Master's thesis:

Limited Memory BFGS for Nonsmooth Optimization

Anders Skajaa

M.S. student
Courant Institute of Mathematical Science
New York University

January 2010

Adviser:
Michael L. Overton
Professor of Computer Science and Mathematics
Courant Institute of Mathematical Science
New York University

Abstract

We investigate the behavior of seven algorithms when used for nonsmooth optimization. Particular emphasis is put on the BFGS method and its limited memory variant, the LBFGS method. Numerical results from running the algorithms on a range of different nonsmooth problems, both convex and nonconvex, show that LBFGS can be useful for many nonsmooth problems. Comparisons via performance profiles show that for large-scale problems – its intended use – it compares very well against the only other algorithm for which we have an implementation targeted at that range of problems. For small- and medium-scale nonsmooth problems, BFGS is a very robust and efficient algorithm and amongst the algorithms tested, there is an equally robust, but somewhat less efficient alternative.

Contents

1 INTRODUCTION

The field of unconstrained optimization is concerned with solving the problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1.1}$$

i.e. finding an $x^\star \in \mathbb{R}^n$ that minimizes the *objective function* f. Analytically finding such an $x^\star$ is generally not possible so iterative methods are employed. Such methods generate a sequence of points $\{x^{(j)}\}_{j \in \mathbb{N}}$ that hopefully converges to a minimizer of f as $j \to \infty$. If the function f is convex, that is

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \tag{1.2}$$

for all $x, y \in \mathbb{R}$ and all $\lambda \in [0, 1]$, then all local minimizers of f are also global minimizers [BV04]. But if f is not convex, a local minimizer approximated by an iterative method may not be a global minimizer. If f is continuously differentiable then (1.1) is called a smooth optimization problem. If we drop this assumption and only require that f be continuous, then the problem is called *nonsmooth*. We are interested in algorithms that are suitable for nonconvex, nonsmooth optimization. There is a large literature on convex nonsmooth optimization algorithms; see particularly [HUL93]. The literature for the nonconvex case is smaller; an important early reference is [Kiw85].

Recently, Lewis and Overton [LO10] have shown in numerical experiments that the standard BFGS method works very well when applied directly without modifications to nonsmooth test problems as long as a weak Wolfe line search is used. A natural question is then if the standard limited memory variant (LBFGS) works well on large-scale nonsmooth test problems. In this thesis we will, through numerical experiments, investigate the behavior of LBFGS when applied to small-, medium- and large-scale nonsmooth problems.

In section 2 we give the motivation for BFGS and LBFGS for smooth optimization. In section 3, we discuss a line search suitable for nonsmooth optimization. In section 4 we show the performance of LBFGS when applied to a series of illustrative test problems and in section 5 we compare seven optimization algorithms on a range of test problems, both convex and nonconvex.

A website[1] with freely available MATLAB-code has been developed. It contains links and files for the algorithms, MATLAB-files for all the test problems and scripts that run all the experiments. The site is still under development.

[1] http://www.student.dtu.dk/~s040536/nsosite/webfiles/index.shtml

2 METHODS FOR SMOOTH OPTIMIZATION

In the first part of this section we assume that f is continuously differentiable.

2.1 NEWTON'S METHOD

At iteration j of any *line search method* a search direction $d^{(j)}$ and a step length α_j are computed. The next iterate $x^{(j+1)}$ is then defined by

$$x^{(j+1)} = x^{(j)} + \alpha_j d^{(j)}. \tag{2.1}$$

Line search methods differ by how $d^{(j)}$ and α_j are chosen.

In *Newton's method* the search direction is chosen by first assuming that the objective function may be well approximated by a quadratic around $x^{(j)}$:

$$f(x^{(j)}+d^{(j)}) \approx f(x^{(j)}) + (d^{(j)})^T \nabla f(x^{(j)}) + \frac{1}{2}(d^{(j)})^T \nabla^2 f(x^{(j)}) d^{(j)} =: \mathcal{T}_j(d^{(j)}) \tag{2.2}$$

Looking for a minimizer of the function on the right hand side of (2.2), we solve $\nabla \mathcal{T}_j(d^{(j)}) = 0$ and obtain

$$d^{(j)} = -\left[\nabla^2 f(x^{(j)})\right]^{-1} \nabla f(x^{(j)}), \tag{2.3}$$

assuming that $\nabla^2 f(x^{(j)})$ is positive definite. Equation (2.3) defines the *Newton step*. If f itself is a quadratic function, (2.2) would be exact and by applying (2.1) once with $\alpha_j = 1$, we would have minimized f in one iteration. If f is not quadratic we can apply (2.1) iteratively with $d^{(j)}$ defined by (2.3).

The step length $\alpha_j = 1$ is, in a way, natural for Newton's method because it would take us to the minimizer of the quadratic function locally approximating f. In a neighborhood of a unique minimizer $x^\star$ the Hessian must be positive definite. Hence most implementations of Newton-like methods first try $\alpha_j = 1$ and only choose something else if the reduction in f is not sufficient.

One can show that, using a line search method as described in section 3, the iterates converge to stationary points (usually a local minimizer $x^\star$). Further, once the iterates come sufficiently close to $x^\star$, the convergence is *quadratic* under standard assumptions [NW06]. Although quadratic convergence is a very desirable property there are several drawbacks to Newton's method. First, the Hessian in each iterate is needed. In addition to this, we must at each iterate find $d^{(j)}$ from (2.3) which requires the solution of a linear system of equations - an operation requiring $\mathcal{O}(n^3)$ operations. This means we can use Newton's method only to solve problems with the number of variables up to one thousand or so, unless the Hessian is sparse. Finally, Newton's method needs modifications if $\nabla^2 f(x^{(j)})$ is not positive definite [NW06].

2.2 Quasi-Newton Methods

A standard alternative to Newton's method is a class of line search methods where the search direction is defined by

$$d^{(j)} = -C_j \nabla f(x^{(j)}) \tag{2.4}$$

where C_j is updated in each iteration by a *quasi-Newton* updating formula in such a way that it has certain properties of the inverse of the true Hessian.

As long as C_j is symmetric positive definite, we have $(d^{(j)})^T \nabla f(x^{(j)}) < 0$, that is $d^{(j)}$ is a *descent direction*. If we take $C_j = I$ (the identity matrix), the method reduces to the *steepest descent method*.

By comparing the quasi-Newton step (2.4) to the Newton step (2.3), we see that the two are equal when C_j is equal to the inverse Hessian. So taking a quasi-Newton step is the same as minimizing the quadradic function (2.2) with $\nabla^2 f(x^{(j)})$ replaced by $B_j := C_j^{-1}$. To update this matrix we impose the well known secant equation:

$$B_{j+1}(\alpha_j d^{(j)}) = \nabla f(x^{(j+1)}) - \nabla f(x^{(j)}) \tag{2.5}$$

If we set

$$s^{(j)} = x^{(j+1)} - x^{(j)} \quad \text{and} \quad y^{(j)} = \nabla f(x^{(j+1)}) - \nabla f(x^{(j)}) \tag{2.6}$$

equation (2.5) becomes

$$B_{j+1} s^{(j)} = y^{(j)} \tag{2.7}$$

or equivalently

$$C_{j+1} y^{(j)} = s^{(j)}. \tag{2.8}$$

This requirement, together with the requirement that C_{j+1} be symmetric positive definite, is not enough to uniquely determine C_{j+1}. To do that we further require that

$$C_{j+1} = \text{argmin}_C \|C - C_j\| \tag{2.9}$$

i.e. that C_{j+1}, in the sense of some matrix norm, be the closest to C_j among all symmetric positive definite matrices that satisfy the secant equation (2.8). Each choice of matrix norm gives rise to a different update formula.

2.3 BFGS

The most popular update formula is

$$C_{j+1}^{\mathsf{BFGS}} = \left(I - \rho_j s^{(j)} (y^{(j)})^T\right) C_j \left(I - \rho_j y^{(j)} (s^{(j)})^T\right) + \rho_j s^{(j)} (s^{(j)})^T \tag{2.10}$$

where $\rho_j = \left((y^{(j)})^T s^{(j)}\right)^{-1}$. Notice that the computation in (2.10) requires only $\mathcal{O}(n^2)$ operations because $C_j \left(I - \rho_j y^{(j)} (s^{(j)})^T\right) = C_j - \rho_j (C_j y^{(j)})(s^{(j)})^T$,

Algorithm 1 BFGS

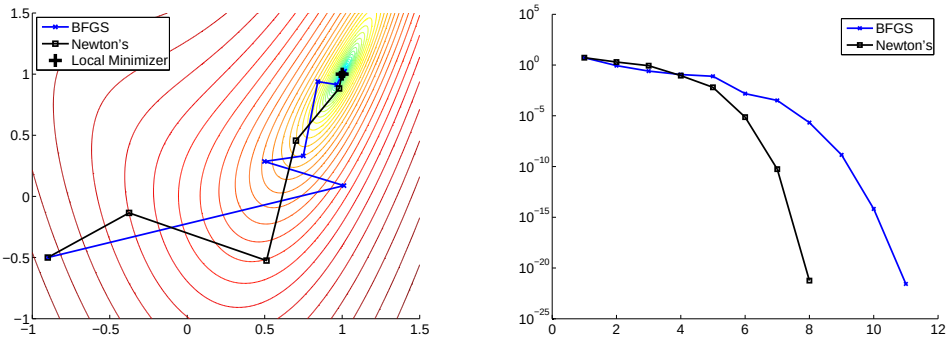
Input: $x^{(0)}$, δ, C_0
  $j \leftarrow 0$
  while true do
    $d^{(j)} \leftarrow -C_j \nabla f(x^{(j)})$
    $\alpha_j \leftarrow \text{LineSearch}(x^{(j)}, f)$
    $x^{(j+1)} \leftarrow x^{(j)} + \alpha_j d^{(j)}$
    Compute C_{j+1} from (2.10) and (2.6)
    $j \leftarrow j + 1$
    if $\|\nabla f(x^{(j)})\| \leq \delta$ **then**
      stop
    end if
  end while
Output: $x^{(j)}$, $f(x^{(j)})$ and $\nabla f(x^{(j)})$.

a computation that needs one matrix-vector multiplication, one vector outer product and one matrix addition.

BFGS[2] is currently considered the most effective and is by far the most popular quasi-Newton update formula. The BFGS algorithm is summed up in Algorithm 1 [NW06].

The success of the BFGS algorithm depends on how well the updating formula for C_j approximates the inverse of the true Hessian at the current iterate. Experiments have shown that the method has very strong self-correcting properties (when the right line search is used) so that if, at some iteration, the matrix contains bad curvature information, it often takes only a few updates to correct these inaccuracies. For this reason, the BFGS method generally works very well and once close to a minimizer, it usually attains superlinear convergence. A simple comparison of the BFGS method and Newton's method is seen in figure 1 on the next page. We see that while Newton's method almost immediately attains quadratic convergence, BFGS needs more iterations, eventually achieving superlinear convergence. For small problems like this one, it is, of course, not expensive to solve a linear system which is needed in Newton's method. BFGS requires only matrix-vector multiplications which brings the computational cost at each iteration from $\mathcal{O}(n^3)$ for Newton's method down to $\mathcal{O}(n^2)$. However, if the number of variables is very large, even $\mathcal{O}(n^2)$ per iteration is too expensive - both in terms of CPU time and sometimes also in terms of memory usage (a large matrix must be kept in memory at all times).

[2]Named after its inventors: Broyden, Fletcher, Goldfarb and Shanno.



(a) *Paths followed.* (b) *Function value at each iterate.*

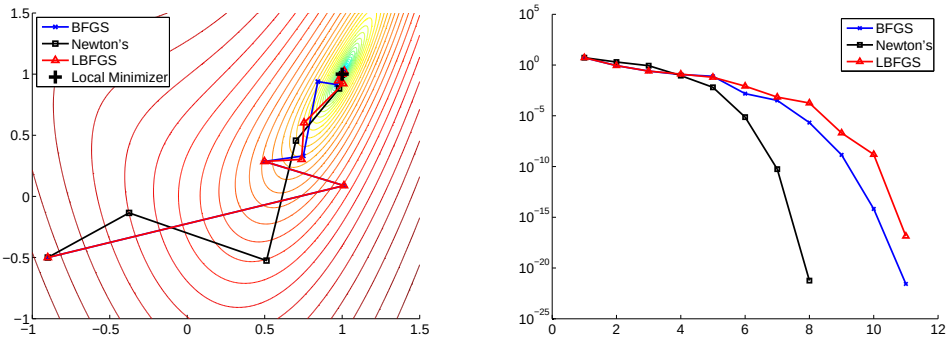
FIGURE 1: *Results from running BFGS (blue) and Newton's method (black) on the smooth 2D Rosenbrock function* $f(x) = (1-x_1)^2+(x_2-x_1^2)^2$ *with* $x^{(0)} = (-0.9, -0.5)$. *The minimizer is* $x^\star = (1, 1)$ *and* $f(x^\star) = 0$.

2.4 LIMITED MEMORY BFGS

A less computationally intensive method when n is large is the *Limited-Memory BFGS method* (LBFGS), see [Noc80, NW06]. Instead of updating and storing the entire approximated inverse Hessian C_j, the LBFGS method never explicitly forms or stores this matrix. Instead it stores information from the past m iterations and uses only this information to implicitly do operations requiring the inverse Hessian (in particular computing the next search direction). The first m iterations, LBFGS and BFGS generate the same search directions (assuming the initial search directions for the two are identical and that no scaling is done – see section 5.1.5). The updating in LBFGS is done using just $4mn$ multiplications (see Algorithm 2 [NW06]) bringing the computational cost down to $\mathcal{O}(mn)$ per iteration. If $m \ll n$ this is effectively the same as $\mathcal{O}(n)$. As we shall see later, often LBFGS is successful with $m \in [2, 35]$ even when $n = 10^3$ or larger. It can also be argued that the LBFGS method has the further advantage that it only

Algorithm 2 Direction finding in LBFGS

$q \leftarrow \gamma_j \nabla f(x^{(j)})$, with $\gamma_j = ((s^{(j-1)})^T y^{(j-1)})((y^{(j-1)})^T y^{(j-1)})^{-1}$
for $i = (j-1) : (-1) : (j-m)$ **do**
    $\alpha_i \leftarrow \rho_i (s^{(i)})^T q$
    $q \leftarrow q - \alpha_i y^{(i)}$
end for
for $i = (j-m) : 1 : (j-1)$ **do**
    $\beta \leftarrow \rho_i (y^{(i)})^T r$
    $r \leftarrow r + s^{(i)}(\alpha_i - \beta)$
end for
Output: $d^{(j)} = -r$



(a) *Paths followed.* (b) *Function value at each iterate.*

FIGURE 2: *Results from running BFGS (blue), LBFGS with $m = 3$ (red), and Newton's method (black) on the smooth 2D Rosenbrock function $f(x) = (1 - x_1)^2 + (x_2 - x_1^2)^2$ with $x^{(0)} = (-0.9, -0.5)$. The minimizer is $x^\star = (1, 1)$ and $f(x^\star) = 0$.*

uses relatively *new* information. In the BFGS method, the inverse Hessian contains information from *all* previous iterates. This may be problematic if the objective function is very different in nature in different regions.

In some cases the LBFGS method uses as many or even fewer function evaluations to find the minimizer. This is remarkable considering that even when using the same number of function evaluations, LBFGS runs significantly faster than full BFGS if n is large.

In figure 2 we see how LBFGS compares to BFGS and Newton's method on the same problem as before. We see that for this particular problem, using $m = 2$, LBFGS performs almost as well as the full BFGS.

Experiments show that the optimal choice of m is problem dependent which is a drawback of the LBFGS method. In case of LBFGS failing, one should first try to increase m before completely discarding the method. In very few situations (as we will see later on a nonsmooth example) the LBFGS method may need $m > n$ to converge, in which case LBFGS is more expensive than regular BFGS.

	Newton's	BFGS	LBFGS
Work per iteration	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(mn)$

2.5 APPLICABILITY TO NONSMOOTH OPTIMIZATION

Now suppose that the objective function f is not differentiable everywhere, in particular that it is not differentiable at a minimizer $x^\star$.

In Newton and quasi-Newton methods, we assumed that f may be well approximated by a quadratic function in a region containing the current iterate. If we are close to a point of nonsmoothness, this assumption no longer holds.

When f is locally Lipschitz, we have by Rademacher's theorem [Cla83] that f is differentiable almost everywhere. Roughly speaking, this means that the probability that an optimization algorithm that is initialized randomly will encounter a point where f is not differentiable is zero. This is what allows us to directly apply optimization algorithms originally designed for smooth optimization to nonsmooth problems. In fact we are *assuming* that we never encounter such a point throughout the rest of this thesis. This ensures that the methods used stay well defined.

Simple examples show that the steepest descent method may converge to nonoptimal points when f is nonsmooth [HUL93, LO10] and Newton's method is also unsuitable when f is nonsmooth.

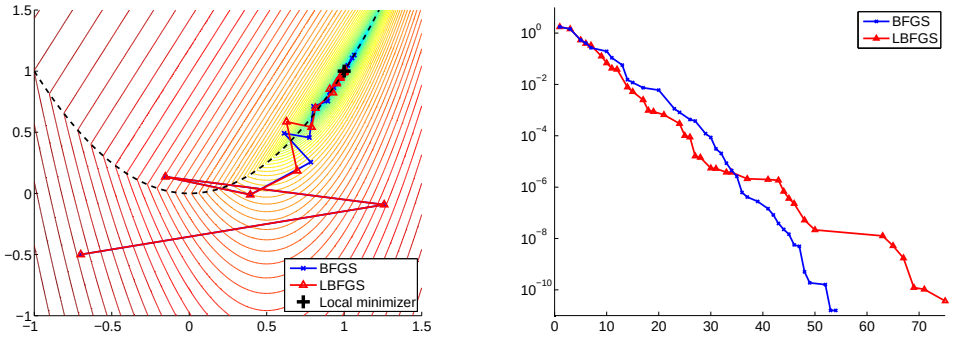
2.6 Quasi-Newton Methods in the Nonsmooth Case

Although standard quasi-Newton methods were developed for *smooth* optimization it turns out [LO10] that they often succeed in minimizing nonsmooth objective functions when applied directly without modification (when the right line search is used — see section 3).

In figure 3 we see the performance of BFGS and LBFGS with a weak Wolfe line search when applied directly to the function

$$f(x) = (1 - x_1)^2 + |x_2 - x_1^2| \tag{2.11}$$

This function is not smooth at the local minimizer $x^\star = (1, 1)$. In figure 3(b) we see that both methods succeed in finding the minimizer. Comparing the rate of convergence to what we saw in figure 2 on the preceding page where we applied BFGS to the smooth Rosenbrock function, we notice that



(a) *Paths followed. The black dashed line indicates the curve $x_2 = x_1^2$ across which f is nonsmooth.*

(b) *Function value at each iterate after the line search.*

FIGURE 3: *Results from running BFGS (blue) and LBFGS with $m = 3$ (red) on the 2D nonsmooth Rosenbrock function (2.11) with $x^{(0)} = (-0.7, -0.5)$. The local minimizer $x^\star = (1, 1)$ is marked by a black dot and $f(x^\star) = 0$. LBFGS used 76 function evaluations and BFGS used 54 to reduce f below 10^{-10}.*

the convergence is now approximately linear instead of superlinear. This is due to the nonsmoothness of the problem.

Most of the rest of this thesis is concerned with investigating the behavior of quasi-Newton methods - in particular the LBFGS method - when used for nonsmooth (both convex and nonconvex) optimization. We will show results from numerous numerical experiments and we will compare a number of nonsmooth optimization algorithms. First, however, we address the issue of the line search in more detail.

3 LINE SEARCH

Once the search direction in a line search method has been found, a procedure to determine how long a step should be taken is needed (see (2.1)).

Assuming the line search method has reached the current iterate $x^{(j)}$ and that a search direction $d^{(j)}$ has been found, we consider the function

$$\phi(\alpha) = f(x^{(j)} + \alpha d^{(j)}), \quad \alpha \geq 0 \tag{3.1}$$

i.e. the value of f from $x^{(j)}$ in the direction of $d^{(j)}$. Since $d^{(j)}$ is a descent direction, there exists $\hat{\alpha} > 0$ small enough that $\phi(\hat{\alpha}) < f(x^{(j)})$. However, for a useful procedure, we need steps ensuring sufficient decrease in the function value and steps that are not too small. If we are using the line search in a quasi-Newton method, we must also ensure that the approximated Hessian matrix remains positive definite.

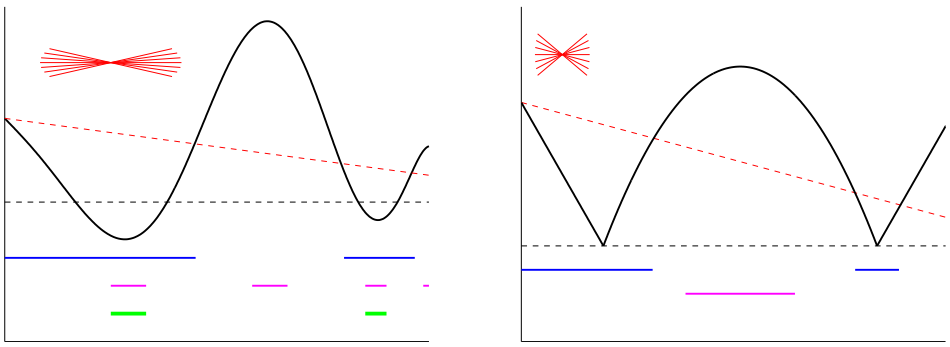
3.1 STRONG WOLFE CONDITIONS

The *Strong Wolfe* conditions require that the following two inequalities hold for α_j:

$$\phi(\alpha_j) \leq \phi(0) + \alpha_j c_1 \phi'(0) \qquad \text{Armijo condition} \tag{3.2}$$

$$|\phi'(\alpha_j)| \leq c_2 |\phi'(0)| \qquad \text{Strong Wolfe condition} \tag{3.3}$$

with $0 < c_1 < c_2 < 1$. The Armijo condition ensures a sufficient decrease in the function value. Remembering that $\phi'(0)$ is negative (because $d^{(j)}$ is a descent direction), we see that the Armijo condition requires that the decrease be greater if α_j is greater.



(a) *Smooth* $\phi(\alpha)$. (b) *nonsmooth* $\phi(\alpha)$.

FIGURE 4: *Steps accepted by the strong Wolfe conditions. Dashed red line is the upper bound on $\phi(\alpha)$ from the Armijo condition. Red lines in upper left hand corner indicate accepted slopes $\phi'(\alpha)$ by the strong Wolfe condition. Lower blue line indicates points accepted by Armijo. Magenta line indicates points accepted by strong Wolfe. Green line indicates points accepted by both.*

The strong Wolfe condition ensures that the derivative $\phi'(\alpha_j)$ is reduced in absolute value. This makes sense for smooth functions because the derivative at a minimizer of $\phi(\alpha)$ would be zero. Figure 4 shows the intervals of step lengths accepted by the strong Wolfe line search.

It is clear that the strong Wolfe condition is not useful for nonsmooth optimization. The requirement (3.3) is bad because for nonsmooth functions, the derivative near a minimizer of $\phi(\alpha)$ need not be small in absolute value (see figure 4(b) on the previous page).

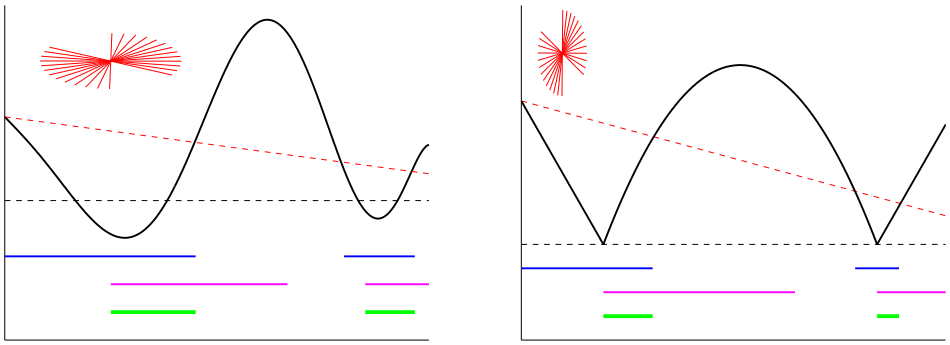
3.2 WEAK WOLFE CONDITIONS

The *Weak Wolfe* conditions are

$$\phi(\alpha_j) \leq \phi(0) + \alpha_j c_1 \phi'(0) \qquad \text{Armijo condition} \qquad (3.4)$$

$$\phi'(\alpha_j) \geq c_2 \phi'(0) \qquad \text{Weak Wolfe condition} \qquad (3.5)$$

with $0 < c_1 < c_2 < 1$. The only difference from the strong Wolfe condition is that there is no longer an upper bound on the derivative $\phi'(\alpha_j)$. The same lower bound still applies. This makes a significant difference in the case of a nonsmooth $\phi(\alpha)$ as seen by comparing figure 4 to figure 5. When $\phi(\alpha)$ is nonsmooth the absolute value of the derivative $\phi'(\alpha)$ may never get small enough to satisfy the strong Wolfe condition (see figure 4(b) on the previous page). Therefore the weak Wolfe conditions are better suited for line search methods for nonsmooth optimization. The weak Wolfe condition is all that is needed to ensure that C_{j+1} in Algorithm 1 is positive definite [NW06].



(a) *Smooth* $\phi(\alpha)$. (b) *nonsmooth* $\phi(\alpha)$.

FIGURE 5: *Steps accepted by the weak Wolfe conditions. Dashed red line is the upper bound on $\phi(\alpha)$ from the Armijo condition. Red lines in upper left hand corner indicate accepted slopes $\phi'(\alpha)$ by the weak Wolfe condition. Lower blue line indicates points accepted by Armijo. Magenta line indicates points accepted by weak Wolfe. Green line indicates points accepted by both. Compare with figure 4 on the previous page.*

3.3 BRACKETING ALGORITHM

An effective algorithm [Lem81, LO10] for finding points satisfying the weak Wolfe conditions is given in algorithm 3.

This procedure generates a sequence of nested intervals in which there are points satisfying the weak Wolfe conditions. In [LO10] it is proved that a weak Wolfe step is always found as long as a point where f is not differentiable is never encountered. As mentioned, this is very unlikely, so for all practical purposes the procedure always returns an α that is a weak Wolfe step (unless rounding errors interfere).

We use this line search procedure with $c_1 = 10^{-4}$ and $c_2 = 0.9$ for the BFGS and the LBFGS implementations throughout this thesis.

Algorithm 3 Weak Wolfe Line Search

Input: ϕ and ϕ'
  $\alpha := 1$, $\mu := 0$, $\nu := \infty$
  while true do
    if $\phi(\alpha) > \phi(0) + \alpha c_1 \phi'(0)$ **then** {*Armijo (3.4) fails*}
      $\nu := \alpha$
    else if $\phi'(\alpha) < c_2 \phi'(0)$ **then** {*Weak Wolfe (3.5) fails*}
      $\mu := \alpha$
    else {*Both (3.4) and (3.5) hold so stop*}
      stop
    end if
    if $\nu < \infty$ **then**
      $\alpha := (\mu + \nu)/2$
    else
      $\alpha := 2\alpha$
    end if
  end while
Output: α

4 LBFGS and BFGS for Nonsmooth Optimization

That the LBFGS method can be used for nonsmooth optimization may be surprising at first since it was originally developed as a limited memory version (hence well suited for large-scale problems) of the *smooth* optimization algorithm BFGS. We here present results showing that LBFGS, in fact, works well for a range of nonsmooth problems. This should be seen as an extension and large-scale version of the numerical results in [LO10] where the full BFGS method is tested on the same problems.

4.1 Definitions

4.1.1 Random starting points

Whenever we speak of a *random* starting point, we mean a starting point $x^{(0)} \in \mathbb{R}^n$ drawn randomly from the uniform distribution on $[-1, 1]^n$. To do this we use the built in function `rand` in Matlab. Exceptions to this rule will be mentioned explicitly.

4.1.2 Rate of convergence

From this point on, when we refer to the *rate of convergence* of an algorithm when applied to a problem, we mean the *R-linear* rate of convergence of the *error in function value*. Using the terminology of [NW06], this means there exist constants $C > 0$ and $r \in (0, 1)$ such that

$$|f_k - f^\star| \leq Cr^k \tag{4.1}$$

The number r is called the *rate of convergence*. Since equation (4.1) means that

$$\log |f_k - f^\star| \leq \log C + k \log r$$

we see that for a rate of convergence r, there is a line with slope $\log r$ bounding the numbers $\{\log |f_k - f^\star|\}$ above. To estimate the rate of convergence of f_k with respect to a sequence n_k, we do a least squares fit to the points $(n_k, \log |f_k - f^\star|)$ and the slope of the optimal line is then $\log r$.

To get a picture of the performance of the algorithms in terms of a useful measure, we actually use for f_k the function value *accepted by the line search* at the end of iteration k and for n_k the *total number of function evaluations used* - including those used by the line search - at the end of iteration k.

For r close to 1, the algorithm converges slowly while it is fast for small r. For this reason we will, when showing rates of convergence, always plot the quantity $-\log(1 - r)$, whose range is $(0, \infty)$ and which will be large if the algorithm is slow and small if the algorithm is fast.

4.1.3 V- AND U-SPACES

Many nonsmooth functions are *partly smooth* [Lew02]. This concept leads to the notions of the U- and V-spaces associated with a point – most often the minimizer. In the convex case, this notion was first discussed in [LOS00].

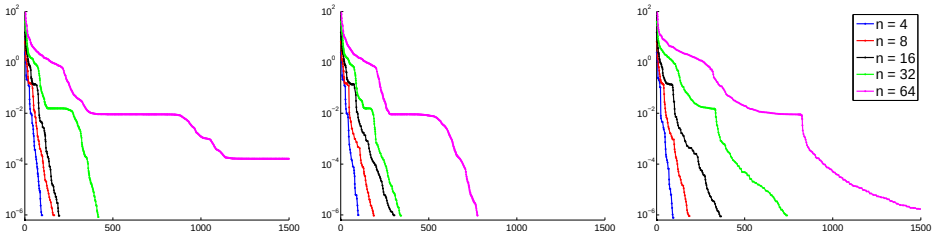
Roughly this concept may be described as follows: Let $\mathcal{M}$ be a manifold containing x that is such that f restricted to $\mathcal{M}$ is twice continuously differentiable. Assume that $\mathcal{M}$ is chosen to have maximal dimension, by which we mean that its tangent space at x has maximal dimension (n in the special case that f is a smooth function). Then the subspace tangent to $\mathcal{M}$ at x is called the U-space of f at x and its orthogonal complement the V-space. Loosely speaking, we think of the U-space as the subspace spanned by the directions along which f varies smoothly at x and the V-space as its orthogonal complement. This means that $\psi(t) = f(x + ty)$ varies smoothly around $t = 0$ only if the component of y in the V-space of f at x vanishes. If not, ψ varies nonsmoothly around $t = 0$.

Consider the function $f(x) = \|x\|$. The minimizer is clearly $x = 0$. Let y be any unit length vector. Then $\psi(t) = f(ty) = |t|\|y\| = |t|$ which varies nonsmoothly across $t = 0$ regardless of y. The U-space of f at $x = 0$ is thus $\{0\}$ and the V-space is $\mathbb{R}^n$.

As a second example, consider the nonsmooth Rosenbrock function (2.11). If $x_2 = x_1^2$, the second term vanishes so when restricted to the manifold $\mathcal{M} = \{x : x_2 = x_1^2\}$, f is smooth. The subspace tangent to $\mathcal{M}$ at the local minimizer $(1, 1)$ is $U = \{x : x = (1, 2)t,\ t \in \mathbb{R}\}$ which is one-dimensional. The orthogonal complement is $V = \{x : x = (-2, 1)t,\ t \in \mathbb{R}\}$ which is also one-dimensional.

In [LO10] it is observed that running BFGS produces information about the $U-$ and V-spaces at the minimizer. At a point of nonsmoothness the gradients jump discontinuously. However at a point of nonsmoothness we can approximate the objective function arbitrarily well by a smooth function. Close to the point of nonsmoothness, the Hessian of that function would have extremely large eigenvalues to accomodate the rapid change in the gradients. Numerically there is no way to distinguish such a smooth function from the actual nonsmooth objective function (assuming the smooth approximation is sufficiently exact).

When running BFGS, an approximation to the inverse Hessian is continuously updated and stored. Very large curvature information in the Hessian of the objective corresponds to very small eigenvalues of the inverse Hessian. Thus, it is observed in [LO10] that one can obtain information about the $U-$ and V-spaces at the minimizer by simply monitoring the eigenvalues of the inverse Hessian approximation C_j when the iterates get close to the minimizer.



(a) *LBFGS with* $m = 15$. (b) *LBFGS with* $m = 25$. (c) *BFGS*.

FIGURE 6: *LBFGS and BFGS on (4.2) for different* n. *Vertical axis: Function value at the end of each iteration. Horizontal axis: Total number of function evaluations used. The number of updates used in LBFGS was* $m = 25$ *and* $x^{(0)} = (1, \ldots, 1)$.

4.2 LBFGS AND BFGS ON FIVE ACADEMIC PROBLEMS

4.2.1 TILTED NORM FUNCTION

We consider the convex function

$$f(x) = w\|Ax\| + (w-1)e_1^T Ax \tag{4.2}$$

where e_1 is the first unit vector, $w = 4$ and A is a randomly generated symmetric positive definite matrix with condition number n^2. This function is not smooth at its minimizer $x^\star = 0$.

In figure 6 we see typical runs of LBFGS and BFGS on (4.2). It is clear from figures 6(a) and 6(b) that the performance of LBFGS on this problem depends on the number of updates m used. We will investigate this further in section 4.4.

Comparing the three plots in figure 6 we see that the linear rate of convergence varies differently with n for LBFGS and BFGS. This is also what we see in figure 7, where we have shown observed rates of convergence for ten runs with random $x^{(0)}$ for different n. Since all runs were successful for both LBFGS and BFGS, ten red and ten blue crosses appear on each vertical line. The curves indicate the mean of the observed rates.

We see that for smaller n, LBFGS generally gets better convergence rates but gets relatively worse when n increases.

4.2.2 A CONVEX, NONSMOOTH FUNCTION

We consider the function

$$f(x) = \sqrt{x^T Ax} + x^T Bx \tag{4.3}$$

where we take

$$A = \left(\begin{array}{c|c} M & 0 \\ \hline 0 & 0 \end{array}\right) \tag{4.4}$$

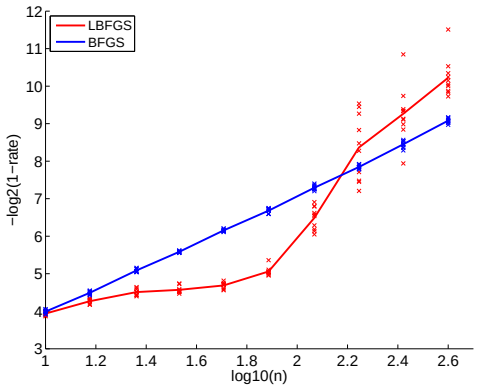
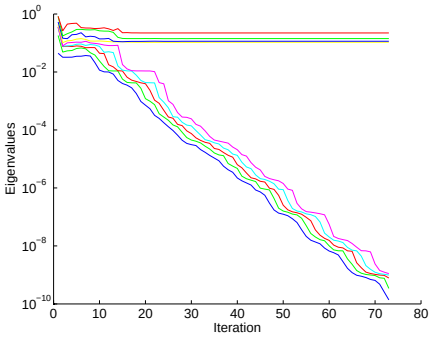


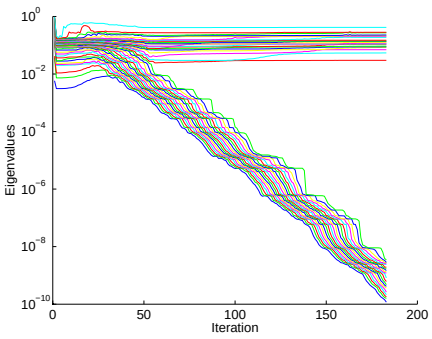
FIGURE 7: *Rates of convergence for BFGS and LBFGS on (4.2) as function of n for ten runs with randomly chosen $x^{(0)}$ and cond$(A) = n^2$. Number of updates for LBFGS was $m = 25$.*

and where $M \in \mathbb{R}^{\lceil n/2 \rceil \times \lceil n/2 \rceil}$ is a randomly generated symmetric positive definite matrix with condition number $\lceil n/2 \rceil^2$ and $B \in \mathbb{R}^{n \times n}$ is a randomly generated symmetric positive definite matrix with condition number n^2.

First notice that for $y \in \text{Null}(A)$, the function $h(t) = f(ty)$ varies smoothly with t around $t = 0$ since the first term of (4.3) vanishes, leaving only the smooth term $t^2 y^T B y$. The null space of A is a linear space so the U-space of f at 0 is the null space of A and the V-space is the range of A. Since we have chosen A such that $\dim(\text{Null}(A)) = \lceil n/2 \rceil$, we expect to see $\lceil n/2 \rceil$ of the eigenvalues of the approximated inverse Hessian constructed in BFGS go to zero and this is exactly what we see in experiments (see figure 8). In figure 9 on the following page we show the convergence rates of LBFGS and BFGS as a function of the number of variables. We see that for this



(a) $n = 9$. *Exactly 5 eigenvalues go to zero.*



(b) $n = 40$. *Exactly 20 eigenvalues go to zero.*

FIGURE 8: *Eigenvalues of the approximation to the inverse Hessian constructed in the course of running BFGS on (4.3).*

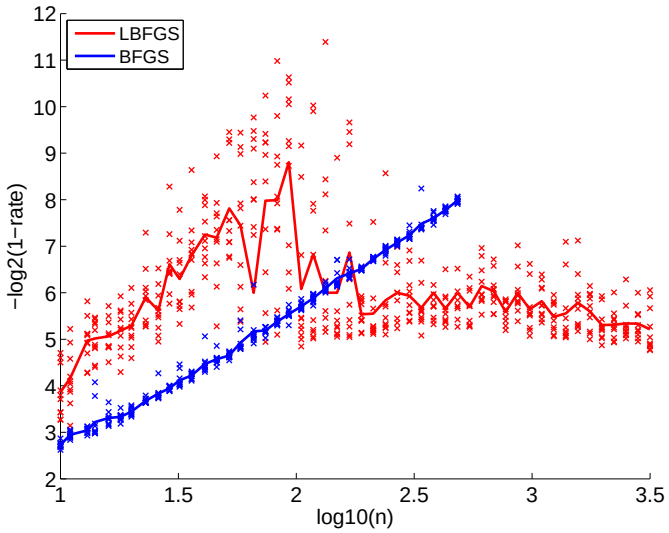


FIGURE 9: *Rates of convergence for BFGS and LBFGS on (4.3) as function of n for ten runs with randomly chosen $x^{(0)}$. Since all runs were successful, ten red and ten blue crosses appear on all vertical lines.*

problem, BFGS gets better rates of convergence for smaller n, while LBFGS relatively does better and better as n grows. Overall, the behavior of LBFGS on this problem is much more erratic than that of BFGS.

As an illustration of the difference in running times of the two algorithms, see the data in table 1.

alg.\n	10	130	250	370	490
LBFGS	0.0392	0.2077	0.2809	0.3336	0.4378
BFGS	0.0099	0.2013	0.8235	3.0115	7.2507

TABLE 1: *Total time in seconds spent by the two algorithms solving (4.3) for different number of variables n.*

4.2.3 A NONCONVEX, NONSMOOTH FUNCTION

We consider the function

$$f(x) = \sqrt{\delta + \sqrt{x^T A x} + x^T B x} \tag{4.5}$$

where A and B are chosen as in (4.3). As described in [LO10], this function is nonconvex for $\delta < 1$ and its Lipschitz constant is $\mathcal{O}(\delta^{-1/2})$ as $x \to 0^+$. The unique minimizer is $x^\star = 0$ and $f(x^\star) = \sqrt{\delta}$.

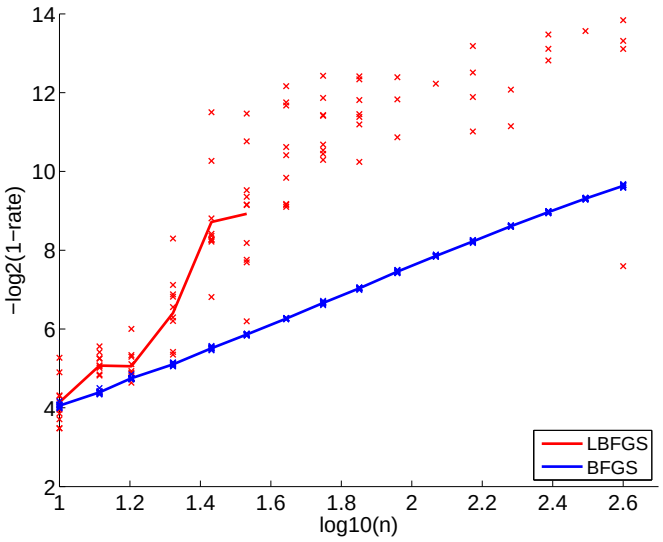


FIGURE 10: *Rates of convergence for BFGS and LBFGS on (4.5) as function of n for ten runs with randomly chosen $x^{(0)}$. Number of updates used for LBFGS was $m = 35$. All BFGS runs were successful, but most LBFGS failed. Red crosses were plotted for each successful run.*

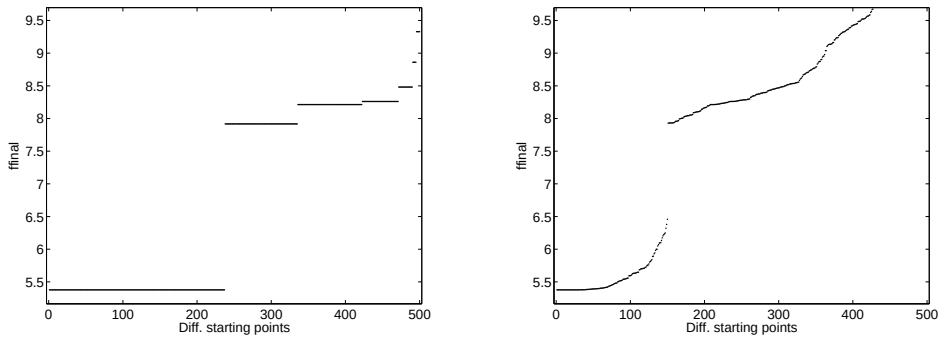
We fix $\delta = 10^{-3}$ and investigate how LBFGS and BFGS perform when varying the number of variables n. We get the situation illustrated in figure 10. On this problem we see that LBFGS has real difficulties. For the few runs that are actually successful, it needs on the order of 10^4 function evaluations to drive the function value below f_{target} (see (4.8) on page 23). BFGS performs very similarly on this problem as on (4.3). For $n \geq 100$, LBFGS fails on average on about 80% of the runs effectively making it useless on this problem.

4.2.4 A GENERALIZED NONSMOOTH ROSENBROCK FUNCTION

When f is smooth and nonconvex, it may have multiple local minimizers and saddle points. While BFGS and LBFGS may certainly converge to non-global local minimizers, they are not likely to converge to other stationary points, that is points x where $\nabla f(x) = 0$ but x is not a local minimizer (although in principle this could happen). When f is nonsmooth, the appropriate generalization is *Clarke stationarity*, that is points x where $0 \in \partial f(x)$, where $\partial f(x)$ is the Clarke subdifferential or generalized gradient [Cla83, BL00, RW98].

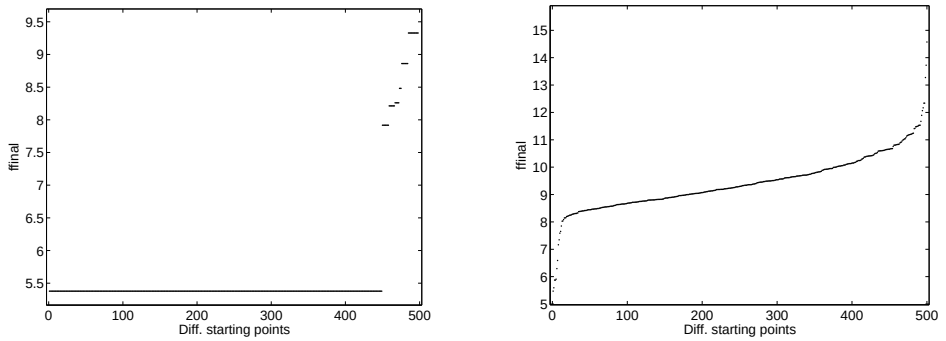
We consider the function

$$f(x) = \sum_{i=1}^{n-1} V\frac{i}{n}\left|x_{i+1} - \frac{i}{n}x_i^2\right| + U\frac{i}{n}(1 - x_i)^2 \tag{4.6}$$



(a) *BFGS* (b) *LBFGS with* $m = 30$

FIGURE 11: *Function value at termination of LBFGS and BFGS on (4.6) for 500 randomly started runs.*



(a) *ShorRLesage* (b) *LMBM with* $m = 30$

FIGURE 12: *Function value at termination of ShorRLesage and LMBM on (4.6) for 500 randomly started runs.*

which is nonconvex and nonsmooth. We fix $n = 12$ and set $V = 10$ and $U = 1$ and run LBFGS and BFGS 500 times with starting points from $[0, 5]^n$ drawing the points from the uniform distribution. Sorting and plotting the final values of f found give the result seen in figure 11(a). BFGS apparently finds approximately 7 locally minimal values when started randomly in $[0, 5]^n$. For LBFGS (figure 11(b)) the picture is quite different. Very few runs terminate at what appears to be the global minimizing value $f^\star \approx 5.38$.

For comparison we have done the same experiment (with the same starting points) for two other algorithms, ShorRLesage and LMBM, that are both further described in 5.1. We see that only ShorRLesage finds the same 7 minima that BFGS found, but that their basins of attraction appear to have different sizes compared to those of BFGS.

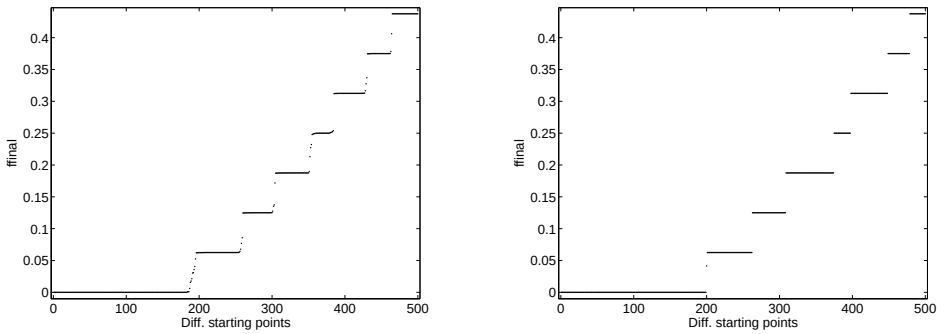
These results make it clear that when f is nonconvex and has multiple local minimizers, we cannot expect an algorithm to find the global minimizer from any starting point. See Section 4.3 below.

4.2.5 NESTEROV'S NONSMOOTH CHEBYSHEV-ROSENBROCK FUNCTION

The previous example apparently has multiple local minima. The example described next has multiple Clarke stationary points, but only one of them, the global minimizer, is a local minimizer; the others are non-locally-minimizing Clarke stationary points. The objective function is Nesterov's nonsmooth Chebyshev-Rosenbrock function given by

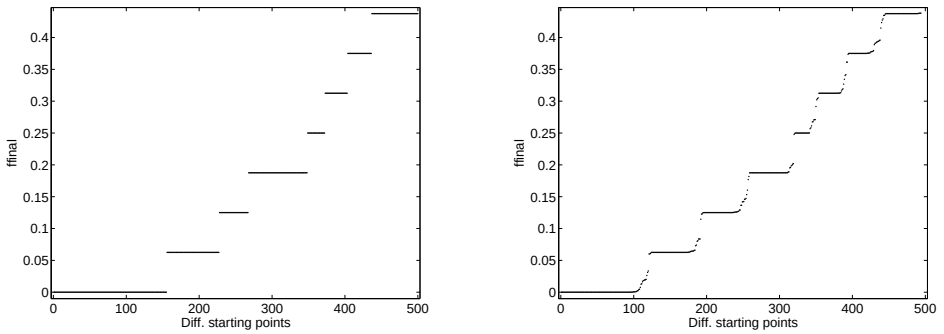
$$f(x) = \frac{1}{4}|x_1 - 1| + \sum_{i=1}^{n-1} |x_{i+1} - 2|x_i| + 1| \,. \tag{4.7}$$

As described in [LO10], minimizing f for $x \in \mathbb{R}^n$ is equivalent to minimizing the first term subject to $x \in \mathcal{S} = \{x : x_{i+1} = 2|x_i| - 1, i = 1, \ldots, n-1\}$. Because of the oscillatory and nonsmooth nature of $\mathcal{S}$ the function is very difficult to minimize even for relatively small n. There are 2^{n-1} Clarke stationary points on $\mathcal{S}$ [Gur09]. We fix $n = 4$ and as in section 4.2.4, we run four different algorithms 500 times with random starting points again drawing the points from the uniform distribution on $[0, 5]^n$. The results are seen in figures 13 and 14. We expect that there are eight Clarke stationary points including one global minimizer and this is precisely what we see. For each algorithm, there are 8 plateaus, meaning that all algorithms found all Clarke stationary points and the global minimizing value $f^\star = 0$. This shows that while it is unlikely though in principle possible for BFGS to converge to non-locally-minimizing *smooth* stationary points (that is points where $\nabla f(x) = 0$, but x is not a local minimizer), it is certainly *not* unlikely that BFGS, LBFGS and these other algorithms converge to non-locally-minimizing *nonsmooth* stationary points (that is Clarke stationary points).



(a) *BFGS* (b) *LBFGS with* $m = 30$

FIGURE 13: *Function value at termination of LBFGS and BFGS on (4.7) for 500 randomly started runs.*



(a) *ShorRLesage* (b) *LMBM with* $m = 30$

Figure 14: *Function value at termination of ShorRLesage and LMBM on (4.7) for 500 randomly started runs.*

4.3 Definition of Success

The results of sections 4.2.4 and 4.2.5 lead us to use the following criterion for successful runs: We will say that an algorithm successfully solves a problem if it in N randomly started runs successfully reduces f below

$$f_{\text{target}} := f^\star + \epsilon(|f^\star| + 1) \tag{4.8}$$

in at least $\lceil \gamma N \rceil$ of the runs, where $\gamma \in (0, 1]$. Here $f^\star$ denotes the global minimizing value and ϵ is a tolerance. We will determine $f^\star$ by running all algorithms under comparison multiple times with random starting points and using the minimal value found – unless of course it is known analytically.

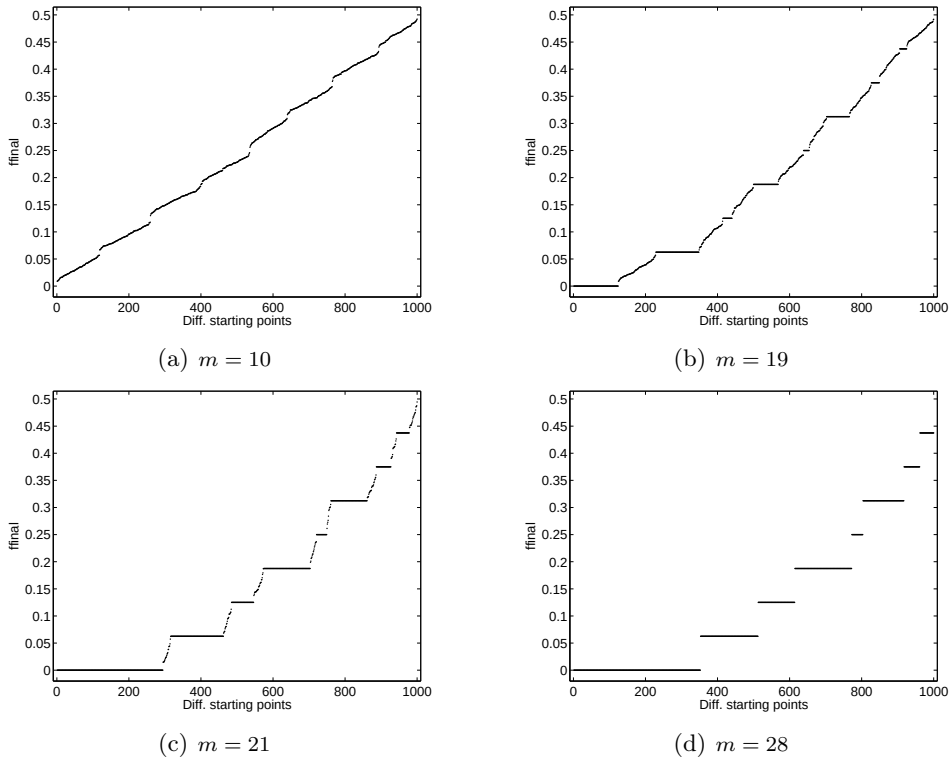
If f is convex, or is nonconvex but apparently does not have multiple minimal or Clarke stationary values, we use $\gamma = 0.7$. Otherwise the numbers N and γ will be problem dependent. We always use $\epsilon = 10^{-4}$.

4.4 LBFGS Dependence on the Number of Updates

We saw in figure 13(b) that LBFGS with $m = 30$ successfully found all eight Clarke stationary points including the global minimizer of the Chebyshev-Rosenbrock function (4.7) from section 4.2.5. Making plots similar to figure 13(b) but for different values of the number of updates m, we get the picture seen in figure 15 on the following page. From these plots, one might expect that for any given function f, there exists some lower bound value of m for which LBFGS starts succeeding.

For the objective function (4.7), one can determine such an m in the following way: Given a sorted vector $\boldsymbol{v} \in \mathbb{R}^N$, that is one that satisfies $\boldsymbol{v}_i \leq \boldsymbol{v}_{i+1}, i = 1, \ldots, N-1$, let d be such that $d(\boldsymbol{v}) \in \mathbb{R}^N$ and

$$[d(\boldsymbol{v})]_j = \min(\boldsymbol{v}_j - \boldsymbol{v}_{j-1}, \boldsymbol{v}_{j+1} - \boldsymbol{v}_j), \qquad 2 \leq j \leq N-1$$



(a) $m = 10$ (b) $m = 19$

(c) $m = 21$ (d) $m = 28$

FIGURE 15: *Results from one thousand randomly started runs of LBFGS on (4.7) with $n = 4$ for four different values of m.*

i.e. the vector of distances to the nearest neighbor in $\boldsymbol{v}$ (the first and last element are simply the distance to the *only* neighbor).

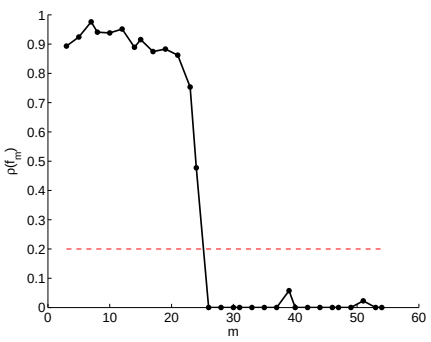
If the final values of f for N randomly started runs of LBFGS with m updates are stored (and sorted) in the vector $\boldsymbol{f}_m$, then we can expect that the number

$$\rho(\boldsymbol{f}_m) = \frac{\text{mean}(d(\boldsymbol{f}_m))}{2(\max(\boldsymbol{f}_m) - \min(\boldsymbol{f}_m))N^{-1}} \tag{4.9}$$

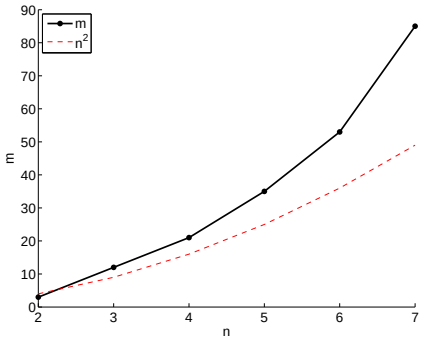
should be small if LBFGS with that m succeeded and not small if it failed. Notice that the denominator in (4.9) is the expected average distance to the nearest neighbor in a sorted vector with entries chosen from a uniform distribution on $[\min(\boldsymbol{f}_m), \max(\boldsymbol{f}_m)]$. So it simply represents a normalization to ease comparison when N varies.

In figure 16(a) on the following page, we see computed values of $\rho(\boldsymbol{f}_m)$ for different m with $n = 4$ on the Chebyshev-Rosenbrock function (4.7). We see that there is a clear cut-off point at about $m = 25$ where the average distance to the nearest neighbor drops dramatically. This agrees with the plots shown in figure 15.

Doing all of this for several values of the number of variables n and defining the smallest m for which $\rho(\boldsymbol{f}_m)$ drops below a certain tolerance

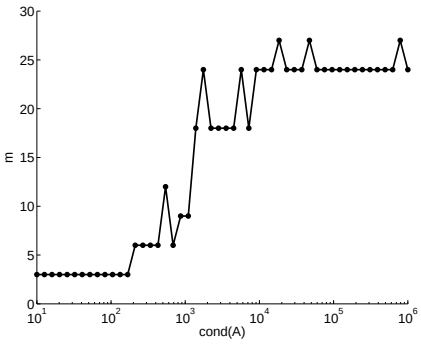


(a) *The number $\rho(\boldsymbol{f}_m)$ as function of m (see (4.9)).*

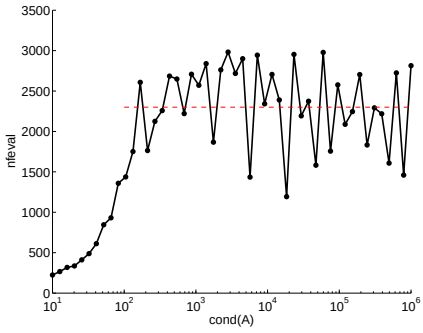


(b) *Smallest m required for LBFGS to reliably converge to a Clarke stationary point as function of n.*

FIGURE 16: *Plots illustrating the smallest necessary m for LBFGS to reliably converge to a Clarke stationary point of (4.7) on page 22.*



(a) *Smallest m required for LBFGS to reliably minimize the tilted norm function with $n = 100$, $w = 4$ as function of cond(A).*



(b) *Number of function evaluations used when minimizing the tilted norm function with $n = 100$, $w = 4$ as function of cond(A).*

FIGURE 17: *Performance of LBFGS on the tilted norm function.*

(here chosen to be 0.2) to be the smallest m required for success, we get the plot seen in 16(b). We see that for this particular problem, we actually need $m > n^2$ which makes LBFGS much more computationally expensive than full BFGS.

We now turn to a different objective function for which experiments previously suggested (figure 6 on page 17) that the performance of LBFGS is dependent on m. The objective function is the *tilted norm function* of section 4.2.1. We fix $w = 4$, $n = 100$ and vary the condition number of A. The required m to find the minimizer as a function of the condition number of A is seen in figure 17(a).

We see that although initially the required m increases as cond(A) increases, it appears that at $m \approx 25$, the algorithm succeeds regardless of

cond(A). Figure 17(b) on the previous page also seems to confirm this: From about cond(A) $\approx 10^3$ and $m \approx 25$ the number of function evaluations needed to minimize the function does not significantly increase although there are fluctuations.

5 Comparison of LBFGS and Other Methods

5.1 Other Methods in Comparison

To get a meaningful idea of the performance of LBFGS it must be compared to other currently used algorithms for the same purpose. The real strength of LBFGS is the fact that it only uses $\mathcal{O}(n)$ operations in each iteration (when $m \ll n$) and so it is specifically useful for large-scale problems – i.e. problems where n is of the order of magnitude $\geq 10^3$. The only other method that we know capable of feasibly solving nonsmooth nonconvex problems of this size, and for which we have access to an implementation, is the LMBM method presented in [HMM04]. This means that for large-scale problems, we have only one method to compare against. To get a more complete comparison we also compare LBFGS to other nonsmooth optimization algorithms for small- and medium-scale problems.

5.1.1 LMBM

The Limited Memory Bundle Method (LMBM) was introduced in [HMM04]. The method is a hybrid of the bundle method and the limited memory variable metric methods (specifically LBFGS). It exploits the ideas of the variable metric bundle method [HUL93, LV99, Kiw85], namely the utilization of null steps and simple aggregation of subgradients, but the search direction is calculated using a limited memory approach as in the LBFGS method. Therefore, no time-consuming quadratic program needs to be solved to find a search direction and the number of stored subgradients does not depend on the dimension of the problem. These characteristics make LMBM well suited for large-scale problems and it is the only one (other than LBFGS) with that property among the algorithms in comparison.

We used the default parameters except the maximal bundle size, called n_a. Since it was demonstrated in [HMM04] that LMBM on the tested problems did as well or even slightly better with $n_a = 10$ than with $n_a = 100$, we use $n_a = 10$.

Fortran code and a MATLAB mex-interface for LMBM are currently available at M. Karmitsa's website[3].

5.1.2 RedistProx

The Redistributed Proximal Bundle Method (RedistProx) introduced in [HS09] by Hare and Sagastizábal is aimed at nonconvex, nonsmooth optimization. It uses an approach based on generating cutting-plane models, not of the objective function, but of a local convexification of the objective function. The

[3] http://napsu.karmitsa.fi/

corresponding convexification parameter is calculated adaptively "on the fly". It solves a quadratic minimization problem to find the next search direction which typically requires $\mathcal{O}(n^3)$ operations. To solve this quadratic program, we use the MOSEK QP-solver [MOS03], which gave somewhat better results than using the QP-solver in the MATLAB Optimization Toolbox.

RedistProx is not currently publicly available. It was kindly provided to us by the authors to include in our tests. The authors note that this code is a preliminary implementation of the algorithm and was developed as a proof-of-concept. Future implementations may produce improvement in its results.

5.1.3 ShorR

The Shor-R algorithm dates to the 1980s [Sho85]. Its appeal is that it is easy to describe and implement. It depends on a parameter β, equivalently $1 - \gamma$ [BLO08], which must be chosen in $[0, 1]$. When $\beta = 1$ ($\gamma = 0$), the method reduces to the steepest descent method, while for $\beta = 0$ ($\gamma = 1$) it is a variant of the conjugate gradient method. We fix $\beta = \gamma = 1/2$ and follow the basic definition in [BLO08], using the weak Wolfe line search of section 3.3. However, we found that ShorR frequently failed unless the Wolfe parameter c_2 was set to a much smaller value than the 0.9 used for BFGS and LBFGS, so that the directional derivative is forced to change sign in the line search. We used $c_1 = c_2 = 0$. The algorithm uses matrix-vector multiplication and the cost is $\mathcal{O}(n^2)$ per iteration. The code is available at the website mentioned in the introduction of this thesis.

5.1.4 ShorRLesage

We include this modified variant of the Shor-R algorithm written by Lesage, based on ideas in [KK00]. The code, which is much more complicated than the basic ShorR code, is publicly available on Lesage's website[4].

5.1.5 BFGS

We include also the classical BFGS method originally introduced by Broyden, Fletcher, Goldfarb and Shanno [Bro70, Fle70, Gol70, Sha70]. The method is described in section 2.2 and experiments regarding its performance on nonsmooth examples were done by Lewis and Overton in [LO10].

We use the same line search as for the LBFGS method (described in section 3.3 and Algorithm 3) and we take $C_0 = \|\nabla f(x^{(0)})\|^{-1} I$ and scale C_1 by $\gamma = ((s^{(0)})^T y^{(0)})((y^{(0)})^T y^{(0)})^{-1}$ after the *first* step as suggested in [NW06]. This same scaling is also suggested in [NW06] for LBFGS but before *every* update. BFGS and LBFGS are only equivalent the first m (the number

[4] http://www.business.txstate.edu/users/jl47/

Name	Implementation	Method	Language
BFGS	Overton & Skajaa	Standard (full) BFGS	MATLAB
GradSamp	Overton	Gradient Sampling	MATLAB
LBFGS	Overton & Skajaa	Limited Memory BFGS	MATLAB
LMBM	Haarala	Limited Memory Bundle	Fortran77
RedistProx	Hare & Sagastizábal	Var. Proximal Bundle	MATLAB
ShorR	Overton	Shor-R	MATLAB
ShorRLesage	Lesage	Modified Shor-R	MATLAB

TABLE 2: *Overview of the different algorithms compared.*

of updates in LBFGS) steps if *none* of these scalings are implemented. For the tests reported below, we use the recommended scalings for every iteration of LBFGS and the first iteration of BFGS.

As described in section 2.2, BFGS uses matrix-vector multiplication and thus requires $\mathcal{O}(n^2)$ operations in each iteration. The code is available at the website mentioned in the introduction of this thesis.

5.1.6 GradSamp

The Gradient Sampling (GradSamp) algorithm was first introduced in [BLO02] in 2002 and analysis of the method was given in [BLO05] and [Kiw07]. At a given iterate, the gradient of the objective function on a set of randomly generated nearby points (within the *sampling radius* ϵ) is computed, and this information is used to construct a local search direction that may be viewed as an approximate ϵ-steepest descent direction. The descent direction is then obtained by solving a quadratic program. Gradient information is not saved from one iteration to the next, but discarded once a lower point is obtained from a line search. In [BLO05], the gradient sampling algorithm was found to be very effective and robust for approximating local minimizers of a wide variety of nonsmooth, nonconvex functions, including non-Lipschitz functions. However, it is much too computationally expensive for medium- or large-scale use. We use the MOSEK QP-solver [MOS03] to solve the quadratic subproblem in each iteration. The code for GradSamp is publicly available from Overton's website[5].

The proximal bundle code PBUN by Lukšan and Vlček [LV97] is publicly available at Luksan's website[6]. It is written in Fortran77, but we were unable to obtain a current mex-interface for MATLAB. For this reason, we have not included PBUN in our comparisons.

[5] http://www.cs.nyu.edu/faculty/overton/papers/gradsamp/

[6] http://www.cs.cas.cz/luksan/

n\param	M_{it}	m
10	1000	7
50	1000	20
200	1000	35
1000	5000	35
5000	5000	35
10000	5000	35

TABLE 3: *Parameters used for experiments on the test problems of section 5. m is the number of updates used in LBFGS and LMBM. M_{it} was the maximal number of iterations allowed.*

5.2 Methodology and Testing Environment

The numerical experiments in this thesis were performed using a desktop computer running GNU/Linux with a quad core 64-bit 3.0 GHz Intel processor (Q9650) and 8 GB of RAM.

RedistProx, LBFGS, BFGS, ShorR and ShorRLesage were all run as MATLAB source code while the LMBM algorithm was run via a mex-interface to the original Fortran source code provided by the author of LMBM [HMM04]. For this reason, it is not reasonable to compare the algorithms in terms of CPU time (Fortran runs faster than MATLAB). Instead we compare the algorithms in terms of how many function evaluations they use. One should always keep in mind that the limited memory methods (LBFGS and LMBM) always run considerably faster than the $\mathcal{O}(n^2)$-methods when n is large.

We use the default parameters in all the codes except as noted above and in table 3.

5.2.1 Termination criteria

We used the definition of succesful runs described in section 4.3.

For some of the test problems, we knew the optimal value $f^\star$ analytically. In the cases where we did not, we ran all the algorithms many times with very high iteration limit and chose $f^\star$ to be the best value found.

For most of the test problems the algorithms did not find multiple local minimal values. The exceptions were the generalized nonsmooth Rosenbrock function (section 4.2.4), the Schatten norm problem (section 5.3.3) and the three problems in appendix A.3. In addition, there was one problem for which the algorithms found multiple Clarke stationary points, namely Nesterov's nonsmooth Chebyshev-Rosenbrock function (section 4.2.5).

We stopped the solvers whenever the function value dropped below f_{target} (see (4.8)). In such cases we call a run "successful". Otherwise, it "failed", and the run was terminated either because the maximum number of iterations

M_{it} was reached or one of the following solver-specific events occurred:

- LBFGS, BFGS and ShorR
  1. Line search failed to find acceptable step
  2. Search direction was not a descent direction (due to rounding errors)
  3. Step size was less than a certain tolerance: $\alpha \leq \epsilon_\alpha$
- RedistProx
  1. Maximum number of restarts reached indicating no further progress possible
  2. Encountered "bad QP" indicating no further progress possible. Usually this is due to a non-positive definite Hessian occuring because of rounding errors.
- ShorRLesage
  1. Step size was less than a certain tolerance: $\alpha \leq \epsilon_\alpha$
- GradSamp
  1. Line search failed to find acceptable step
  2. Search direction was not a descent direction (due to rounding errors)
- LMBM
  1. Change in function value was less than a certain tolerance (used default value) for 10 consecutive iterations, indicating no further progress possible
  2. Maximum number of restarts reached indicating no further progress possible

5.3 Three Matrix Problems

We now consider three nonsmooth optimization problems arising from matrix applications. For the problems in this section, we use $M_{\text{it}} = 20000$.

5.3.1 An eigenvalue problem

We first consider a nonconvex relaxation of an entropy minimization problem taken from [LO10]. The objective function to be minimized is

$$f(X) = \log E_K(A \circ X) \tag{5.1}$$

Alg.\ $n = L^2$	4	16	100	144	196	256
ShorR	0/15	0/103	0/225	0/451	0/1292	0/1572
ShorRLesage	0/14	0/54	0/83	0/142	0/322	0/354
LMBM	0/5	20/86	0/197	30/940	55/2151	100/–
LBFGS	0/4	0/81	0/135	0/459	0/915	10/2496
BFGS	0/4	0/43	0/56	0/106	0/159	0/218

TABLE 4: *Performance of five different algorithms on (5.1). Each entry in table means "percentage of failed runs"/"average number of function evaluations needed for successful runs" with* $\epsilon = 10^{-4}$. *There were* 20 *randomly started runs for each* n. *The number of updates for LMBM and LBFGS was* 20. *Maximal allowed number of function evaluations was* $M_{it} = 20000$.

subject to X being an $L \times L$ positive semidefinite symmetric matrix with ones on the diagonal, where A is a fixed matrix, $\circ$ denotes the componentwise matrix-product and where $E_K(X)$ denotes the product of the K largest eigenvalues of X. Details about how to formulate this problem as an unconstrained problem with $n = L^2$ variables can be found in [LO10]. It is this version of the problem we are solving here. For A we use the $L \times L$ leading submatrices of a covariance matrix from Overton's website[7] and we take $K = L/2$. With this data, it appears that this problem does not have multiple local minimizers or Clarke stationary points [LO10]. Thus we consider a run successful if the algorithm is able to bring f below f_{target} with $\epsilon = 10^{-4}$ (see (4.8)). In table 4 we see how five different algorithms perform on the problem (5.1). ShorR, ShorRLesage and BFGS look like the more robust algorithms but BFGS is most efficient. LMBM is consistently the worst, using more function evaluations than any other solver. Furthermore, as n grows it fails on many of the runs. LBFGS does quite well when n is small but the need for function evaluations seems to grow rapidly as n gets bigger. RedistProx was not included in this comparison because it consistently failed to solve this problem.

5.3.2 A CONDITION NUMBER PROBLEM

We now consider the objective function

$$f(x) = \text{cond}(A + XX^T) \tag{5.2}$$

where A is a fixed randomly chosen symmetric positive semidefinite matrix of size $(5k) \times (5k)$ with k a positive integer. The variable is $x = \text{vec}(X)$, where X is of size $(5k) \times k$. This problem was introduced in [GV06], where an analytical expression for the optimal value is given. This is the optimal value we are using when defining f_{target} (see (4.8)) for this problem. The problem

[7] `http://cs.nyu.edu/overton/papers/gradsamp/probs/`

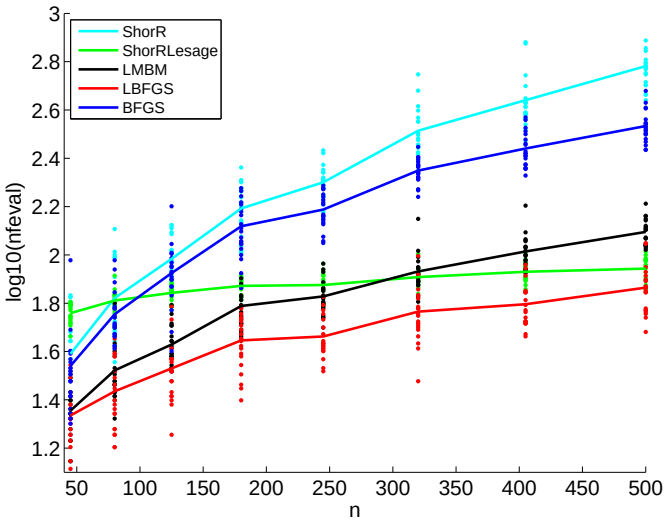


FIGURE 18: *Number of function evaluations required by five different algorithms to bring f in (5.2) below f_{target} (defined in (4.8)). There were $N = 20$ runs for each algorithm and we used $M_{it} = 20000$.*

Alg.\n	45	80	125	180	245	320	405	500
ShorR	0/39	0/66	0/96	0/156	0/200	0/327	0/437	0/605
ShorRLesage	0/57	0/65	0/70	0/74	0/75	0/81	0/85	0/88
LMBM	0/23	0/33	0/43	0/61	0/67	0/85	0/103	0/125
LBFGS	0/22	0/27	0/34	0/44	0/46	0/58	0/62	0/73
BFGS	0/35	0/57	0/84	0/131	0/154	0/223	0/276	0/342

TABLE 5: *Performance of five different algorithms on (5.2). Read the table like table 4 on the preceding page. We used $\epsilon = 10^{-4}$ and there were 20 randomly started runs for each n. Number of updates for LMBM and LBFGS was 35.*

does not seem to have local minimizers which was confirmed by numerous numerical experiments similar to those carried out in sections 4.2.4 and 4.2.5.

Table 5 shows the average number of function evaluations required by five different algorithms to reduce f below f_{target} with $\epsilon = 10^{-4}$ in $N = 20$ randomly started runs for different n. We used $n = 5k^2$, $k = 3, 4, \ldots, 10$. The algorithm RedistProx was not included in this comparison because it consistently failed to solve this problem. For this problem, LBFGS used fewer function evaluations than any other algorithm tested. Unlike the other methods, the number of function evaluations used by ShorRLesage is relatively independent of n, making it inefficient for small n but relatively good for large n. The condition number of a matrix is strongly pseudoconvex on matrix space [MY09], and this may explain why all runs were successful for

all algorithms except RedistProx.

5.3.3 THE SCHATTEN NORM PROBLEM

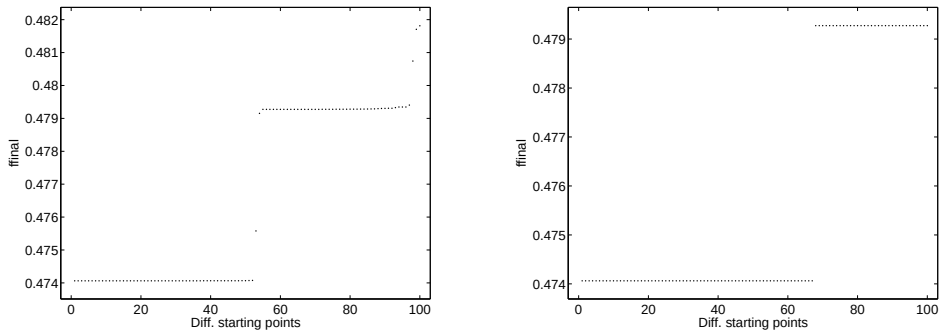
We now consider the problem

$$f(x) = \rho\|Ax - b\|_2 + \sum_{j=1}^{n} (\sigma_j(X))^p \tag{5.3}$$

where $x = \mathrm{vec}(X)$ and $\sigma_k(X)$ denotes the k'th largest singular value of X. The first term is a penalty term enforcing the linear constraints $Ax = b$. The number $p \in (0, 1]$ is a parameter. If $p = 1$, the problem is the convex nuclear norm problem that has received much attention recently, see e.g. [CR08]. For $p < 1$, f is nonconvex and not Lipschitz (when $\sigma \to 0^+$ the gradient explodes because it involves expressions of the type σ^{p-1}). For this reason, one might expect that this problem is very difficult to minimize for $p < 1$.

In figure 19, we see the function value at termination of two different algorithms when started randomly 100 times on (5.3). It is clear from the plots that there are two locally minimizing values of f that seem to attract both ShorRLesage and BFGS. Using the definition described in 4.3, we randomly start all the algorithms under comparison $N = 100$ times and call an algorithm successful if it drives f below f_{target} with $\epsilon = 10^{-4}$, see (4.8), where $f^\star$ is the function value corresponding to the lowest of the two plateaus in figure 19 in at least half of the runs ($\gamma = 0.5$).

The number of successful runs and the average number of required function evaluations are shown in table 6. GradSamp looks very robust reducing f below f_{target} in 89% of the runs. However it uses about 10 times more function evaluations than the other algorithms to do so. The performances of ShorRLesage and BFGS are very similar and both better than LMBM. The



(a) *BFGS* (b) *ShorRLesage*

FIGURE 19: *Function value at termination of 100 randomly started runs on (5.3) with $n = 8$, $p = 0.90$, $\rho = 1$ and $A \in \mathbb{R}^{4\times 8}$ and $b \in \mathbb{R}^4$ randomly chosen. Notice different scales on vertical axes.*

Alg.	$f_{\text{final}} \leq f_{\text{target}}$	Succ.	Avg. func. eval.
ShorR	20%	−	−
ShorRLesage	68%	+	365
LMBM	51%	+	410
LBFGS	0%	−	−
BFGS	62%	+	332
GradSamp	89%	+	3202
RedistProx	0%	−	−

TABLE 6: *Results of 100 randomly started runs on (5.3) with $n = 8$, $p = 0.90$ and $\rho = 1$. $A \in \mathbb{R}^{4\times 8}$ and $b \in \mathbb{R}^4$ were randomly chosen from a standard normal distribution. We used $M_{it} = 20000$.*

remaining three algorithms all fail to get f below f_{target} in more than 50% of the runs.

5.4 COMPARISONS VIA PERFORMANCE PROFILES

5.4.1 PERFORMANCE PROFILES

To better compare the performance of the different methods we use *performance profiles* which were introduced in [DM02].

Let $\mathcal{P} = \{p_1, p_2, \dots\}$ be a set of problems and let $\mathcal{S} = \{s_1, s_2, \dots\}$ be a set of solvers. If we want to compare the performance of the solvers in $\mathcal{S}$ on the problems in $\mathcal{P}$, we proceed as follows:

Let $a_{p,s}$ denote the absolute amount of computational resource (e.g. CPU time, memory or number of function evaluations) required by solver s to solve problem p. If s fails to solve p then we set $a_{p,s} = \infty$.

Let m_p denote the *least* amount of computational resource required by any solver in the comparison to solve problem p, i.e. $m_p = \min_{s\in\mathcal{S}}\{a_{p,s}\}$ and set $r_{p,s} = a_{p,s}/m_p$. The performance profile for solver s is then

$$\rho_s(\tau) = \frac{|\{p \in \mathcal{P} : \log_2(r_{p,s}) \leq \tau\}|}{|\mathcal{P}|}. \tag{5.4}$$

Consider the performance profile in figure 20 on the next page. Let us look at just one solver s. For $\tau = 0$ the numerator in (5.4) is the number of problems such that $\log_2(r_{p,s}) \leq 0$. This is only satisfied if $r_{p,s} = 1$ (because we always have $r_{p,s} \geq 1$). When $r_{p,s} = 1$, we have $a_{p,s} = m_p$ meaning that solver s was best for $\rho_s(0)$ of all the problems. In figure 20 on the following page we see that Solver 4 was best for about 56% of the problems, Solver 2 was best for about 44% of the problems and Solvers 1 and 3 were best for none of the problems.

At the other end of the horizontal axis we see that ρ_s denotes the fraction of problems which solver s succeeded in solving. If the numerator in (5.4) is

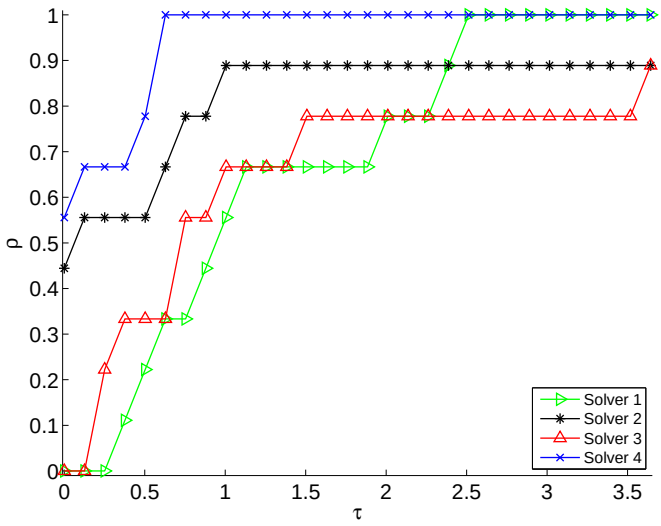


FIGURE 20: *Example of performance profile.*

not equal to $|\mathcal{P}|$ when τ is very large, then this means there were problems that the solver failed to solve. In figure 20 we see that solvers 1 and 4 solved all problems while solvers 2 and 3 solved 89% of the problems.

Generally we can read directly off the performance profile graphs the relative performance of the solvers: The higher the performance profile the better the corresponding solver. In figure 20, we therefore see that Solver 4 overall does best.

In all the performance profiles that follow the computational resource that is measured is the number of function evaluations.

5.4.2 NONSMOOTH TEST PROBLEMS F1–F9

We now present results from applying the seven (six when $n > 10$) algorithms to the test problems seen in appendix A.1 on page 45. These problems were taken from [HMM04] and they can all be defined with any number of variables. In table 7 on the following page information about the problems is seen. In table 3 on page 30 we see what parameters were used when solving the problems.

The first five problems F1-F5 are convex and the remaining four F6-F9 are not convex. We have determined through experiments that apparently none of these nonconvex problems have multiple local minima at least for $n \leq 200$. In [HMM04] the same nine problems plus one additional problem were used in comparisons. However, because that additional problem has several local minima, we have moved that problem to a different profile – see section 5.4.4.

We run all the algorithms $N = 10$ times with randomly chosen starting

Problem	Convex	$f^\star$
F1	+	0
F2	+	0
F3	+	$-(n-1)/2$
F4	+	$2(n-1)$
F5	+	$2(n-1)$
F6	−	0
F7	−	0
F8	−	–
F9	−	0

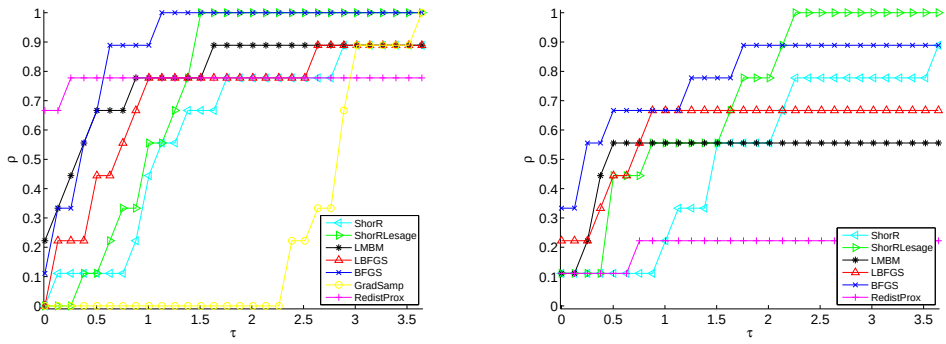
TABLE 7: *Information about the problems F1-F9. The column Cvx. indicates whether or not the problem is convex. $f^\star$ shows the optimal value.*

points (see section 4.1.1) and use the definition of success described in section 4.3, here with $\gamma = 0.7$ and $\epsilon = 10^{-4}$.

Figure 21 shows the performance profiles for the seven algorithms on the problems F1-F9 with $n = 10$ and $n = 50$.

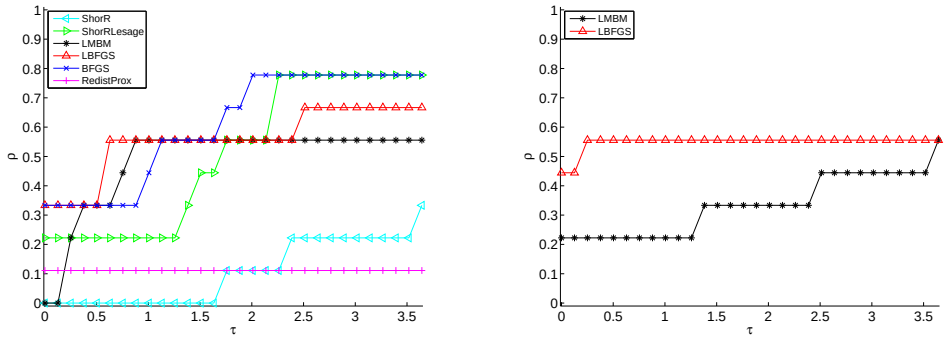
For $n = 10$ only BFGS, ShorRLesage and GradSamp successfully solve all the problems to within the required accuracy. RedistProx is both for $n = 10$ and $n = 50$ the least robust algorithm, however for $n = 10$, it uses fewest function evaluations in 6 of the 7 problems it solves. BFGS, LBFGS and LMBM use significantly fewer function evaluations than ShorR, ShorRLesage and GradSamp.

The picture for $n = 50$ is somewhat similar, except that the two limited memory algorithms now fail on 3 and 4 of the nine problems. Only ShorRLesage solves all nine problems but BFGS still solves all but one prob-



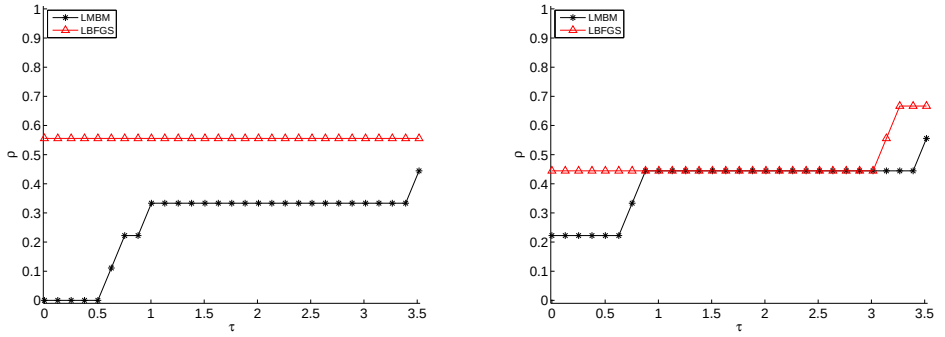
(a) $n = 10$ (b) $n = 50$

FIGURE 21: *Performance profiles for the seven algorithms on the test problems F1-F9.*



(a) $n = 200$ (b) $n = 1000$

FIGURE 22: *Performance profiles for six algorithms on the test problems F1-F9 for* $n = 200$ *and the two limited memory methods for* $n = 1000$.



(a) $n = 5000$ (b) $n = 10000$

FIGURE 23: *Performance profiles for the seven algorithms on the test problems F1-F9.*

lem and still uses significantly fewer function evaluations.

When we increase n to 200, no algorithm solves more than 7 of the nine problems. Only BFGS and ShorRLesage solve that many and again, between the two, BFGS uses much fewer function evaluations. LBFGS is best on 3 of the 6 problems it solves and uses fewer function evaluation than the other limited memory method LMBM, which solves a total of 5 problems. It is also interesting to note that ShorRLesage is much more efficient and reliable than ShorR on these problems.

Now looking at the results for $n = 1000$, 5000 and 10000 shown in figure 22(b) and figure 23 we see that LBFGS is consistently and by quite a large margin the better performer of the two limited memory algorithms on these problems. While LMBM was better than LBFGS for $n = 10$ it appears to lose more and more ground to LBFGS when n grows. As described in section 4.1.3 on page 16, we can estimate the dimension of the V-space at the

P \ n	10	19	28	37	46	55	64	gen.	asymp.
F1	1	0	5	4	5	3	9	?	?
F2	5	5	5	5	5	5	5	5	$\mathcal{O}(1)$
F3	9	18	27	36	45	54	63	$n-1$	$\mathcal{O}(n)$
F4	10	19	28	37	46	55	64	n	$\mathcal{O}(n)$
F5	2	2	2	2	2	2	2	2	$\mathcal{O}(1)$
F6	1	1	1	1	1	1	1	1	$\mathcal{O}(1)$
F7	10	19	28	37	46	55	64	n	$\mathcal{O}(n)$
F8	7	16	25	33	43	52	61	$n-3$?	$\mathcal{O}(n)$
F9	1	1	1	1	1	1	1	1	$\mathcal{O}(1)$

TABLE 8: *Computed V-space information about the problems F1-F9.*

minimizer by running BFGS and counting the number of very small eigenvalues of the approximated inverse Hessian at the final iterate – assuming that BFGS terminated at, or very close to, the minimizer. Table 8 shows these computed V-space dimensions.

We see that the problem set F1-F9 includes both problems where the V-space is low dimensional and problems where the V-space is almost the full space. This means that these experiments do not favor an algorithm better able to solve one or the other type of problem.

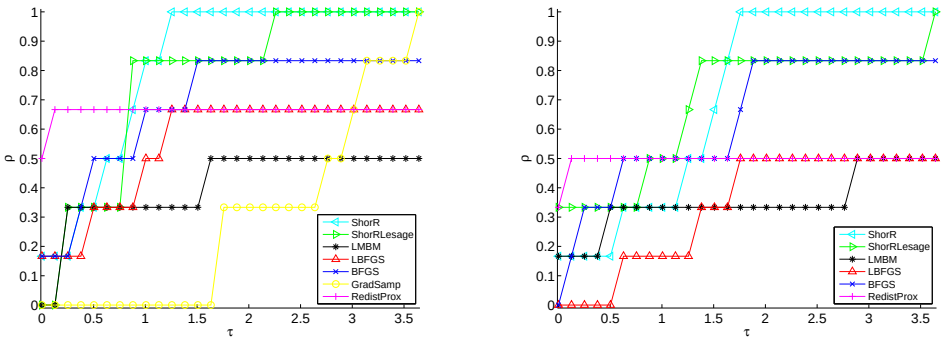
5.4.3 NONSMOOTH TEST PROBLEMS T1–T6

We now consider the results from applying the seven algorithms to the problems from `TEST29` seen in appendix A.2. These problems were taken from [LTS+02] and may be defined with any number of variables. For all problems, we have confirmed via numerical experiments that they apparently do not have multiple local minima. Table 3 on page 30 shows the parameters used for these test problems.

As for the problems F1-F9, we do $N = 10$ randomly initialized runs and use $\gamma = 0.7$.

Figure 24 on the next page shows the performances for $n = 10$ and $n = 50$. We see that RedistProx performs the fastest on most of the problems, but fails to solve some problems. For $n = 10$, three algorithms successfully solve all 6 problems, namely ShorRLesage, ShorR and GradSamp, while BFGS solves 5. For both $n = 10$ and $n = 50$, the two limited memory algorithms do worst and specially for $n = 50$ the result is very poor. Both methods bring f below the target on only *three* of the problems. For $n = 10$, LBFGS is slightly better than LMBM and for $n = 50$ LMBM is the slightly better algorithm.

In figure 25 on the following page we see the results for $n = 200$ and $n = 1000$. For $n = 200$, only BFGS solves 4 of 6 problems – the rest solve



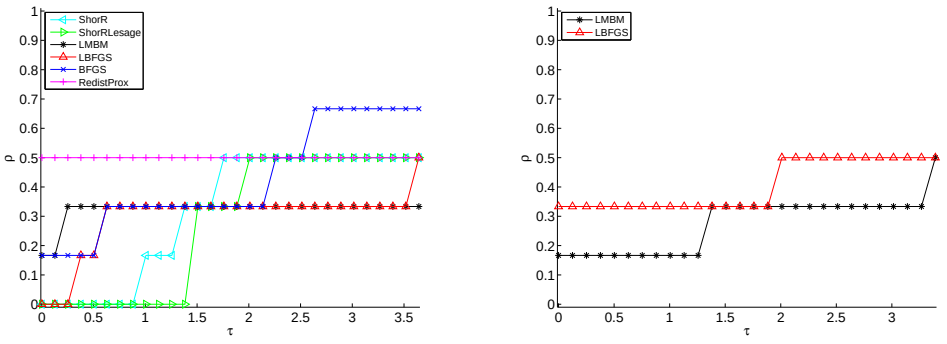
(a) $n = 10$ (b) $n = 50$

FIGURE 24: *Performance profiles for the seven algorithms on the test problems T1-T6.*

either 3 or 2 problems. On this problem set, RedistProx does much better than that of section 5.4.2. Although it is still not very robust, it uses very few function evaluations to solve the problems that it successfully solves.

When $n = 1000$, only the two limited memory algorithms are feasible to use and we see from figure 25(b) that both algorithms succeed on 3 of the 6 problems. LBFGS is best on two of them and LMBM on the last.

It is clear from these experiments that the problems T1-T6 are generally harder than the problems F1-F9 for which we presented results in section 5.4.2. In table 9 on the next page, we see the V-space information that BFGS was able to provide for the problems T1-T6. We see that as for the problems F1-F9, we have also in this problem set problems of both high and low V-space dimension – it was, however, much more difficult to determine how many eigenvalues of the inverse Hessian were bounded away from zero. To

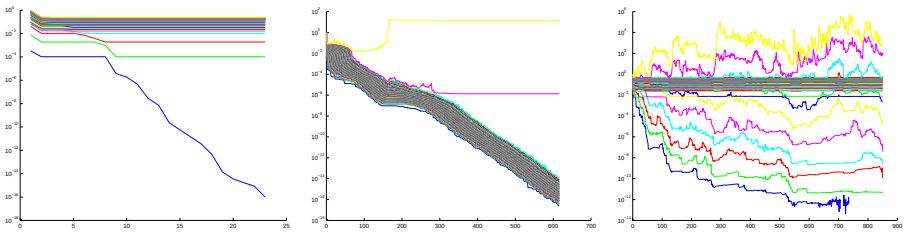


(a) $n = 200$ (b) $n = 1000$

FIGURE 25: *Performance profiles for six algorithms on the test problems T1-T6 for* $n = 200$ *and the two limited memory methods for* $n = 1000$.

P \ n	10	20	30	40	50	60	70	gen.	asymp.
T1	10	20	30	40	50	60	70	n	$\mathcal{O}(n)$
T2	4	5	6	7	7?	7?	8?	?	$\mathcal{O}(1)$?
T3	1	1	1	1	1	1	1	1	$\mathcal{O}(1)$
T4	8	18	28	38	49(!)	58	68	$n-2$?	$\mathcal{O}(n)$
T5	10	20	30	40	50	60	70	n	$\mathcal{O}(n)$
T6	10	20	30	40	50	60	70	n	$\mathcal{O}(n)$

TABLE 9: *Computed V-space information about the problems T1-T6. A question-mark indicates that BFGS was unable to solve the problem to an accuracy sufficient to display any useful V-space information.*



(a) *T3* (b) *T4* (c) *T2*

FIGURE 26: *Eigenvalues of C_j as function of iteration counter j for selected problems from the problem set T1-T6 with $n = 60$.*

see the difficulty, consider the plots in figure 26. In the left and the middle pane of figure 26 it is not hard to see that the eigenvalues separate in two groups: those that go to zero and those that do not. In fact we can count that for (a) T3, one eigenvalue goes to zero (b) T4, all but two eigenvalues go to zero. As seen in the right pane, it is essentially not possible for (c) T2 to determine how many eigenvalues go to zero before BFGS terminates.

For the problems F1-F9 from section 5.4.2, we were able to compare the two limited memory algorithms when $n = 5000$ and $n = 10000$. For the problem set T1-T6, however, neither method succeeds in enough cases to make comparison meaningful.

5.4.4 NONCONVEX PROBLEMS WITH SEVERAL LOCAL MINIMA

In this section, we present results from comparing the seven algorithms on problems that have several local minima or non-locally-minimizing Clarke stationary points. These problems are the three problems P1-P3 shown in appendix A.3 along with the nonsmooth Rosenbrock function of Section 4.2.4, the Nesterov problem of 4.2.5 and the Schatten norm problem of section 5.3.3 but here all with $n = 10$ variables.

For these problems we do $N = 10$ randomly initialized runs and set

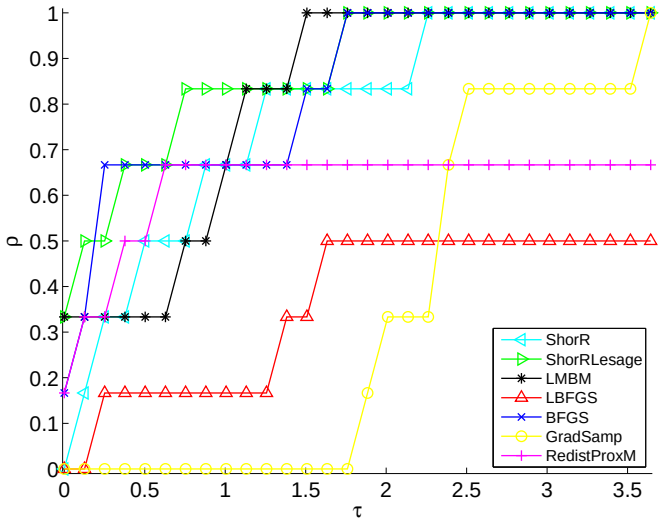


FIGURE 27: *Performance profile for seven algorithms on the six test problems with several local minima or non-locally-minimizing Clarke stationary points, with $n = 10$ variables and $\gamma = 0.3$. There were $N = 10$ randomly initialized runs and the starting points from a uniform distribution on $[0, 2]^n$.*

$\gamma = 0.3$. This relatively small γ was chosen because the problems are known to have several local minima or non-locally-minimizing Clarke stationary points which we know attract the iterates. Hence it is reasonable to expect that the algorithms more often fail to reduce f below f_{target} than if only one such point was present. We drew the starting points from a uniform distribution on $[0, 2]^n$.

In figure 27 we see how the algorithms compare on these problems. Five of the seven algorithms succeed in solving all of the six problems. In terms of function evaluations, BFGS and ShorRLesage do somewhat better than ShorR and LMBM and these do much better than GradSamp. For these problems, LBFGS is the poorest performer with only three of the six problems successfully solved. RedistProx is a lot better than LBFGS in these problems, but as for the other problem sets, it is not very robust, solving only four of six problems.

5.5 SUMMARY OF OBSERVATIONS

In the previous sections we have found that for

- small- and medium-scale problems ($1 \leq n < 1000$),
  - BFGS was, together with ShorRLesage, generally the most robust algorithm solving totally 45 of 51 problems. In terms of function evaluations, it was the best algorithm on the problems F1-F9 for $n = 10, 50$ and 200. It was also the most efficient on the eigenvalue problem (5.1) and the Schatten norm problem (5.3). It should also be noted for its robustness and its consistent convergence rates, see figures in section 4.2.
  - ShorRLesage was equally robust solving totally 47 of 51 problems. In terms of function evaluations, it was rarely the most efficient except on the small scale versions of the problems T1-T6 and the condition number problem (5.2). It used more function evaluations than BFGS on the problems F1-F9 but on the problems T1-T6 it was slightly more robust than BFGS.
  - ShorR was satisfactory only for the very small scale versions of the problems T1-T6 and only in terms of robustness. Its performance dropped dramatically for larger n. With a few exceptions, ShorR appeared to be worse than ShorRLesage, so there is no reason to prefer it.
  - RedistProx was generally the least predictable of the tested algorithms. It was unable to solve any of the matrix problems and did not compare very well on the performance profiles for the problem set F1-F9 – except when $n = 10$, where it used few function evaluations but failed to solve two of the problems. It was overall not very robust, failing on seven of nine the problems for $n = 50$ and eight of nine for $n = 200$. On the problem set T1-T6, it failed on two of the six problems for $n = 10$ and on three of six for $n = 50$. On the other hand, in the same problem set, it was best on almost all of the problems that it successfully solved. On the nonconvex problems of section 5.4.4, it was about as fast as the best methods on this set, but failed on two of six problems.
  - LBFGS was, on the small- and medium-scale problems, comparable to LMBM. It was more efficient on the condition number problem (5.2), but clearly worse on the Schatten norm problem (5.3). On the eigenvalue problem (5.1), it was more robust than LMBM. On the problems F1-F9, LBFGS was overall marginally better than LMBM for $n \leq 200$. On the problems T1-T6, LBFGS was best for smaller n, but the two were about equal for larger n. Generally the behavior of LBFGS is much more erratic than

that of BFGS. This was particularly clear on the problems of section 4.2. LBFGS did very poorly on the nonconvex problems of section 5.4.4. Here it solved only three of six problems and it did so using significantly more function evaluations than every other algorithm except GradSamp.

- LMBM was similar in its behavior to LBFGS on most small-scale problems. The only exception was the nonconvex problems of section 5.4.4. Here LMBM did well solving all problems with about the same use of function evaluations as the other successful algorithms.
- GradSamp was included only in the performance profiles for $n = 10$. The reason for this was its intense use of function evaluations. It was simply infeasible to solve bigger problems. However, it should be noted that it was very robust, solving all problems on which it was applied. Its need for function evaluations was far greater than that of any other method.

• large-scale problems ($n \geq 1000$),

- LBFGS was notably superior to LMBM. On the test problems F1-F9, LBFGS got relatively better compared to LMBM when n was bigger. For $n = 1000$, both methods solved five of nine problems and LBFGS was best on all but one of them. For $n = 5000$, LBFGS again solved five while LMBM solved four of nine problems and LBFGS was best on all of them. Similarly, LBFGS was superior for $n = 10000$. On the problem set T1-T6, which was overall a harder set of problems than F1-F9, LBFGS and LMBM both successfully solved three problems for $n = 1000$ and LBFGS was more efficient on two of them. However, neither method succeeded when $n = 5000$ or $n = 10000$. Overall, LBFGS was clearly superior to LMBM for large n, which is a surprising conclusion, considering that LMBM was specifically designed for large-scale nonsmooth optimization while LBFGS was originally meant for smooth problems.

A Test Problems

A.1 Nonsmooth Test Problems F1–F9

The following nine problems were taken from [HMM04]. They are all nonsmooth at the minimizer. The first five are convex and the following four are nonconvex. They can all be formulated with any number of variables and we will refer to them as F1 through F9. They were implemented in MATLAB by the author of this thesis based on the Fortran implementations from Karmitsa's website (see page 27). Information about all the problems is gathered in table 7 on page 37 and table 8 on page 39.

F1: Generalization of MAXQ

$$f(x) = \max_{1 \le i \le n} x_i^2 \tag{A.1}$$

F2: Generalization of MAXHILB

$$f(x) = \max_{1 \le i \le n} \left| \sum_{j=1}^{n} \frac{x_j}{i+j-1} \right| \tag{A.2}$$

F3: Chained LQ

$$f(x) = \sum_{i=1}^{n-1} \max\left\{-x_i - x_{i+1}, -x_i - x_{i+1} + \left(x_i^2 + x_{i+1}^2 - 1\right)\right\} \tag{A.3}$$

F4: Chained CB3 I

$$f(x) = \sum_{i=1}^{n-1} \max\left\{x_i^4 + x_{i+1}^2, (2-x_i)^2 + (2-x_{i+1})^2, 2e^{-x_i+x_{i+1}}\right\} \tag{A.4}$$

F5: Chained CB3 II

$$f(x) = \max\left\{\sum_{i=1}^{n-1}\left(x_i^4 + x_{i+1}^2\right), \sum_{i=1}^{n-1}\left((2-x_i)^2 + (2-x_{i+1})^2\right), \sum_{i=1}^{n-1}\left(2e^{-x_i+x_{i+1}}\right)\right\} \tag{A.5}$$

F6: Number of Active Faces

$$f(x) = \max_{1 \le i \le n}\left\{g\left(-\sum_{i=1}^{n} x_i\right), g(x_i)\right\}, \quad \text{where } g(y) = \ln(|y|+1) \tag{A.6}$$

F7: Nonsmooth Generalization of Brown Function 2

$$f(x) = \sum_{i=1}^{n-1}\left(|x_i|^{x_{i+1}^2+1} + |x_{i+1}|^{x_i^2+1}\right) \tag{A.7}$$

F8: Chained Mifflin 2

$$f(x) = \sum_{i=1}^{n-1} \left(-x_i + 2(x_i^2 + x_{i+1}^2 - 1) + \frac{7}{4} \left| x_i^2 + x_{i+1}^2 - 1) \right| \right) \tag{A.8}$$

F9: Chained Crescent I

$$f(x) = \max \left\{ \sum_{i=1}^{n-1} \left(x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1 \right), \sum_{i=1}^{n-1} \left(-x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1 \right) \right\} \tag{A.9}$$

A.2 NONSMOOTH TEST PROBLEMS T1–T6

These test problems are from the test library `TEST29` from [LTS+02], which contains many more problems. They were also used in [HMM04].

T1: Problem 2 from `TEST29`

$$f(x) = \max_{1 \le i \le n} |x_i| \tag{A.10}$$

T2: Problem 5 from `TEST29`

$$f(x) = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} \frac{x_j}{i + j - 1} \right| \tag{A.11}$$

T3: Problem 6 from `TEST29`

$$f(x) = \max_{1 \le i \le n} \left| (3 - 2x_i)x_i + 1 - x_{i-1} + x_{i-1} \right|, \quad \text{with } x_0 = x_{n+1} = 0 \tag{A.12}$$

T4: Problem 11 from `TEST29`

$$f(x) = \sum_{k=1}^{2(n-1)} |f_k(x)| \tag{A.13}$$

where (A.14)

$$f_k(x) = x_i + x_{i+1}((1 + x_{i+1})x_{i+1} - 14) - 29 \quad \text{if } \mathrm{mod}(k, 2) = 0 \tag{A.15}$$

$$f_k(x) = x_i + x_{i+1}((5 - x_{i+1})x_{i+1} - 2) - 13 \quad \text{if } \mathrm{mod}(k, 2) = 1 \tag{A.16}$$

$$i = \lfloor (k + 1)/2 \rfloor \tag{A.17}$$

T5: Problem 22 from `TEST29`

$$f(x) = \max_{1 \le i \le n} \left| 2x_i + \frac{1}{2(n+1)^2} \left(x_i + \frac{i}{n+1} + 1 \right)^3 - x_{i-1} - x_{i+1} \right| \tag{A.18}$$

with $x_0 = x_{n+1} = 0$.

T6: Problem 24 from `TEST29`

$$f(x) = \max_{1 \le i \le n} \left| 2x_i + \frac{10}{(n+1)^2} \sinh(10x_i) - x_{i-1} - x_{i+1} \right| \tag{A.19}$$

with $x_0 = x_{n+1} = 0$.

A.3 Other Nonsmooth Test Problems

The problems presented here all have either multiple local minima or non-locally-minimizing Clarke stationary points.

P1: Problem 13 from `TEST29`

$$f(x) = \sum_{k=1}^{2(n-2)} \left| y_l + \sum_{h=1}^{3} \frac{h^2}{l} \prod_{j=1}^{4} \frac{x_{i+j}}{|x_{i+j}|} \left| x_{i+j}^{j/hl} \right| \right| \tag{A.20}$$

where $i = 2\lfloor (k+3)/4 \rfloor - 2$, $l = \mathrm{mod}(k-1, 4) + 1$ and $y_1 = -14.4$, $y_2 = -6.8$, $y_3 = -4.2$ and $y_4 = -3.2$. Further n must be even.

P2: Problem 17 from `TEST29`

$$f(x) = \max_{1 \le i \le n} \left| 5 - (j+1)(1 - \cos x_i) - \sin x_i - \sum_{k=5j+1}^{5j+5} \cos x_k \right| \tag{A.21}$$

where $j = \lfloor (i-1)/5 \rfloor$. Further n must be a multiple of 5.

P3: Chained Crescent II

This problem was used as a tenth problem (F10) in the set F1-F9 (see appendix A.1) in [HMM04], but since we determined that there were several local minimal values, we moved it to this problem set.

$$f(x) = \sum_{i=1}^{n-1} \max \left\{ x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1, -x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1 \right\} \tag{A.22}$$

References

[BL00] J. Borwein and A. S. Lewis. *Convex Analysis and Nonlinear Optimization: Theory and Examples*. Springer, 2000.

[BLO02] J. V. Burke, A. S. Lewis, and M. L. Overton. Two numerical methods for optimizing matrix stability. *Linear Algebra Appl.*, 351-352:117–145, 2002.

[BLO05] J. V. Burke, A. S. Lewis, and M. L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM J. Optimization*, 15:751–779, 2005.

[BLO08] J. V. Burke, A. S. Lewis, and M. L. Overton. The speed of Shor's R-algorithm. *IMA J. Numer. Anal.*, 28(4):711–720, 2008.

[Bro70] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6(1):76–90, 1970.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Cla83] F. H. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley, New York. Reprinted by SIAM, Philadelphia, 1983.

[CR08] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Found. of Comput. Math.*, 9:717–772, 2008.

[DM02] E.D. Dolan and J.J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[Fle70] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.

[Gol70] D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[Gur09] M. Gurbuzbalaban, 2009. Private communication.

[GV06] C. Greif and J. Varah. Minimizing the condition number for small rank modifications. *SIAM J. Matrix Anal. Appl.*, 29(1):82–97, 2006.

[HMM04] M. Haarala, K. Miettinen, and M. M. Makela. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software*, 19(6):673–692, 2004.

[HS09] W. Hare and C Sagastizabal. A redistributed proximal bundle method for nonconvex optimization. *submitted to SIAM J. Opt*, 2009.

[HUL93] J.B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms*. Springer Verlag, 1993.

[Kiw85] K.C. Kiwiel. *Methods of Descent for Nondifferentiable Optimization. Lecture Notes in Mathematics 1133*. Springer Verlag, 1985.

[Kiw07] K.C. Kiwiel. Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization*, 18(2):379–388, 2007.

[KK00] F. Kappel and A. Kuntsevich. An implementation of Shor's R-algorithm. *Computational Optimization and Applications*, 15:193–205, 2000.

[Lem81] C. Lemaréchal. A view of line searches. *Optimization and Optimal Control, Auslender, Oettli, Stoer Eds, Lecture Notes in Control and Information Sciences 30*, pages 59–78, 1981.

[Lew02] A. S. Lewis. Active sets, nonsmoothness, and sensitivity. *SIAM J. on Optimization*, 13(3):702–725, 2002.

[LO10] A.S. Lewis and M.L. Overton. Nonsmooth optimization via BFGS. *Submitted to SIAM J. Optimization*, 2010.

[LOS00] C. Lemaréchal, F. Oustry, and C. Sagastizábal. The U-Lagrangian of a convex function. *Trans. Amer. Math. Soc.*, 352:711.729, 2000.

[LTS+02] L. Lukšan, M. Tuma, M. Siska, J. Vlček, and N. Ramesova. UFO 2002. Interactive system for universal functional optimization. Research report, Academy of Sciences of the Czech Republic, 2002.

[LV97] L. Lukšan and J. Vlček. PBUN, PNEW - a bundle-type algorithms for nonsmooth optimization. Research report, Academy of Sciences of the Czech Republic, 1997.

[LV99] L. Lukšan and J. Vlček. Globally convergent variable metric methods for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory and Applications*, 102(3):593–613, 1999.

[MOS03] MOSEK. MOSEK optimization software, 2003. `http://www.mosek.com/`.

[MY09] Pierre Maréchal and Jane J. Ye. Optimizing condition numbers. *SIAM J. Optim.*, 20(2):935–947, 2009.

[Noc80] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[RW98] R.T. Rockafellar and R.J.B. Wets. *Variational Analysis*. Springer, New York, 1998.

[Sha70] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[Sho85] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer Verlag, 1985.