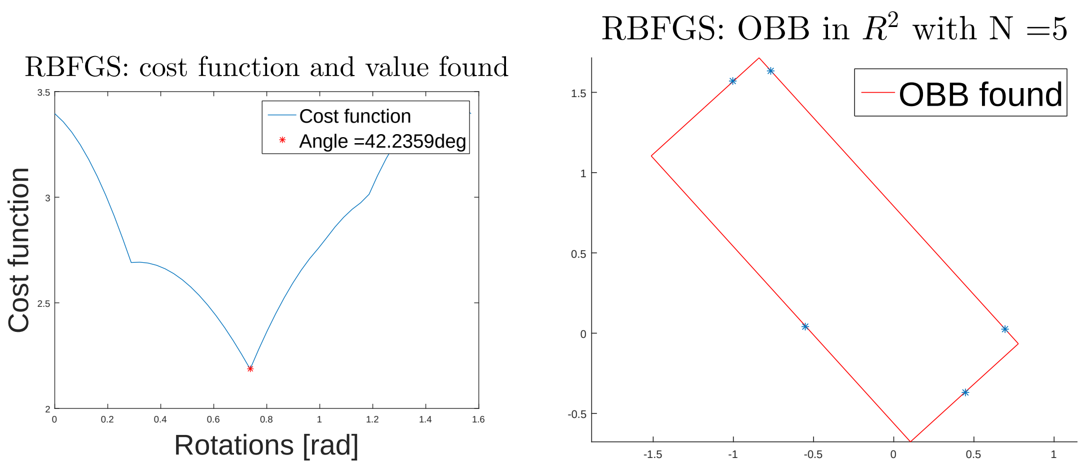


FIGURE 4.1: Solution to OBB with RBFGS for 5 random points in $\mathbb{R}^2$

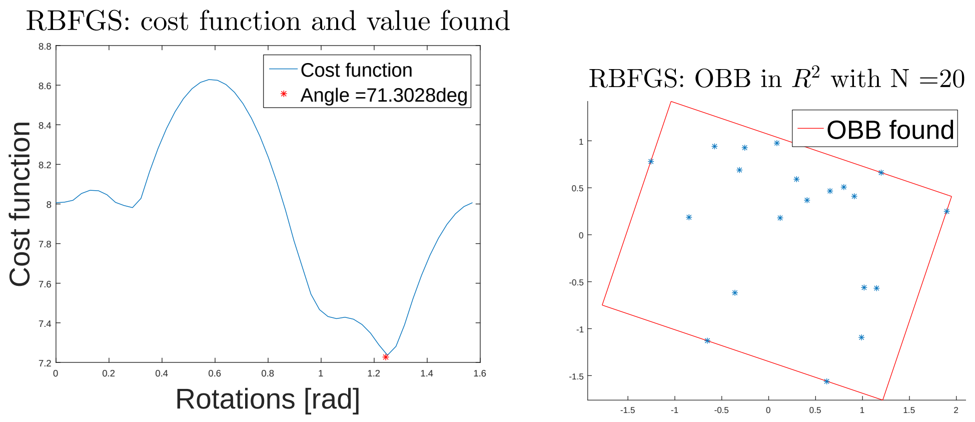


FIGURE 4.2: Solution to OBB with RBFGS for 20 random points in $\mathbb{R}^2$

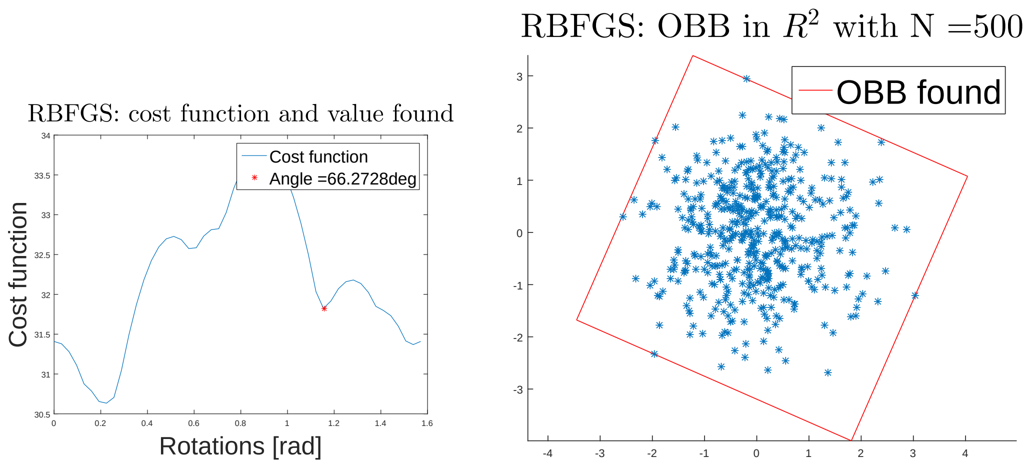


FIGURE 4.3: Solution to OBB with RBFGS for 500 random points in $\mathbb{R}^2$

$\varepsilon_d = 1e-4$. Bear in mind that for some value of n, the dimension of O_n is $d = n(n-1)/2$. Some notations are given in the table 4.1. They will serve throughout all the experiments.

What we see first is an expected increase in the complexity with respect to the dimension n. For $n = 8$, the solver reached 2000 iterations which was the maximum allowed and it then stopped. What happens is that the iterations have indeed reached a local

Definition of abbreviations	
iter	Number of iterations
nf	Number of cost function evaluations
ng	Number of gradient evaluations
nH	Number of action of inverse Hessian
nR	Number of retractions
nV	Number of vector transport

TABLE 4.1: Notations for the measurements of the tests

n	d	iter	nf	ng	nH	nR	nV	Time [s]
2	1	8	31	19	8	30	41	0.01
3	3	33	84	60	33	83	282	0.17
4	6	52	121	85	52	120	602	0.28
8	28	2000	5398	3693	2000	5397	160004	221.51

TABLE 4.2: Number of operations performed during resolution of OBB. 500 points with RBFGS

minima but the stopping criterion failed to trigger soon enough. This happens commonly in larger dimensions and we will make a more detailed analysis of the stopping criterion a bit further in this section. For dimensions 2 to 4, the stopping criterion was satisfied after the number of iterations indicated. The number of actions of the inverse Hessian is equal to the iteration counter, since only one action is used per iteration to get the search direction. Except for the first value $f(x_0)$, every time the cost is evaluated, a retraction is used. This is why we observe $nR + 1 = nf$. The number of gradients evaluated, ng, is a bit smaller than nf. This is due to the line search.

Finally, nV is quite larger compared to the other counters. There are two main reasons for this. There are all the vector transport performed to update the inverse Hessian at each step. And also all the gradients that must be transported for the stopping criterion once the iterations become close to one another. As the dimension increases, we have noticed that the stopping criterion takes an increasing part of the running time. It is difficult to afford the use of this stopping criterion for increasing dimensions. Luckily, for the OBB problem the vector transports are the identity since we use the intrinsic approach. So they do not increase the computation time.

As a warning to the reader, it should be said that the computation time is not a great measure of the performances. Especially the time in absolute value which is almost irrelevant. It is highly dependent on the processor used and all the operations on the platform at running time. It only makes sense to compare times for operations ran at the same moment on the same platform. The computation times are also more reliable because the library is in C++ as opposed to other languages like Matlab. So even if we have to use it with caution, the running time can still be a great indicator of the efficiency of the methods. Because, in applications, it is usually the running time that matters after all. What we see here, in table 4.2, is that the time seems to increases enormously with the dimension. As n doubles for 2 to 4, the computation time takes a factor of about 28. We have observed that this is largely due to the stopping criterion. Indeed let us now perform the same tests with a modified stopping criterion. We use a simple

$$\frac{|f(x_k) - f(x_{k-1})|}{|f(x_k)| + 1} < 1e - 8. \tag{4.1}$$

n	d	iter	nf	ng	nH	nR	nV	Time [s]
2	1	3	21	12	3	20	17	0.001
4	6	33	65	48	33	64	113	0.01
8	28	220	370	294	220	369	733	0.10
16	120	1535	2397	1965	1535	2396	5034	1.28

Table 4.3: Number of operations performed during resolution of OBB. Stopping criterion from equation (4.1). 500 points with RBFGS

Results for this approached stopping criterion are in table 4.3. First, it appears that the dependence on the dimension n is still problematic. As the number of iterations increases quickly. But this time, for $n = 8$, the approached stopping criterion was triggered. After 220 iterations only, the stationary point was reached. So there is something to fix in the nonsmooth stopping criterion algorithm that did not work for $n = 8$ in the first example. With the approached stopping condition, the iterations are performed much faster because there is no need to solve a quadratic program at each step. This allowed us to solve the problem up to $n = 16$ in 1.28 second.

Rate of convergence

An important measure of efficiency for a method is the rate of convergence. We will present the rate of convergence of the function values. Since the optimal value $f(x^*)$ is not available in the applications, we use a common trick. The last value reached is taken as an approximation of $f(x^*)$.

We observe that the rate of convergence is linear. On figure 4.4, we show the error in the cost function in log scale. It was expected since the Euclidean version of this algorithm was already showing linear convergence on nonsmooth functions in $\mathbb{R}^n$.

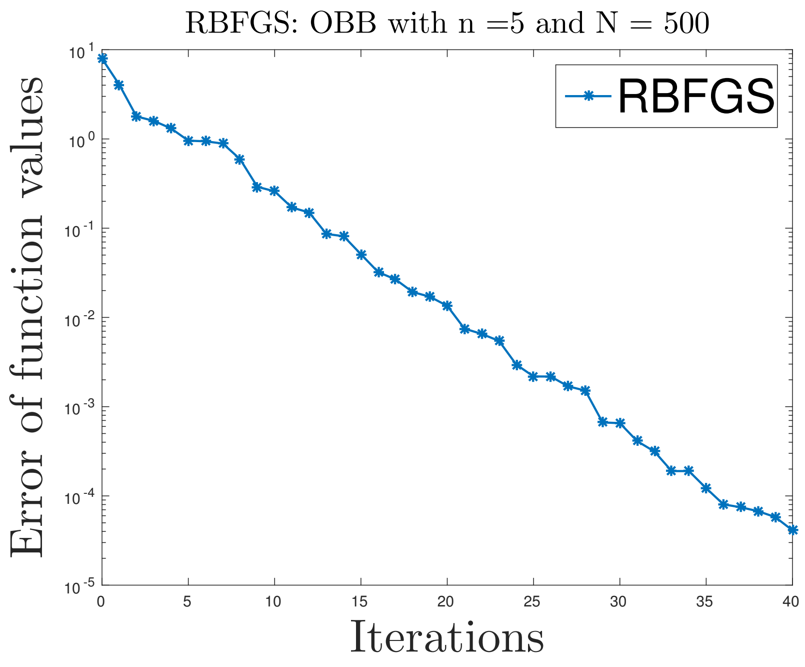


Figure 4.4: The convergence rate of RBFGS for OBB is linear.

We also observed that the RBFGS seems to always generate descent directions. In fact we have never encounter cases where that did not seem to hold. Even though this is not guaranteed by any theory at the moment.

The case $n = 3$

This case $n = 3$ deserves special attention because it is the most prone to applications. We use the nonsmooth stopping criterion since it works well for low dimensions. We will focus on problems with $N = 500$ points because it seems general enough. Increasing N will not change the nature of the problem but only the cost of computations. On table 4.4, we fixed $n = 3$ and had N increasing. The values are an average over 100 problems solved. We see that the number of iterations hardly changes as N increases. If anything it seems to decreases. Our guess is that it might be because the shape of the cost function is a bit more spiky for smaller values of N. It makes it harder for the nonsmooth stopping criterion to stop and this increases iter, nf and nV. On the other end, the computation time clearly increased when N becomes large enough ($N = 10^6$). This should only be due to the cost of the functions evaluations. Evaluating the cost function, as we know from section 3.2, increases in $\mathcal{O}(Nn^2)$.

N	iter	nf	nV	Time [s]
5	27	74	213	0.11
100	20	46	130	0.05
500	16	34	88	0.02
10 000	17	32	88	0.4
1 000 000	19	36	110	2.94

Table 4.4: Average over 100 cases for the number of operations performed during resolution of OBB. Number of point N increasing for $n = 3$.

The RBFGS solves the OBB problem with probability 1 for an random x_0 and the rate of convergence for the function values is linear. The stopping criterion algorithm is very expensive in computation time and does not seem to work well for large dimensions, which we will investigate when we talk about the sparse PCA problem. Increasing the number of points N has an impact on the computation time. We now analyse the line search algorithm for $n = 3$.

The line search

A question raised in [LO13] is whether the method is well scaled. By this we mean, do the search directions generated allows for step sizes of constant length ? If after some number of iterations, some step length is always allowed, we say that the method is well scaled. This makes the line search very efficient. When a unit step is admissible, we would hope that the methods tends towards the Newton method and that the inverse Hessian approximation is accurate. Whether BFGS is well scaled in general is a difficult question. It seems to be understood in smooth optimization [NW00], but still very much unclear in the nonsmooth case [LO13]. We want to find out how the line search (algorithm 5) behaves for partly smooth functions.
In the previous examples (tables [4.2][4.3]), we did not mention the ratio $\frac{nR}{iter}$, which is the average number of trials in the line search before an acceptable step size is found. This number lies around 2 or 3 in the previous examples. This tells us that the inexact line search, on average, took between 2 and 3 trials to find an acceptable step length.

According to [LO13], achieving a low number of trials in the line search is one of the keys to BFGS's efficiency for nonsmooth problems. We are now going to look at the line search parameters and see how they affect the average number of trials and the scale of the directions. The line search parameters are the coefficient c_1 and c_2 from the weak Wolfe conditions. Another degree of freedom in the line search is the strategy for the first step length tried at each step. We first observe the effect of varying values for c_1 and c_2 on this problem.

In general, there are no theoretical justifications to chose $0 < c_1 < c_2 < 1$ for the weak Wolfe conditions. Popular heuristics tend to favour values around $c_1 \approx 10^{-4}$ and $c_2 \approx 0.9$ for quasi-Newton methods[NW00]. The ultimate goal of such an inexact line search is usually only to get to an area near the minima where unit steps allow fast convergence.

For each of the measures of the performances (iter, nf,trials, nV,...), we show a double entrance array with different combinations of c_1, c_2 around the popular values. After experimenting, we decided to show combinations of the values
$c_1 = \begin{pmatrix} 0.2 & 0.1 & 1e-2 & 1e-4 & 1e-10 \end{pmatrix}$ and $c_2 = \begin{pmatrix} 0.5 & 0.7 & 0.9 & 0.999 & 0.99999 \end{pmatrix}$. For each problem, the same starting point x_0 is used for every combination of the parameters c_1, c_2. The stopping criterion algorithm was used with a tolerance of $\varepsilon_d = 10^{-8}$ and $\varepsilon_d = 10^{-2}$. The results are in table 4.5. The measured performances are an average over 1000 problems solved. We first discuss the effect of changing c_1.

Effect of c_1 Reducing c_1 bring the tangent line closer to the horizontal and allows to take larger steps. Reducing c_1 therefore makes it also easier to satisfy the Armijo condition and a smaller decrease in the function is imposed. We see that for all the sub arrays of table 4.5, reducing c_1 below 10^{-2} does not make a difference. As results are practically the same for $c_1 \in \{10^{-2}, 10^{-4}, 10^{-10}\}$. When c_1 becomes larger than 10^{-2}, the number of trials in the line search increases, because it becomes harder to satisfy the Armijo condition. But increasing c_1 forces a bigger decrease in the cost function, so the number of iterations is a bit smaller. But the effect is not very clear (depending on the value of c_2). In short when c_1 increases, the number of trials in the line search increases, but the number of iteration decreases. The effect on the overall computation time is not conclusive. Since it does not have a clear tendency. It might also depend on the number of points N. For instance if N is huge, one would probably like to reduce the number of cost function evaluations as much as possible. And therefore selecting $c_1 \leq 10^{-2}$ might help.

Effect of c_2 When c_2 gets close to 1 in the weak Wolfe condition, the condition becomes almost trivial to satisfy and the derivative is barely constrained to reduce. Taking $c_2 = 0.5$ forces the directional derivative at the next iterate to be twice smaller (and therefore closer to zero). What we see is that taking smaller c_2 indeed reduces the iteration counter because it reduces the derivative more at each iterate. We go from an average of about 14 iterates with $c_2 = 0.5$ to 20 iterations with $c_2 = 0.99999$. As a results, nV also clearly increases with c_2. The same goes for the number of cost function evaluations (nf). It goes from around 40 to 55. The effect of c_2 on the number of trials in the line search is a bit harder to see. Overall, the computation time also increases with c_2. So it looks clear that a value of c_2 towards 0.5 is much more suited in this context than 0.9 for instance. According to [NW00], the value $c_2 = 0.1$ is a good reference for nonlinear conjuguate gradients. We believe that it is interesting but not really conclusive that smaller values for c_2 work better on this problem for a quasi-Newton method.

For the remaining of this section, parameters will have the values $c_1 = 1e - 4$ and $c_2 = 0.5$ unless mentioned otherwise.

Iterations					
c_1 c_2	0.2	0.1	10^{-2}	10^{-4}	10^{-10}
0.5	13.7	13.9	14.5	14.4	14.4
0.7	15.3	14.9	15.1	15.0	15.0
0.9	16.7	17.1	18.4	18.3	18.4
0.999	17.9	18.4	19.4	19.4	19.5
0.99999	18.4	19.0	20.1	19.9	19.9
Cost function evaluation					
c_1 c_2	0.2	0.1	10^{-2}	10^{-4}	10^{-10}
0.5	41	40	37	37	37
0.7	46	42	39	38	38
0.9	49	49	51	49	49
0.999	53	52	53	52	52
0.99999	56	55	56	54	54
Trials in line search					
c_1 c_2	0.2	0.1	10^{-2}	10^{-4}	10^{-10}
0.5	3.12	2.91	2.72	2.69	2.69
0.7	3.10	2.91	2.71	2.69	2.69
0.9	2.96	2.85	2.73	2.69	2.69
0.999	3.01	2.96	2.79	2.73	2.73
0.99999	3.11	2.99	2.83	2.79	2.79
Vector transport					
c_1 c_2	0.2	0.1	10^{-2}	10^{-4}	10^{-10}
0.5	84	83	83	80	80
0.7	104	97	90	87	87
0.9	116	118	127	125	125
0.999	125	127	130	129	130
0.99999	131	134	139	136	135
Computation time $[s]$					
c_1 c_2	0.2	0.1	10^{-2}	10^{-4}	10^{-10}
0.5	0.032	0.034	0.033	0.032	0.031
0.7	0.047	0.044	0.036	0.034	0.034
0.9	0.052	0.056	0.062	0.061	0.061
0.999	0.052	0.055	0.058	0.058	0.058
0.99999	0.056	0.060	0.064	0.063	0.062

TABLE 4.5: Behaviour of the method for different line search parameters c_1 and c_2. Dimension $d = 3$ and $N = 500$ points. Average over 1000 problems. $\varepsilon_d = 10^{-8}$.

Strategy for the first step The previous results were obtained with the previous step length as an initial candidate for the next line search. This is only one among other possibilities. As said in [NW06], "*For methods that do not produce well scaled search directions, [. . .], it is important to use current information about the problem and the algorithm to make the initial guess*". Since we have observed that the directions do not allow, in general, to use a constant step size, we will consider a few heuristics. This will help spare even more trials in the line search. We shall consider four heuristics to estimate α_k, labelled α_k^{01}, α_k^{02}, α_k^{03},α_k^{04}.

The first strategy is to use the previous step as an estimate, we therefore have $\alpha_k^{01} = \alpha_{k-1}$. Another simple approach is to always try 1 as the step length. We write $\alpha_k^{02} = 1$. In [NW06] they propose the following idea that we have adapted to Riemannian optimization. Interpolate the data $f(x_{k-1})$, $f(x_k)$ and $< \text{grad}f(x_{k-1}), p_{k-1} >$ with a quadratic function and take the minimizer as the step length. Which yields

$$\alpha_k^{03} = 2\frac{f(x_k) - f(x_{k-1})}{< \text{grad}f(x_k), p_k >}. \tag{4.2}$$

They pretend that this ratio converges to 1 whenever $x_k \to x^*$ superlinearly. Moreover, the choice

$$\alpha_k^{04} = \min(1, 1.01\alpha_k^{03}) \tag{4.3}$$

seems to fit very well for quasi-Newton methods in the setting of smooth Euclidean functions. They observe that eventually the unit step length is always tried and accepted. Superlinear convergence to the stationary point usually follows in practice. Although this might not apply to nonsmooth Riemannian optimization, it is a promising heuristic. The four heuristics compared are summarized in table 4.6.

Heuristic 1:	$\alpha_k^{01} = \alpha_{k-1}$
Heuristic 2:	$\alpha_k^{02} = 1$
Heuristic 3:	$\alpha_k^{03} \longleftarrow$ formula (4.2)
Heuristic 4:	$\alpha_k^{04} \longleftarrow$ formula (4.3)

Table 4.6: The four different strategies for the initial guess of the step length

Initial stepsize strategy	Iterations	Number of trials	Computation time $[s]$
Heuristic 1	19.4	2.69	0.052
Heuristic 2	14.8	2.51	0.038
Heuristic 3	17.0	2.39	0.040
Heuristic 4	16.6	2.20	0.031

Table 4.7: Analysis of the best initial step size strategy for OBB: $n = 3$ and $N = 500$. Average values over 1000 examples.

We have solved 1000 problems with $n = 3$ and $N = 500$ for each heuristic. In table 4.7, we show the average number of iterations, trials in the line search and computation time. We made the following observations.

Heuristic 1 : We see that using the previous step length as a guess for the next iterate is the worst heuristic among the ones tried. Even though it seems very natural, it yields on average more iterations, more trials in the line search and a larger computation time than all the other heuristics. In practice we observed that the unit step size is not often tried and not often accepted.

Heuristic 2 : Here the unit step size is always tried. So we are guaranteed that it would be used if it can be accepted. We observed that it is not usually accepted. However, this method still outperforms the previous one for the number of trials, iterations and running time. It is the strategy with the largest steps tried so it is the one doing the fewest iterations.

Heuristic 3 : Better than the previous two strategies, but not quite as good as the last one. The unit step size is almost never tried.

Heuristic 4 : This final heuristic from [NW06] is definitely the best we have tried. It minimizes the number of trials in the line search. We have observed that, at the end of the iterations, the unit step length is sometimes tried and even then not always accepted.

We conclude the analysis of the line search. We have observed that for this problem, the values $c_1 = 10^{-4}$ and $c_2 = 0.5$ seem to be the best to use for the weak Wolfe conditions in this setting. It minimizes the number of iterations and trials in the line search. For the initial guess of the step size, formula (4.3) from [NW06] stands out. But it does not allows to reach a constant step size of 1 in general.

4.2.2 Sparse PCA

For the sparse PCA problem, we are hoping to consider much larger instances. As in [JNRS08], we set the value of the penalty parameter μ, relatively to $\|a_{i^*}\|_2$, with a_{i^*} the column of A with the largest ℓ_2 norm. We show the cost function for $n = 2$. Since $\mathbb{S}^1$ depends only on the angle along the sphere. Points on the manifold only depend on a rotation angle. For $\mu = 0.5 * \text{norm}(a_{i^*})$ (figure 4.5), the cost function nearly looks smooth at first glance. But we can see a small discontinuity in the gradient for 90, 180 and 270 degrees on the circle. It corresponds to points $(0, 1)$, $(-1, 0)$ and $(0, -1) \in \mathbb{S}^1$. Those are points where the ℓ_1 norm is not differentiable. There is no gradient at $(1, 0)$ either but it is not apparent on the graph because of the disconnection in the plot.

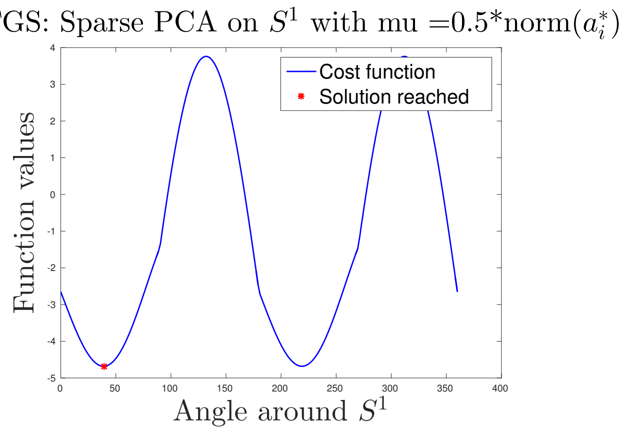


FIGURE 4.5: Cost function and solution found for PCA with $n = 2$

For $\mu = 0.5 * \text{norm}(a_{i^*})$, the breaks are starting to become more apparent. But the solution found by RBFGS (in red, figure 4.6) is still a smooth point. For $\mu = 1.7 * \text{norm}(a_{i^*})$ (figure 4.7), the point reached is at $(0, 1)$, at 90 degrees on the circle, because the penalty is too important and must be minimized as much as possible.

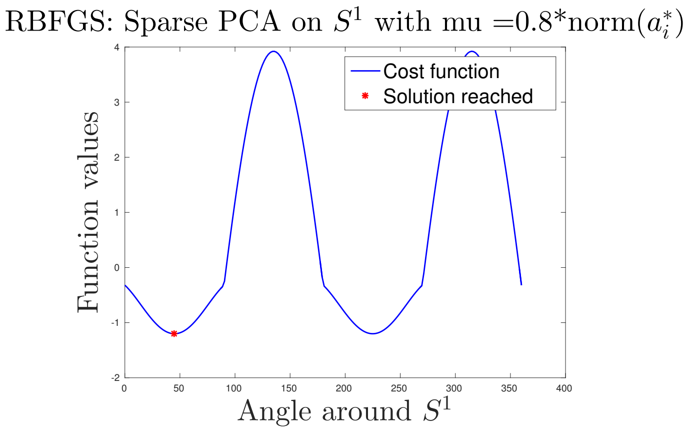


FIGURE 4.6: Cost function and solution found for PCA with $n = 2$

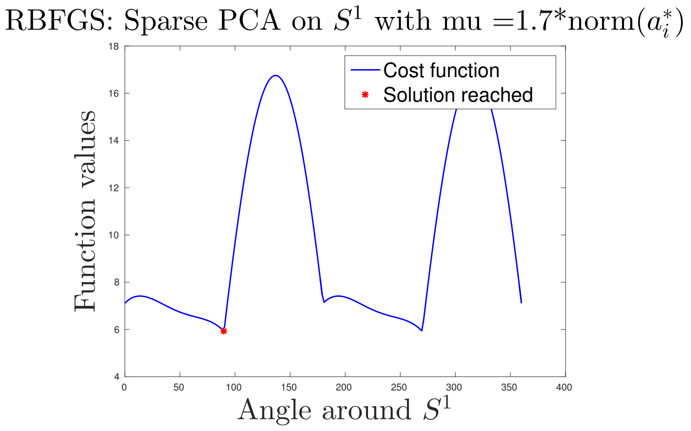
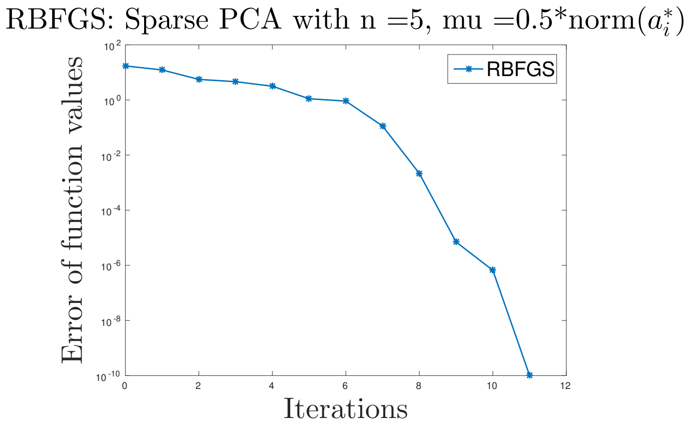


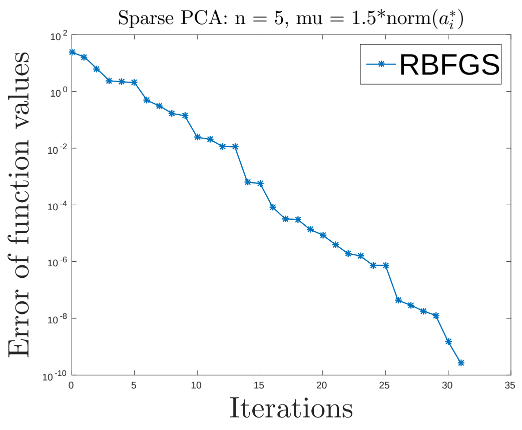
FIGURE 4.7: Cost function and solution found for PCA with $n = 2$

Convergence rate

We now look at the convergence rate for the sparse PCA problem. Things are not so clear in this case. As the value of μ, the parameter for the ℓ_1 penalty has a great influence on the smoothness of the function. For small values of μ, we often see that RBFGS converges superlinearly. For larger values, we fall back to linear convergence, see figure 4.8.



(a) Superlinear rate for $\mu = 0.5$



(b) Linear rate for $\mu = 1.5$

FIGURE 4.8: The rate of convergence for RBFGS on the sparse PCA problem depends on μ.

Stopping criterion

The stopping criterion depends on three parameters. Namely, ε_x, ε_d and J in algorithm 6. The value ε_d is the tolerance on the norm of the smallest vector in the subdifferential. It is for the user to specify, depending on his own requirements in precision. The minimal distance between iterates before the algorithm begins to store gradients is ε_x, and J is the maximum number of gradients stored. We are going to see how the stopping algorithm works on the examples and compare it with two other ideas: algorithm 10 and 7. The former is an explicit expression for the subdifferential based on the form of the cost function and the latter is a gradient sampling test.

In the following analysis, we solve the sparse PCA problem for $n = 5$. The goal is not to modify the value ε_d, we will keep it at 10^{-8} for the upcoming experiments.

On figure 4.9 we show the norm of the error for the three algorithms. Whenever $\|x_k - x_{k-1}\| \geq \varepsilon_x$, it is the norm of the gradient that is taken as the error. Obviously this is not expected to go towards zero. When $\|x_k - x_{k-1}\| < \varepsilon_x$, the error is the norm of the smallest vector in the approached subdifferential. Each method having his own way of approaching $\partial f(x)$. The choice of parameters is resumed in table 4.8. We took $\varepsilon_x = 10^{-2}$. For algorithm 6, we used $J = n + 1$. We see that the iterations using algorithm 6 continue long after the two other other stopping criterion have stopped. The condition $\|x_k - x_{k-1}\| < \varepsilon_x$ was first satisfied at iteration 8, and still it took 30 iterations to stop. We have frequently observed that behaviour and would like to avoid it. Based on this observation, the goal is to find the best parameter values and stopping algorithm, so that the approximation of $\partial f(x)$ is rapidly accurate.

Progressive storage of gradients	Algorithm 6	$\varepsilon_x = 10^{-2}$, $J = n + 1$
Analytical subdifferential for sparse PCA	Algorithm 10	$\varepsilon_f = 0.1 * \|x\|_1$, $\varepsilon_x = 10^{-2}$.
Gradient sampling subdifferential	Algorithm 7	$\varepsilon = 10^{-2}$, $\varepsilon_x = 10^{-2}$

TABLE 4.8: Settings used for figure 4.9.

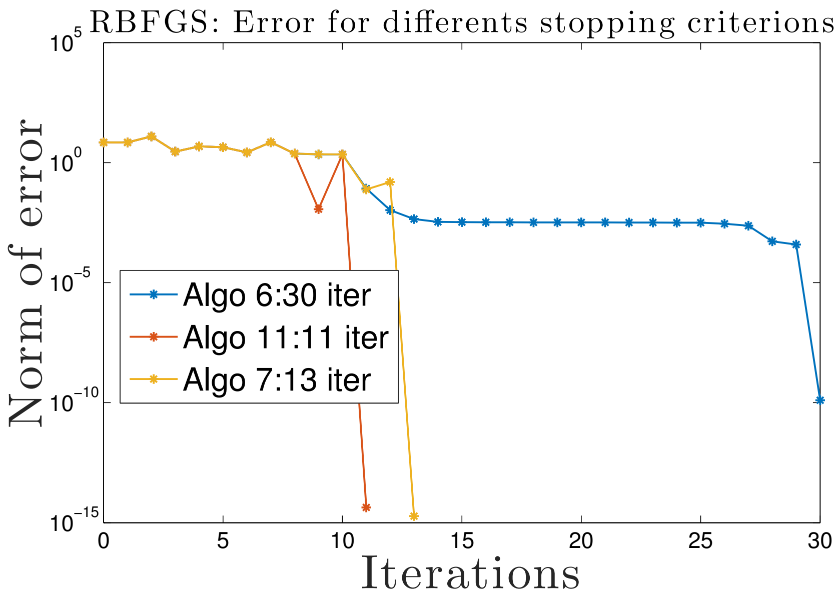


FIGURE 4.9: The algorithm 6 does not work well with $J = n + 1$.

We describe our observations for each algorithm.

Algorithm 6: We have experienced that choosing ε_x larger helps the criterion to trigger more rapidly. On the sphere $\mathbb{S}^{n-1}$, the distance between two points should be $d(x, y) \leq 2 \ \forall x, y \in \mathbb{S}^{n-1}$. This is a characteristic distance that stands as a reference for the distance in the space relative to the size of the space. As opposed to the the absolute distance between two points that has less meaning. We found that $\varepsilon_x = 2 * 10^{-2}$ is a well suited value. It allows the stopping criterion to trigger quite rapidly after the condition $\|x_k - x_{k-1}\| < \varepsilon_x$ was met. The stopping criterion was less stable for $\varepsilon = 10^{-6}$ for instance. As a small displacement might cause to clear all the gradients stored, and make it difficult to recreate the subdifferential. Another observation we have made is that it never seems good that the value J limits the number of gradients stored. The idea behind this value is probably to avoid solving minimum norm problems of too large size. But it seems more important to use all the neighbouring gradients possible. Choosing values close to $J = n+1$ simply does not seem to work. Exception perhaps of some simple case in small dimensions. We have observed that the value j_k (the number of gradients stored) frequently reaches $n + 1$ before the tolerance is met in the algorithm. When working in small dimensions, it can usually be afforded to take $J \approx$ iter-max. So that J does not become restrictive. This was the least stable algorithm of all three.

Algorithm 10: This algorithm is described in section 3.1. The subdifferential is approached using the structure of the cost function. We use that the subgradients of the ℓ_1 norm

are known. The tolerance ε_f is taken as a fraction of the reference value $\|x\|_1$. We experimented good results with $\varepsilon_f = 0.1 * \|x\|_1$. What we observed is that once $\|x_k - x_{k-1}\| < \varepsilon_x$ is satisfied, a first subdifferential test is tried. Usually, all the active functions are detected right away, but the smallest norm in the convex hull lies around 10^{-5}. And then, in most cases it requires a few more iterations for the norm to drop below 10^{-8}. But during theses few iterations, the actives functions in the max do not change. Which seems good because it means that x_k must be close enough to the minimizer for the norm to drop low enough, but the functions that are active in the neighbourhood have been spotted.

Algorithm 7: For the gradient sampling test, the tricky question is to find how many points have to be sampled to approach $\partial f(x)$. This is a very reliable stopping criterion in small dimensions. On the other hand, for larger dimensions, the number of sampling gradients needed for the algorithm to work well is much too large. And computing the convex hull becomes painful.

In table 4.9 we summarize the parameters value that we retain. These are the parameters used in figure 4.10. This is an example of what usually happens when the parameters are set in the way that we have found appropriate. It took 10 iterations for the condition $\|x_k - x_{k-1}\| < 2 * 10^{-2}$ to be satisfied. In this case, the two stopping conditions found the minimizer just after that. And it took 21 iterations for the original algorithm. Adapting the value J has allowed to reduce considerably the number of iterations needed in algorithm 6. We rarely have to wait 30 or more iterations as it was the case on figure 4.9.

Progressive storage of gradients	Algorithm 6	$\varepsilon_x = 2 * 10^{-2}$, $J = 100$
Analytical subdifferential for sparse PCA	Algorithm 10	$\varepsilon_x = 2 * 10^{-2}$, $\varepsilon_f = 0.1 * \|x\|_1$.
Gradient sampling subdifferential	Algorithm 7	$\varepsilon_x = 2 * 10^{-2}$, $\varepsilon = 2 * 10^{-2}$

TABLE 4.9: Settings used for the nonsmooth stopping criterion for $n = 5$ and $\varepsilon_d = 10^{-8}$.

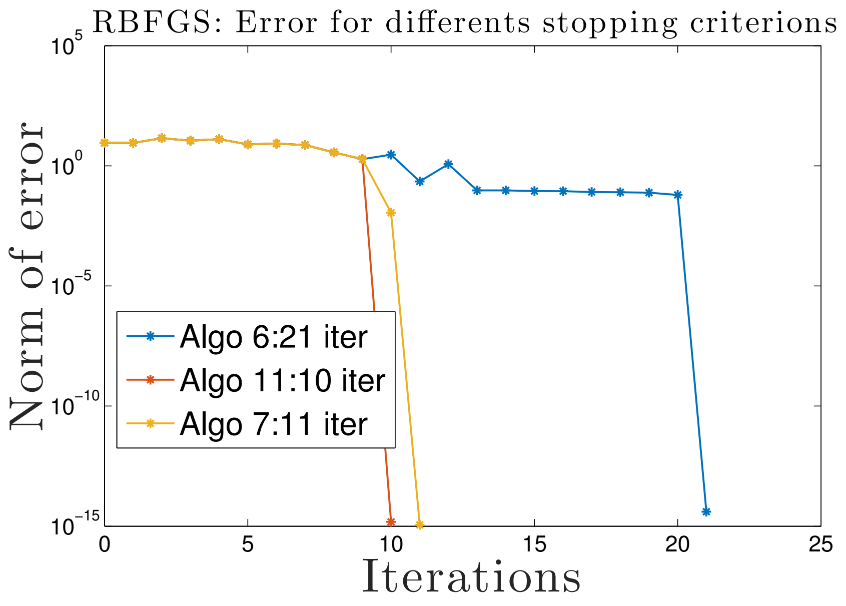


FIGURE 4.10: Different nonsmooth stopping criterions algorithms for the sparse PCA problem.

In conclusion, algorithms 10 and 7 work well in small dimensions. Possibly even better than the originally proposed algorithm 6, that has to recreate the subdifferential by moving around the minimizer. It is clear that the strategies presented cannot be used for large instance of the sparse PCA problem (several thousands of variables). In fact it does not seem that any method that anyway approximates the subdifferential and computes its smallest vector can be used for large dimensions if the computation time is a main concern. A criterion based on the function values or the displacement seems then to be the only option.

4.3 LRBFGS

In this section we present the experiments for the Limited memory RBFGS algorithm 9. We have observed that LRBFGS behaves as RBFGS regarding the line search and the stopping criterion. The real key to this algorithm is to make the best use of the memory parameter m. In general, there are no theoretical results to choose the parameter m. It depends on the problem, the dimension and the number of iterations. Indeed, we note that if m is greater than the number of iteration, the method is equivalent to a full RBFGS. While if $m = 0$, the algorithm reduces to a gradient descent. The reduction expected is in the computation time, since each iteration is cheaper. The cost at each iteration to find the search direction goes from $\mathcal{O}(n^2)$ to $\mathcal{O}(m * n)$. The gain in computation time should therefore be optimal if $m \ll n$ and the number of iterations is large. There are no specific expectations on the rate of convergence or the number of iterations compared to RBFGS. Reference [AS10] is a study of the nonsmooth LBFGS method in $\mathbb{R}^n$. They claim that the memory parameter often works best in the range $[1, 30]$, regardless of the dimension. Something we are looking to confirm. We analyse the behaviour on both test problems separately.

4.3.1 Orthogonal bounding box

On figure 4.11, we simply show the function values for different values of m on the case $n = 5$. The convergence is obtained after roughly 60 iterations. The nonsmooth stopping criterion was used with $\varepsilon_x = 10^{-2}$ and $\varepsilon_d = 10^{-8}$. We observe that the directions being different, the methods do not generate the same sequences of iterates. But they all converge towards the same minimizer and therefore all have the same final function value. We will first take a deep look the main advantage of this method. The time gain at each iteration. For the dimension of reference, $n = 3$, the number of iterations is often under 20. It this case, trying values m greater than the umber of iterations does not make much sense. Therefore, we also tried problems of larger dimensions. As a reminder, when the points are in $\mathbb{R}^n$, the orthogonal group as dimension $d = n(n-1)/2$. And since we use an intrinsic representation, this is the dimension of the tangent vectors stored.

For the following we proceeded as follows. We wanted to solver the same problems with different values of m. The stopping criterion was set to $\dfrac{|f(x_k) - f(x_{k-1})|}{|f(x_k)| + 1} < 10^{-8}$. Which allowed to solve the problem in larger dimensions. Computing the subdifferential would make it painful to solve problems of dimensions $n > 10$ and greatly diminish the importance of the parameter m in the computation time. We have defined the same initial iterate x_0 for every solver. For each case we measured `time-ratio = Running`

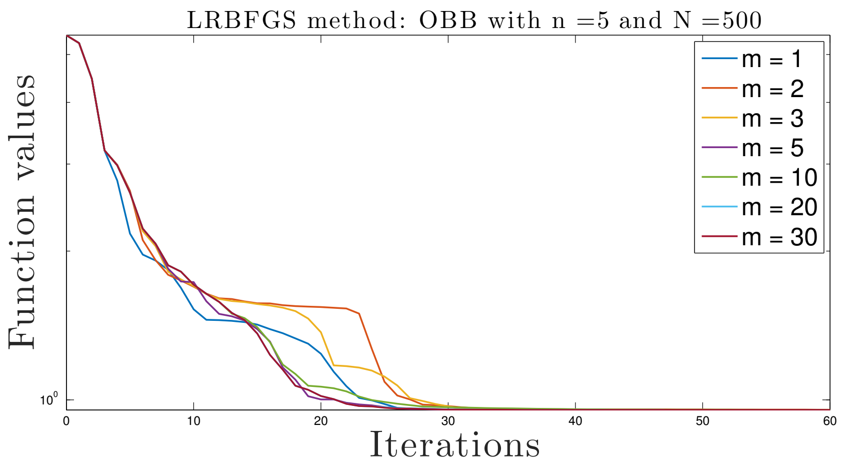


FIGURE 4.11: LRBFGS: Dependence on memory parameter for OBB

`time/iter`. And repeated the procedure 100 times to find the average `time-ratio` for each dimension.

On figure 4.12, we observe the following. As n becomes larger, there is roughly a factor 3 in computation time per iterations between the good values of m (1 and 2) and the worst value tried (30). It is totally expected that the time per iteration increases with m. The question now is which value for m improves the entire minimization process. This is answered in figure 4.13. In this case, $m = 2$ seems to be the best choice. Obviously it is a close call that was made on an average of 100 problems. The values $1, 2, 3$ are probably equally as good in practice. It confirms the claim made in [AS10], saying that the best values for m are usually very small in nonsmooth optimization. We note that RBFGS is just as good as LBFGS with $m = 2$ for small dimensions but is slower compared to any LRBFGS as n increases.

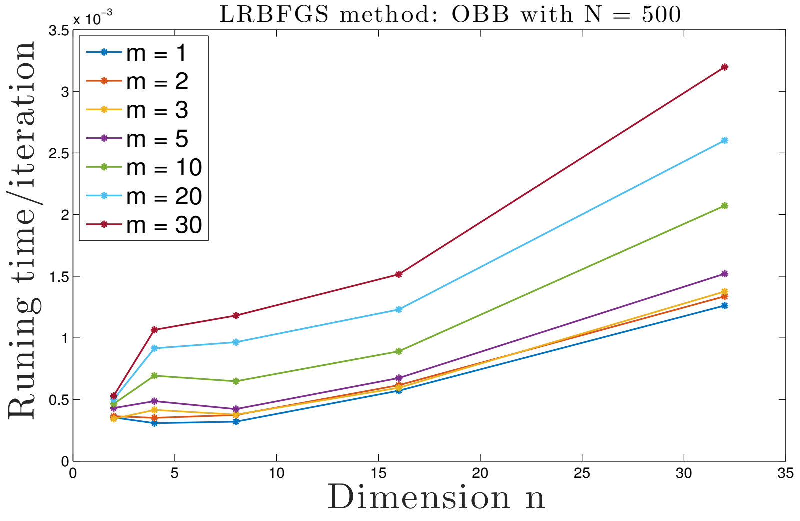


FIGURE 4.12: LRBFGS: Dependence on memory parameter for OBB.

The value of m is supposed to be a trade-off between a smaller computation per time and a better approximation of the Hessian. If m is small, iterations must be faster but we use less information to approximate the Hessian. The argument is often made that the advantage of small values of m is that we work with recent and local information about the Hessian. But this idea does not seem precisely understood at the time [AS10].

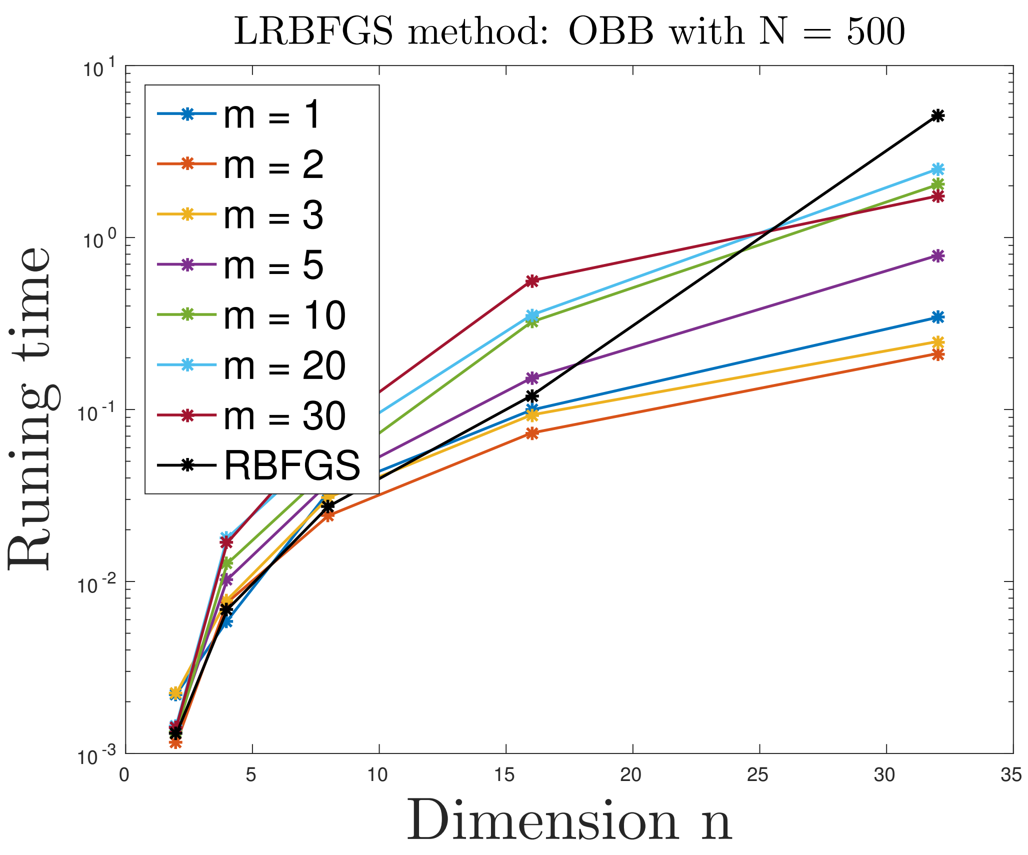


FIGURE 4.13: Comparison of different memory parameters for LRBFGS.

4.3.2 Sparse PCA

For the sparse PCA, instances are meant to be even larger. This should be the right context for the LRBFGS method to fully prove its efficiency.

We see that increasing the memory parameter m has a noticeable effect on the number of iterations. On figure 4.14, curves for small parameters reach the optimal value quicker. This is the representation of one sample problem for $n = 500$. The directions generated by the scheme for small m have helped to converge faster.

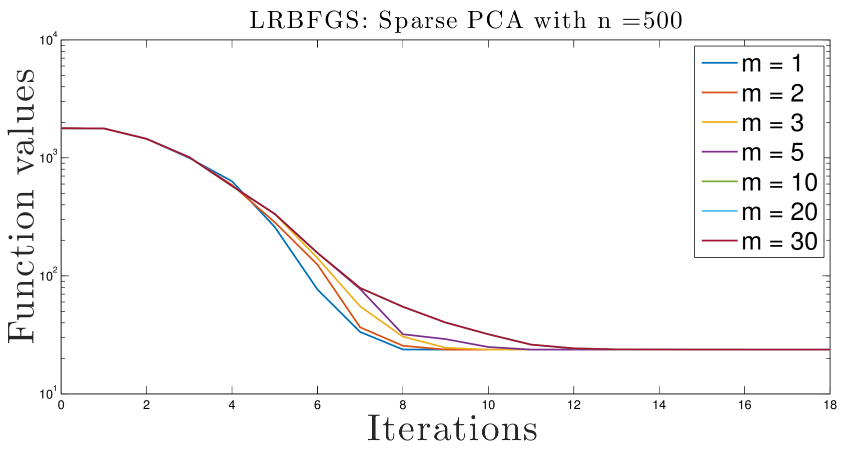


FIGURE 4.14: Dependance of memory parameter for sparse PCA.

The computation time confirms this idea. As the algorithm is much faster for smaller values of m, see figure 4.15. This graph represents the average running time over 100 samples. For each dimension we solved 100 problems and each problem as its own matrix A and starting point, common to all the solvers. Each solver as a different value for m. For the sparse PCA problem, when n becomes greater than 1000, there is a very important gain to use LRBFGS with say $m = 2$, compared to $m \geq 5$ or RBFGS.

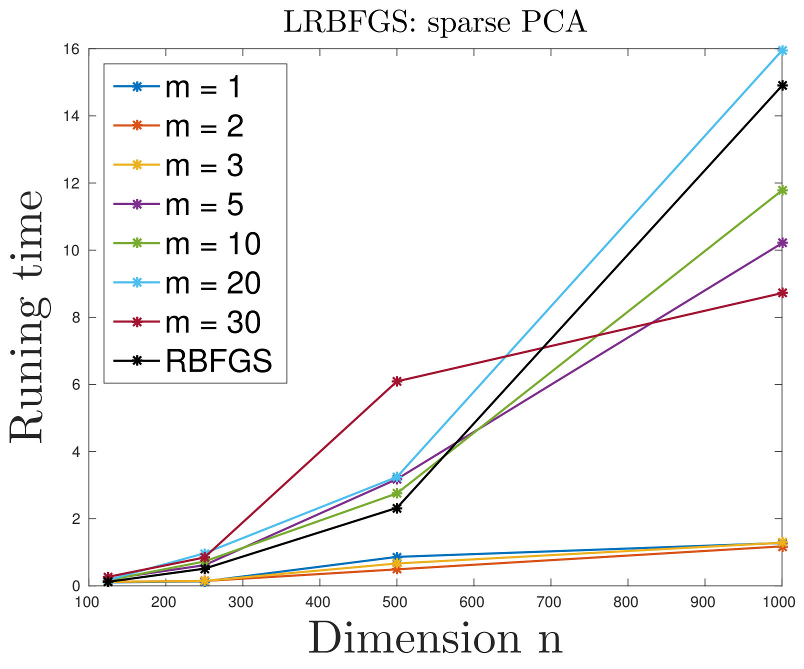


FIGURE 4.15: Dependance of memory parameter for sparse PCA

4.4 Subgradient methods

We now turn to the methods defined in section 2.3, namely subgradient RBFGS and subgradient LRBFGS. We begin with the RBFGS scheme and then we will discuss the limited-memory version.

After experimentation, we chose the following parameters to present the results.

```
% Input parameter for subgradient RBFGS Algorithm
epsilon_d = 1e-8
epsilon_x = 1e-2
x_0 = qf (rand(d,d))
B_0 = I_{d}
lambda = 1e-4
Lambda = 1e4
```

Convergence rate

Even though the method tries to get more information on the function by sampling some nearby gradients, linear convergence is observed, as we see for the OBB problem on (figure 4.16). This comes as a bit of a disappointment. As we might have hoped that the information given by the ε-subdifferential would have helped to converge faster than the careless RBFGS method.

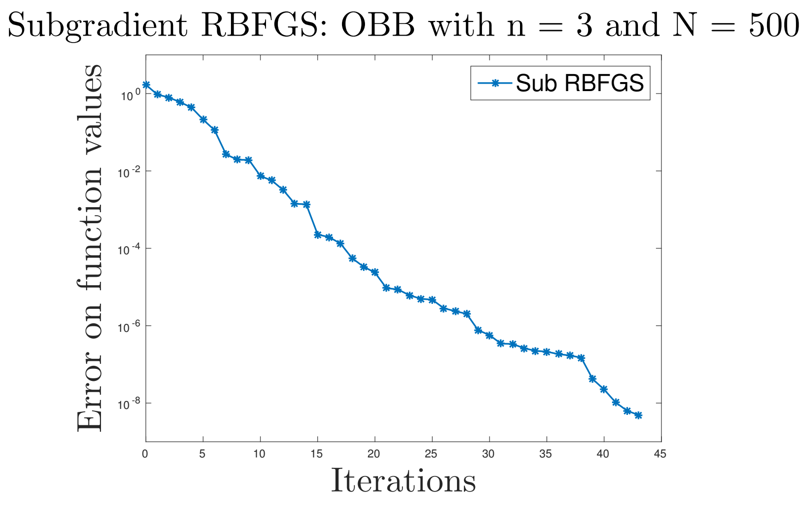


FIGURE 4.16: Linear convergence rate for subgradient RBFGS.

Approximation of the ε-subdifferential

We now wonder, how the subdifferential is approached. Figure 4.17, shows us the time per iteration for the subgradient method compared to the RBFGS algorithm. The time per iteration is small in the RBFGS algorithm. This was be expected since RBFGS simply computes the gradient and does not approximates the subdifferential. The issue is that this difference in time increases with the dimension. It was defined in section 2.3 that the set W_k was build to approximate $\partial_\varepsilon f(x_k)$. We measured the number of vectors added to the set W_k at each iteration. The size of W_k depends on the condition to approximate the subdifferential sufficiently, equation 2.6. We find that in order to sufficiently approximate the subdifferential, the number of vectors in W_k is very large. It seems to behave as the number of sample points needed in the gradient sampling stopping criterion (algorithm 7, which is another context where we sample gradients to approximate the subdifferential). We were not able to detect a pattern, but it requires to be several times larger than the dimension when the dimension is large. For instance with $n = 500$ in the sparse PCA problem, the size of W_k often reaches 2000. Solving minimum norm problems of this size will soon become an issue as the dimension increases. It seems that the first intention was to sample a small number of gradients to approximate the subdifferential and find a subgradient. But to satisfy the condition in equation (2.6) and guarantee convergence, the size of W_k must be large for large n.

As a short conclusion to this section, we found that subgradient methods do not perform well for large scales compared to the RBFGS method. Results are comparable for small scale problems. This is due to the computation of the subgradient that comes at a very high price. The number of gradients that has to be sampled to approach $\partial f(x)$ is too large to find the subgradient efficiently.

Line search

We wondered if the subgradient method would produce directions with a better scaling than RBFGS. So we looked at the number of trials in the line search, as well as the step lengths that were produced by the method. Table 4.10 shows the results. We only show the initial step Heuristic 4 (α_k^{04}, defined in table 4.6), because it gives the best results. We also compare it with the results of RBFGS shown previously. We see that the line search of the subgradient method spent on average 5.78 trials in the line search. This is much more than RBFGS. For the subgradient method, we observed that the unit is very

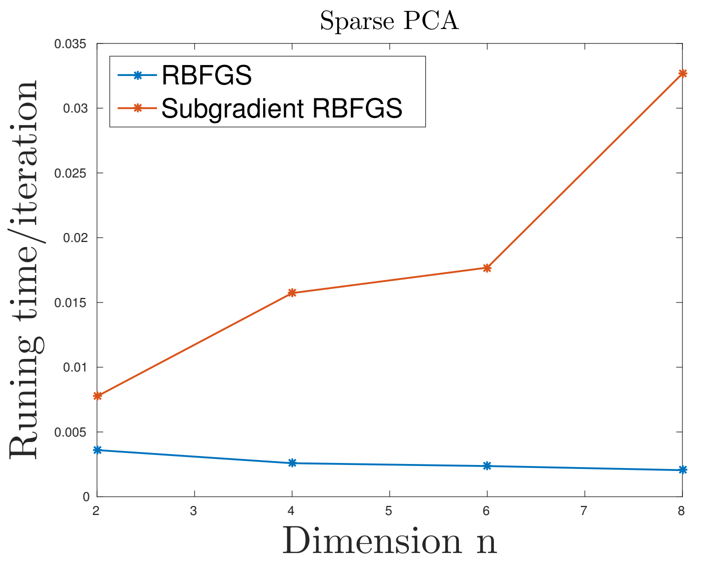


FIGURE 4.17: Computation time per iteration as a function of the dimension for sparse PCA with $\mu = 1 * norm(a_{i^*})$.

Algorithm	Initial stepsize strategy	Iterations	Trials in line search	Time $[s]$
Subgradient RBFGS	α_k^{04}	53.2	5.78	0.09
RBFGS	α_k^{04}	16.6	2.20	0.033

TABLE 4.10: Subgradient RBFGS: Analysis of the best initial step size strategy for OBB: $n = 3$ and $N = 500$. Average values over 100 samples.

rarely tried. This is worst then the RBFGS method where the unit step length was used occasionally at the end of the iterative process.

Limited memory subgradient RBFGS

This method, a subgradient LRBFGS was also considered. What should be remembered is that, since the subgradients methods perform poorly on large dimensions, the limited-memory version is not very useful. Indeed, since we can only solve for small dimensions, this is not a very interesting situation for the subgradient limited memory scheme. The problem of approximating the subdifferential to find a subgradient still holds when we use a limited memory scheme.

4.5 Comparison of Riemannian nonsmooth methods

In this final section we compare all the methods for each test problem.

4.5.1 Orthogonal bounding box

We go back to the preferential case $n = 3$. We have solved 100 problems. Each time a common starting point x_0 and data points were generated randomly. The average measure of efficiency over the sample of size 100 is in table 4.11. The LRBFGS and subgradient LRBFGS methods were used with $m = 2$. In this case, LRBFGS does not do so well compared to RBFGS. The dimension and the number of iteration seems too small to benefit a limited-memory version. Both subgradients methods require more iterations and are notably slower. In conclusion, RBFGS seems to be the best method for the OBB problem with $n = 3$.

Algorithm	iter	nf	ng	nR	Time
RBFGS	12	34	15	33	0.03
LRBFGS	51	151	83	150	0.05
Subgradient RBFGS	50	267	89	266	0.09
Subgradient LRBFGS	44	448	80	447.00	0.13

TABLE 4.11: Comparison of all methods for OBB problem $n = 3$. Average values over 100 problems.

4.5.2 Sparse PCA

For the sparse PCA problem, we are interest in large scales and the best algorithm is the best performer asymptotically, as n grows large. We solved the problem 100 times for $n = 1000$. Higher values become inaccessible for the subgradient methods. The average measures of efficiency are in table 4.12. The LRBFGS and subgradient LRBFGS methods were used with $m = 2$. LRBFGS clearly achieves the lowest total in iter, nf, ng, nR and computation time. Increasing n only makes the advantage of LRBFGS over RBFGS more important. Subgradient methods are slow for such a large dimension. But we note that the number of iteration is also larger.

Algorithm	iter	nf	ng	nR	Time
RBFGS	24	64	43	63	0.08
LRBFGS	16	36	28	35	0.05
Subgradient RBFGS	48	531	429	530	3.03
Subgradient LRBFGS	79	620	233	619	0.65

TABLE 4.12: Comparison of all methods for PCA problem $n = 1000$, average values over 100 problems solved.

The GPower algorithm

For the sparse PCA problem, we will compare the performances of the methods studied against another state of the art algorithm. The generalized power (GPower) method for sparse PCA is defined in [JNRS08]. Their method works as a gradient descent. This algorithm is very specific and was designed for the sparse PCA problem itself. It uses the structure of the cost function advantageously to optimize the formulation of the problem.

We have used the LRBFGS solver with $m = 2$ which is our fastest method for large instances of the PCA problem. The tolerance was set to $\frac{|f(x_k) - f(x_{k-1})|}{|f(x_k)| + 1} < 10^{-6}$. It appears that the GPower method well outperforms LRBFGS. The results are in table 4.13. We averaged the running times of 100 problems solved. For instance at $n = 2000$, LRBFGS solved the problem in 2.25 seconds. While GPower took 0.34 seconds. It is not really surprising that the GPower method is more efficient. This algorithm exploits the structure of the cost function in depth, as reformulations and preprocessing operations are made. While LRBFGS is a "black box" type solver that is much more general. It only uses the function value and the gradient at a given point, ignoring the structure of the function.

Algorithm \ n	250	500	1000	1500	2000
LRBFGS $m = 2$	0.11	0.17	0.58	1.25	2.25
GPower	0.07	0.02	0.08	0.17	0.34

Table 4.13: Running time in seconds. The GPower solver against LRBFGS on sparse PCA problem.

4.5.3 Summary of observations

We conclude this chapter about experimental results with a short summary. We have tried two methods: RBFGS and subgradient RBFGS, as well as their limited-memory variants. The subgradient methods have the very attractive aspect that convergence results exist. Even though further work should investigate whether condition (2.4) on the spectre of the BFGS Hessian approximation holds in practice. As observed, the simpler RBFGS seems more efficient for applications. It appears that counter examples to convergence can only be fabricated with well chosen initial condition. In practice the method converges with probability 1 and seems to produce descent directions only. For the subgradient methods, approaching the subdifferential seems really costly. We are in a situation where we have two methods available. One overall outperforms the other, but the slowest possess convergence results. Both have a linear convergence of the function values towards nonsmooth minimizer. For large scale problems, LRBFGS is the method of choice. With the value $m = 2$ as a good reference. We measured a great gain compared to larger parameters m or with RBFGS. For the OBB problem with $n = 3$, the LRBFGS method does not outperform RBFGS because the dimension and number of iterations seems too small.

Some attention was devoted to the line search.

Conclusion

The starting point of this project was the paper [LO13] on Euclidean nonsmooth BFGS. Their setting of locally Lipschitz functions corresponded to several applications on Riemannian manifolds. The goals were to see if their results could be reproduced in Riemannian optimization. Which we did for the sparse principal component analysis (PCA) and orthogonal bounding box (OBB) problems. We observe that the method converges to a minimizer in practice. Even though there does not seem to exist any current lead to a proof of convergence for this algorithm. The observed rate of convergence to a nonsmooth minimizer is linear. Which was expected based on similar behaviour in $\mathbb{R}^n$ [LO13].

There was also an interest for the behaviour of the line search. For which we provide a complete analysis in the case of our test problems. It seems that there are still some features of nonsmooth BFGS methods which could be better understood. Precisely the scale of the directions generated and the conditioning of the Hessian approximation near a nonsmooth minimizer. The nonsmooth stopping criterion was also studied. Some efficient alternatives to the original algorithm 6 were shown. We can nevertheless conclude that nonsmooth stopping criterion are not easy to use in large dimensions, as they seem forced to approximate the subdifferential. Something we know is expensive for large dimensions.

A second method was then considered, the subgradient RBFGS from [AHHY16]. This algorithm uses a BFGS scheme on a *subgradient oriented* sequence of directions. This method has the notable advantage to come with known convergence results. In practice, we have observed that it is outperformed by RBFGS for large scale problems. As the approximation of the subdifferential comes at a high price.

Even though the main goal was to study the convergence properties of the methods, it is still interesting to conclude by comparing our methods with other existing algorithms for the test problems chosen. For the sparse PCA problem, a comparison with a generalized power method (from [JNRS08]) was conducted. The conclusion is that the specific GPower algorithm outperforms RBFGS and LRBFGS methods. This is not surprising as it uses the structure of the cost function to improve the formulation of the problem. Whereas RBFGS methods do not, they only evaluate the cost function and gradient locally.

Nonsmooth Riemannian optimization is a field in constant evolution. There are numerous techniques being worked on. This work does not pretend to cover all available methods for the problems selected, nor to have given a complete overview of Riemannian nonsmooth algorithms, but hopes to have helped gain insight on the progressing subject of quasi-Newton methods applied to nonsmooth Riemannian optimization.

Bibliography

[AHHY16] P.-A. Absil, S. Hosseini, W. Huang, and R. Yousefpour. *Line search algorithms for locally lipschitz functions on Riemannian manifolds.* Technical report, 2016.

[AMS08] P.-A Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds.* Princeton University Press, 2008.

[AS10] M. L. Overton A. Skajaa. *Master thesis: Limited Memory BFGS for Nonsmooth Optimization.* Courant Institute of Mathematical Science, 2010.

[BA10] Pierre B. Borckmans and P.-A. Absil. *Oriented Bounding Box Computation Using Particle Swarm Optimization.* Conference paper, Department of Mathematical Engineering, Université catholique de Louvain, 2010.

[BKM14] A. Bagirov, N. Karmitsa, and M. M. Mäkelä. *Introduction to Nonsmooth Optimization.* Springer, Theory, Practice and Software, 2014.

[GH15] P. Grohs and S. Hosseini. *ε-subgradient algorithms for locally lipschitz functions on Riemannian manifolds.* Springer, 2015.

[HAG16] Wen Huang, P.-A. Absil, and K. A. Gallivan. *Intrinsic Representation of Tangent Vectors and Vector Transports on Matrix Manifolds.* Technical report UCL-INMA, 2016.

[HGA15] W. Huang, K. A. Gallivan, and P.-A. Absil. A broyden class of quasi-newton methods for riemannian optimization. *SIAM Journal of Optimization*, 2015.

[HU16] S. Hosseini and A. Uschmajew. *A Riemannian gradient sampling algorithm for nonsmooth optimization on manifolds.* University of Bonn, March 2016.

[Hua13] Wen Huang. *Optimization Algorithms on Riemannian Manifolds with Applications.* Electronic Theses, Treatises and Dissertations, 2013.

[JNRS08] M. Journée, Y. Nesterov, P. Richtarik, and R. Sepulchre. *Generalized power method for sparse principal component analysis.* Optimization and Control, 2008.

[LO13] Adrian S. Lewis and Michael L. Overton. *Nonsmooth optimization via quasi-Newton methods.* Mathematical Programming, Springer, 2013.

[M.S13] M. Kleinsteuber M.Seubert. Properties of the bfgs method on riemannian manifolds. *Mathematical System Theory C Festschrift in Honor of Uwe Helmke on the Occasion of his Sixtieth Birthday*, 2013.

[NW00] J. Nocedal and S. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer New York, 2000.

[NW06] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 2006.

[Pow76] M.J.D. Powell. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. *Nonlinear Programming, American Mathematical Society*, 1976. SIAM-AMS Proceedings, vol. IX.

[WH14] K. A. Gallivan W. Huang, P.-A. Absil. *A Riemannian symmetric rank-one trust-region method.* Tech. report UCL-INMA, 2014.