New York University

Master's Thesis

An L-BFGS-B-NS Optimizer for Non-Smooth Functions

Author:
Wilmer Henao

Supervisor:
Michael L. Overton

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in Scientific Computing

in the

Courant Institute of Mathematical Sciences
Department of Mathematics

May 2014

Declaration of Authorship

I, Wilmer Henao, declare that this thesis titled, 'An L-BFGS-B-NS Optimizer for Non-Smooth Functions' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

NEW YORK UNIVERSITY

Abstract

Department of Mathematics

Master of Science in Scientific Computing

An L-BFGS-B-NS Optimizer for Non-Smooth Functions

by Wilmer Henao

This thesis investigates a large scale `L-BFGS-B` optimizer for smooth functions and how it can be modified to optimize non-smooth functions. The new code is called `L-BFGS-B-NS`. The changes required include a relaxation of the Wolfe condition, some other changes to the line search algorithm and changes to the termination condition. Some experiments illustrate the results applied to a non-smooth function.

Acknowledgements

I would like to thank my advisor Michael Overton for all the hours of hard work and for all the great recommendations and changes that led to this thesis.

I also would like to thank Allan Kaku and Anders Skajaa for earlier contributions to other versions of the code, and Jorge Nocedal, Ciyou Zhu, Richard Byrd and Peihuang Lu, for letting us change their original code.

Finally, I would like to thank High Performance Computing at NYU for the computing resources and their helpful assistance.

Contents

List of Figures

List of Tables

Dedicated to my mother and Dr. Ian Malcolm

Chapter 1

Introduction

The problem addressed is to find a local minimizer of the non-smooth minimization problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & l_i \leq x_i \leq u_i, \\ & i = 1, \ldots, n. \end{aligned} \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, is continuous but not differentiable everywhere and n is large.

The `L-BFGS-B` algorithm [BLNZ95] is a standard method for solving large instances of (1.1) when f is a smooth function, typically twice differentiable. The name `BFGS` stands for Broyden, Fletcher, Goldfarb and Shanno, the originators of the `BFGS` quasi-Newton algorithm for unconstrained optimization discovered and published independently by them in 1970 [Bro70, Fle70, Gol70, Sha70]. This method requires storing and updating a matrix which approximates the inverse of the Hessian $\nabla^2 f(x)$ and hence requires $\mathcal{O}(n^2)$ operations per iteration. The `L-BFGS` variant [Noc80], where the `L` stands for "Limited-Memory" and also for "Large" problems, is based on `BFGS` but requires only $\mathcal{O}(n)$ operations per iteration, and less memory. Instead of storing the $n \times n$ Hessian approximations, `L-BFGS` stores only m vectors of dimension n, where m is a number much smaller than n. Finally, the last letter B in `L-BFGS-B` stands for bounds, meaning the lower and upper bounds l_i and u_i in equation (1.1). The `L-BFGS-B` algorithm is implemented in a FORTRAN software package [ZBNM11].

In this thesis, we first give a brief description of the `L-BFGS-B` algorithm at a high level and then we introduce a modified algorithm which is more suitable for functions f which may not be differentiable at their local or global optimal points. We call the new

algorithm L-BFGS-B-NS where NS stands for non-smooth. These changes were implemented in a modified version of the FORTRAN code [Hen14] which can be downloaded from a web repository. We report on some numerical experiments that strongly suggest that the new code should be useful for the non-smooth bound-constrained optimization problem (1.1).

We are grateful to Jorge Nocedal and his coauthors for allowing us to modify the L-BFGS-B code and post the modified version.

Chapter 2

L-BFGS-B

This section is a description of the original `L-BFGS-B` code [ZBNM11, ZBLN97] at a very high level. The original software is intended to find local minimizers of smooth functions. This thesis discusses how to modify the algorithm for non-smooth functions.

2.1 BFGS

`BFGS` is a standard tool for optimization of smooth functions [NW99]. It is a line search method. The search direction is of type $d = -B_k \nabla f(x_k)$ where B_k is the k^{th} approximation to the inverse Hessian of f.[1] This k^{th} step approximation is calculated via the `BFGS` formula

$$B_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) B_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k} \tag{2.1}$$

where $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$. `BFGS` exhibits super-linear convergence on generic problems but it requires $\mathcal{O}(n^2)$ operations per iteration [NW99].

In the case of non-smooth functions, `BFGS` typically succeeds in finding a local minimizer [LO13]. However, this requires some attention to the line search conditions. These conditions are known as the Armijo and weak Wolfe line search conditions and they are a set of inequalities used for the computation of an appropriate step length that reduces the objective function "sufficiently". These inequalities will be explained later in section 3.1.

[1] When it is exactly the inverse Hessian the method is known as Newton's method. Newton's method has quadratic convergence but requires the explicit calculation of the Hessian at every step.

2.2 L-BFGS

`L-BFGS` stands for Limited-memory `BFGS`. This algorithm approximates `BFGS` using only a limited amount of computer memory to update an approximation to the inverse of the Hessian of f. Instead of storing a dense $n \times n$ matrix, `L-BFGS` keeps a record of the last m iterations where m is a small number that is chosen in advance[2]. For this reason the first m iterations of `BFGS` and `L-BFGS` produce exactly the same search directions if the initial approximation B_0 is set to the identity matrix.

Because of this construction, the `L-BFGS` algorithm is less computationally intensive and requires only $\mathcal{O}(mn)$ operations per iteration. So it is much better suited for problems where the number of dimensions n is large.

2.3 L-BFGS-B

Finally `L-BFGS-B` is an extension of `L-BFGS`. The B stands for the inclusion of Boundaries. `L-BFGS-B` requires two extra steps on top of `L-BFGS`. First, there is a step called *gradient projection* that reduces the dimensionality of the problem. Depending on the problem, the gradient projection could potentially save a lot of iterations by eliminating those variables that are on their bounds at the optimum reducing the initial dimensionality of the problem and the number of iterations and running time. After this *gradient projection* comes the second step of *subspace minimization*. During the *subspace minimization* phase, an approximate quadratic model of (1.1) is solved iteratively in a similar way that the original `L-BFGS` algorithm is solved. The only difference is that the step length is restricted as much as necessary in order to remain within the lu-box defined by equation (1.1).

2.3.1 Gradient Projection

The `L-BFGS-B` algorithm was designed for the case when n is large and f is smooth. Its first step is the gradient projection similar to the one outlined in [CGT88, MT89] which is used to determine an active set corresponding to those variables that are on either their lower or upper bounds. The active set defined at point x^* is:

$$\mathcal{A}(x^*) = \{i \in \{1 \dots n\} | x_i^* = l_i \vee x_i^* = u_i\} \tag{2.2}$$

[2] In this thesis $m < 20$, and in practice numbers between 5 and 10 are regularly used. There is no way of knowing *a priori* what choice of m will provide the best results as will be illustrated later.

Working with this active set is more efficient in large scale problems. A pure line search algorithm would have to choose a step length short enough to remain within the box defined by l_i and u_i. So if at the optimum, a large number $\mathcal{B}$ of variables are either on the lower or the upper bound, as many as $\mathcal{B}$ of iterations might be needed. Gradient projection tries to reduce this number of iterations. In the best case, only 1 iteration is needed instead of $\mathcal{B}$.

Gradient projection works on the linear part of the approximation model:

$$m_k(x) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{(x - x_k)^T H_k (x - x_k)}{2} \tag{2.3}$$

where H_k is a `L-BFGS-B` approximation to the Hessian $\nabla^2 f$ stored in the implicit way defined by `L-BFGS`.

In this first stage of the algorithm a piece-wise linear path starts at the current point x_k in the direction $-\nabla f(x_k)$. Whenever this direction encounters one of the constraints, the path turns corners in order to remain feasible. The path is nothing but the feasible piece-wise projection of the negative gradient direction on the constraint box determined by the values l and u. At the end of this stage, the value of x that minimizes $m_k(x)$ restricted to this piece-wise gradient projection path is known as the "Cauchy point" x^c. See Figure 2.1.

2.3.2 Subspace Minimization

The problem with gradient projection is that its search direction does not take advantage of information provided implicitly by the Hessian H_k, and therefore the speed of convergence is at best linear. It is for this reason that a second stage is necessary. Stage 2 (subspace minimization) uses an `L-BFGS` implicit approximation of the inverse Hessian matrix restricted to the free variables that are not in the active set $\mathcal{A}(x^c)$.

The starting position for stage 2 will be the previously found Cauchy point and the goal is to find a new $\bar{x} = x^c + \alpha^* \hat{d}$. The idea at a higher level is to minimize (2.3) over the free variables subject to their lower and upper bounds. First, the `L-BFGS` algorithm provides a new search direction $\hat{d}^u$ of the *unconstrained* problem that takes implicit advantage of approximations of the Hessian matrix restricted to the free variables. After an unconstrained search direction has been found, the constraints are taken into account and the search direction is restricted to the l, u bounding box via a step length factor α^*. The step length is chosen so that the new point $\bar{x}$ satisfies the Armijo and Wolfe[3]

[3]The Armijo and Wolfe conditions will be explained on section 3.1

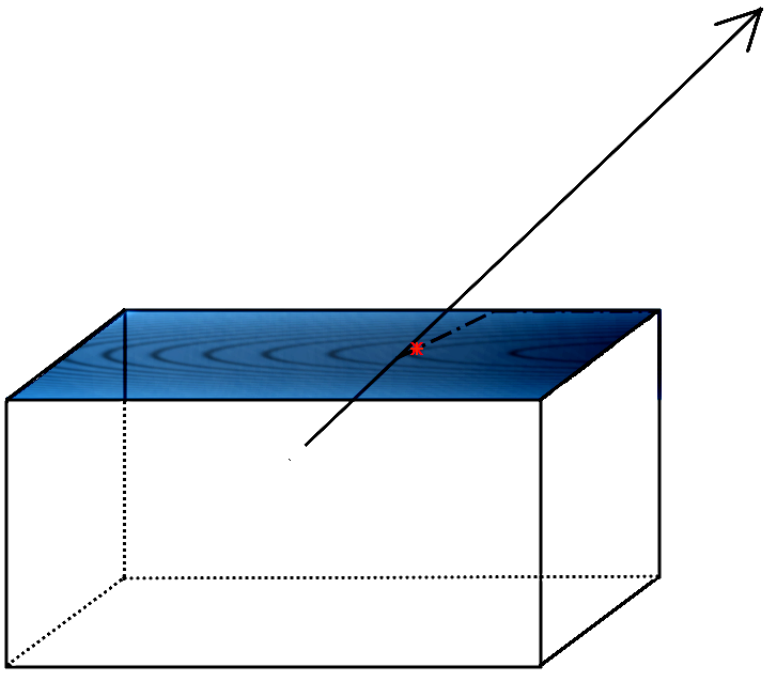

FIGURE 2.1: The arrow represents the direction of the negative gradient. The dotted path represents the projected gradient path. The contours represent the level sets of the model. The optimal point (the '*' in red) is the Cauchy point x^c

conditions. A restriction on the step length is added so that the next iteration stays feasible. Sometimes it is not possible to satisfy the Wolfe condition due to the bounded nature of the problem, so in these cases, only the Armijo condition needs to be satisfied. Once this step length is found, the next step is to check the termination condition. If the termination condition fails, a new gradient projection and subspace minimization will be needed and the method repeats. If the termination condition is successful, the program exits with an appropriate exit message.

Chapter 3

Modifications to the L-BFGS-B Algorithm

We made three main changes to the original `L-BFGS-B` algorithm. They concern the line search Wolfe conditions, the line search methodology, and the termination condition.

3.1 The Armijo and Wolfe conditions

It is accepted that the Armijo and Wolfe conditions work very well whenever the function f is smooth [LF01]. The Armijo condition, also known as the sufficient decrease requirement in the direction d_k, is defined as

$$f(x_k + \alpha_p d_k) \leq f(x_k) + c_1 \alpha_k d_k^T \nabla f(x_k) \tag{3.1}$$

where $0 < c_1 < 1$ is a constant, often $c_1 = 10^{-4}$ [NW99]. This condition guarantees "sufficient decrease" of the function. It is possible to continue decreasing without ever reaching the optimum if the Armijo condition is not required as is shown in Figure 3.1.

The other condition, which is the one that was actually changed, is the curvature condition, of which the most popular version is the "strong Wolfe" curvature condition:

$$|d_k^T \nabla f(x_k + \alpha_k d_k)| \leq c_2 |d_k^T \nabla f(x_k)| \tag{3.2}$$

Here d_k represents the search direction and c_2 is a constant such that $0 < c_1 < c_2 < 1$; often $c_2 = 0.9$ [NW99]. The strong Wolfe condition is a natural choice for optimization

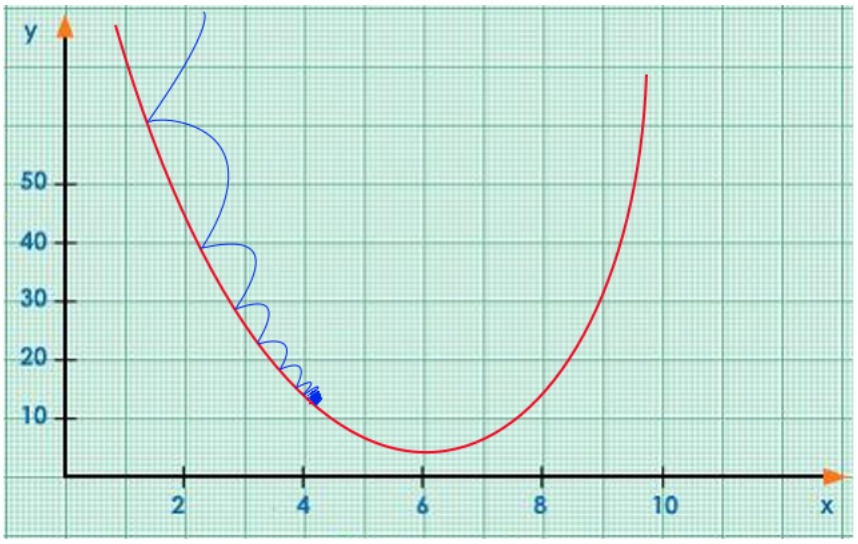

FIGURE 3.1: In this figure, the iterations always reduce the value of the function a little bit, but never enough to go below 12

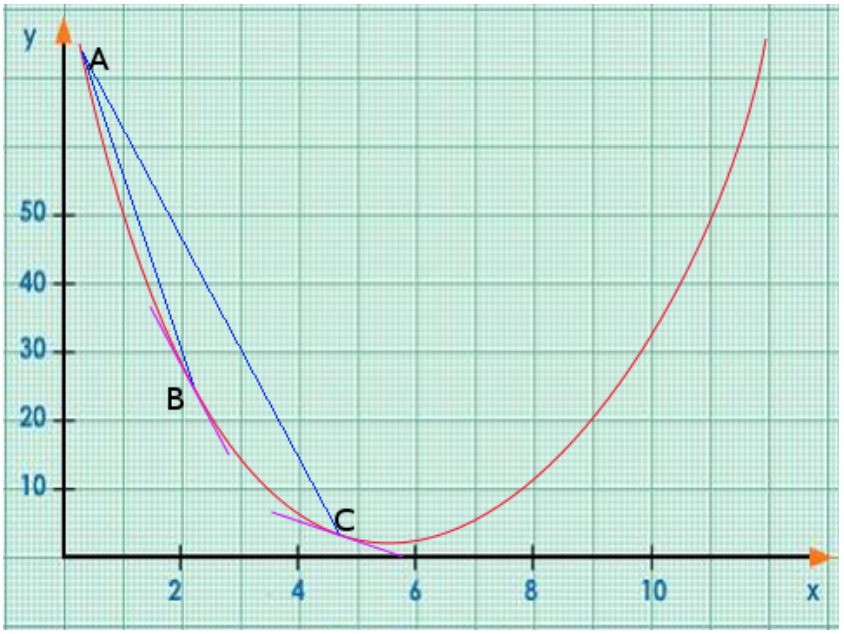

FIGURE 3.2: The logic of the Wolfe conditions is this. Starting at point A, Point B is a step in the right direction, however, point C offers a "flatter" tangent and should be closer to the optimum which has a tangent of zero (Smooth case).

of smooth functions. Its goal is to find a step length long enough that the slope has been reduced "sufficiently" as illustrated in figure 3.2, but the problem is that the condition, as it is, does not work well for the non-smooth case. This is because near the minimal points there may be abrupt changes in the gradient. A good example of this problem is the function $f(x) = |x|$, where the slope never becomes flat near the optimal point.

The weak Wolfe condition defined as

$$d_k^T \nabla f(x_k + \alpha_k d_k) \geq c_2 d_k^T \nabla f(x_k) \tag{3.3}$$

can be used in the non-smooth case. It is all that is needed to guarantee that the `BFGS` updated inverse Hessian approximation is positive definite [NW99]. This weak version is suited for the problems in this thesis and it was implemented as part of the line search algorithm explained in the next section.

3.2 The Line Search Methodology

The original FORTRAN software [ZBNM11] contains a line search subroutine. It was partially changed for the purpose of this thesis. The old version of the code was commented out.

The change in the Wolfe condition has already been described, but there is an additional problem with the function *dcstep* in the non-smooth case. The function *dcstep* was designed to work only with smooth functions in mind. The algorithm in *dcstep* takes advantage of quadratic and cubic approximations to the function in order to calculate step lengths that satisfy Armijo and Wolfe conditions. These second and third order approximations do not work well in the non-smooth case, and the optimizer breaks down using the line search as it is.

The solution to this particular issue is to use a line search similar to the one suggested in [LO13] and in [OS12]. This approach is to double the step length while the Armijo condition is violated, and once the interval has been bracketed, do bisection until both the Armijo and Wolfe conditions are satisfied. The only difference with the approach in this thesis is that the line search in HANSO can double its step length up to 30 times, whereas in this thesis, the step length can double only as long as it is less than the maximum value that guarantees feasibility of the solution (the maximum established in the first step of the original line search). Also, as in the HANSO line search, the number of bisections is limited to 30.

3.3 The Termination Condition

In the case of smooth functions, `L-BFGS-B` checks whether the algorithm has converged by means of the *projected gradient* which is nothing but the projection of the negative gradient onto the bounding box defined by l and u. If this projected gradient has a small norm the algorithm terminates. In the case of non-smooth functions however, the

function at the minimum may have a "wedge". In this wedge the projected gradient may not vanish (it is not defined at the "bottom" of the wedge, such as is the case for $f(x) = |x|$ at $x = 0$). Furthermore, if there is a sequence of points that approaches the optimum x in a direction $\vec{p}$, the projected gradients corresponding to this sequence of points might be completely different from the projected gradients associated with a sequence of points that approach the optimum x from a different direction.

3.3.1 Termination Condition Sub-algorithm

Lewis and Overton formulate an algorithm that gives a practical solution to this problem in section 6.3 of [LO13] in the case of unconstrained non-smooth optimization. They suggest computing the norm of the smallest vector that is part of the convex hull of gradients evaluated at points near the optimum candidate x and terminate if this is sufficiently small. The neighborhood is defined as those points at which the gradient has already been evaluated with a distance to x smaller than a small tolerance $\tau_x > 0$ and no more than $J \in \mathbb{N}$ iterations back in history. This list of gradients is referred to as the set $\mathcal{G}$ [LO13].

With this list $\mathcal{G}$ of gradients at hand, the next step is to find the vector with the minimal norm contained in the convex hull of these gradients. If the minimum such norm is smaller than another tolerance τ_d, the algorithm terminates.

In order to find this vector, there is the need to solve a quadratic problem. Every vector in the convex hull can be expressed as a convex combination Gz of those vectors in $\mathcal{G}$, where G is the matrix with columns made up of gradients in $\mathcal{G}$ and z is such that $\sum z_i = 1$ and $z_i \geq 0$.

The objective is to find the right combination of z that minimizes the norm $||Gz||_2$. This is equivalent to solving the following optimization problem

$$\begin{aligned} \min \quad & q(z) = ||Gz||_2^2 = z^T G^T G z \\ \text{s.t.} \quad & \sum z_i = 1 \\ & z_i \geq 0. \end{aligned} \tag{3.4}$$

The solution to this problem z^* defines the associated vector Gz^*, so if $||Gz^*||_2 < \tau_d$ the algorithm terminates.

In the bound-constrained case, it is important to notice that instead of the gradient we have to work with the projected gradient. In the unconstrained case, if a component of the gradient is not zero this yields a direction of descent. But in the bounded case, it

may be impossible to reduce f in this direction because the boundary may have been reached. For this reason the gradients have to be projected onto the bounding box, and it is these projected gradients that we incorporate in the termination condition of `L-BFGS-B-NS`.

3.3.2 The Solution of the Quadratic Program

The solution of the quadratic program (3.4) is obtained using a practical primal-dual method. This is the same method implemented by Skajaa [Ska10] in his thesis. His code qpspecial was implemented in FORTRAN for this thesis. The method is the well known Mehrotra's Predictor-Corrector algorithm applied to quadratic programming, as explained in chapter 16 of [NW99].

Chapter 4

Experimental Results

The L-BFGS-B implementation was tested on the high performance cluster machines at NYU. In order to run these tests it was necessary to create a series of PBS files[1] using a PBS generator script. This script generator created PBS files which in turn run bash shell scripts[2]. Several of these shell scripts are available at the repository [Hen14]. The main reason to run scripts this way is because it achieves parallelism, and because the system sends confirmation e-mails and statistics about the different stages of the processes giving a lot of control to the practitioner.

4.1 Exit Messages

The original L-BFGS-B optimizer displays different messages depending on the condition that triggered the exit. The following is a list of some of the most common exit messages in the original L-BFGS-B optimizer along with an explanation of how we generalized them in L-BFGS-B-NS.

- "ABNORMAL_TERMINATION_IN_LNSRCH" This message means that there was a problem with the line search and the program's exit was premature. In L-BFGS-B, it typically occurs for non-smooth functions where the line search breaks down. In L-BFGS-B-NS, it typically occurs when the limit on the number of bisections in the line search (30) is exceeded.

[1] PBS stands for Portable Batch System. This is software that performs job scheduling. It is used by High Performance Computing at NYU (and many other High Performance Computing Centers) to allocate computational tasks. In order to run jobs at the high performance clusters, a series of PBS batch files need to be created.

[2] Bash is a command processor. Each Bash script that was created includes a series of computer commands, namely, execution of the original L-BFGS-B software and the new code L-BFGS-B-NS.

- "CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_LT_PGTOL": Means that convergence was achieved because the norm of the projected gradient is small enough. We made just one change to the original `L-BFGS-B` code: in order to have results that are comparable with the results obtained with the new code, we terminated when the 2-norm of the projected gradient was less than the tolerance $\tau = 10^{-6}$, instead of the ∞-norm. Notice that this convergence message does not apply to `L-BFGS-B-NS` because of particular requirements for non-smooth functions involving the convex hull of projected gradients as explained in section 3.3. Instead it is replaced by:

- "CONVERGENCE: ZERO_GRAD_IN_CONV_HULL" This means that the termination condition discussed in section 3.3 was satisfied[3]. We set $\tau_d = 10^{-6}$, $\tau_x = 10^{-3}$ and $J = 10$ the maximum number of gradients in the optimality check.

- "CONVERGENCE: REL_REDUCTION_OF_F_LT_FACTR*EPSMCH": This convergence condition is achieved whenever the relative reduction of the value of function f is smaller than a predefined factor times the machine precision ϵ. This exit message does not apply to our tests. It was disabled by setting the factor "FACTR" to zero, both in our runs of `L-BFGS-B-NS` and our tests using the original code `L-BFGS-B`.

The limit on the number of iterations was set to 10000.

4.2 Modified Rosenbrock Function

Consider a modified version of the Rosenbrock function problem [Ros61]:

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} |x_i - x_{i-1}^2|^p \tag{4.1}$$

We can study the properties of function f based on the properties of the function $\phi(t_i)$, where $\phi(t_i) = |t_i|^p$ and $t_i = x_i - x_{i-1}^2$. The properties of the function depend on the value of the p parameter[4]. This function can be proven to be locally Lipschitz continuous whenever $p \geq 1$. However, its second derivative blows up at zero whenever $p < 2$. Note that although $\phi(t_i)$ is convex for $p \geq 1$, f is not convex.

[3] This does not mean that the resulting vector is exactly equal to zero, but it is small enough to satisfy the termination condition.

[4] The original Rosenbrock function had values of $p = 2$, $n = 2$ and the second term was multiplied by 100.

The properties of $\phi(t_i)$ can be separated into different cases. Whenever $p > 1$ the derivative can be represented as:

$$\frac{d}{dt}\phi(t) = \pm p|t|^{p-1} \tag{4.2}$$

and therefore, the limit of the derivative exists and is equal to zero near $t = 0$:

$$\lim_{t\to 0} \frac{d}{dt}\phi(t) = 0$$

From this we conclude that f has a smooth first derivative for $p > 1$.

However, if $p = 1$, $\phi(t) = |t|$, and the absolute value function is not differentiable at $t = 0$. Note that in this case, $\phi(t)$ is Lipschitz continuous at $t = 0$.

The second derivative provides a bit more of information.

$$\frac{d^2}{dt^2}\phi(t) = \pm p(p-1)|t|^{p-2} \tag{4.3}$$

If $p \geq 2$ the function is twice continuously differentiable. However if $p < 2$, the second derivative becomes $\frac{p(p-1)}{|t|^q}$, where $q = 2 - p > 0$, and this second derivative blows up as $|t| \to 0$. The special case $p = 1$ has second derivative equal to zero since $p(p-1) = 0$ except at $t = 0$ where it is undefined. For $p < 1$, ϕ is not Lipschitz continuous at $t = 0$.

Having explained the characteristics of the function, the next thing that needs to be defined is the region to be tested. We chose the region to be defined by the "box" with boundaries

$$x_i = \begin{cases} [-100, 100] & \text{if } i \in \text{ even numbers} \\ [10, 100] & \text{if } i \in \text{ odd numbers} \end{cases} \tag{4.4}$$

The initial point was chosen to be the midpoint of the box, plus a different small perturbation for each dimension, chosen so that the line search does not reach the boundary of several dimensions in one step:

$$x_i = \frac{u_i + l_i}{2} - \left(1 - 2^{1-i}\right) \tag{4.5}$$

It is important to note that this choice of initial point makes the problem more difficult to solve. The problem is easier if the midpoint is chosen.

p	n	m	L-BFGS-B results				L-BFGS-B-NS results			
			iters.	#fg	f	NPG	iters.	#fg	f	NSVCHPG
2	100	5	29	131	452116.014385974	2.92E-06	34	67	452116.014385974	1.46E-08
2	100	10	25	67	452116.014385974	7.37E-05	32	45	452116.014385974	2.29E-07
2	100	20	25	28	452116.014385974	8.97E-07	29	47	452116.014385974	1.20E-04
2	200	5	32	74	913376.515331672	1.97E-07	34	62	913376.515331677	8.44E-07
2	200	10	27	32	913376.515331672	5.26E-07	32	43	913376.515331672	1.04E-07
2	200	20	26	29	913376.515331672	5.80E-07	33	58	913376.515331677	3.98E-08
2	1000	5	26	68	4603460.52289722	2.61E-04	37	80	4603460.52289732	9.85E-07
2	1000	10	26	71	4603460.52289722	5.93E-04	33	45	4603460.52289733	5.89E-07
2	1000	20	30	95	4603460.52289722	1.18E-05	33	59	4603460.52289732	9.02E-07
2	5000	5	27	122	23053880.5607232	4.44E-03	23	41	23053880.5607253	2.19E-07
2	5000	10	25	28	23053880.5607256	8.96E-07	32	40	23053880.5607253	8.40-07
2	5000	20	26	80	23053880.560724	3.75E-03	17	43	23053880.5607232	1.08E-07
2	10000	5	26	68	46116905.6080045	2.80E-06	12	232	46116905.6079994	5.41E-06
2	10000	10	25	67	46116905.6080057	7.70E-03	18	297	46116905.6080044	3.66E-05
2	10000	20	28	73	46116905.608006	5.18E-06	18	297	46116905.6080044	3.66E-05

TABLE 4.1: Satisfactory results for the original algorithm `L-BFGS-B` and for `L-BFGS-B-NS` applied to the Modified Rosenbrock function with $p = 2$. NPG: Norm of Projected Gradient with tolerance 10^{-6}. NSVCHPG: Norm of Smallest Vector in Convex Hull of Projected Gradients with $\tau_d = 10^{-6}, \tau_x = 10^{-3}$

We tested both the original `L-BFGS-B` optimizer and the new code `L-BFGS-B-NS` on the modified Rosenbrock function with p varying between 2 and 0.9 and n varying between 100 and 10000, with the variable memory parameter m set to 5, 10 and 20.

4.2.1 Performance of `L-BFGS-B` and `L-BFGS-B-NS` on the Modified Rosenbrock Function

In the tables, iters shows the number of iterations, $\#fg$ shows the number of function and gradient evaluations taken, and f shows the final computed function value that was achieved by the optimization. In the case of `L-BFGS-B`, *NPG* shows the 2-norm of the projected gradient with termination tolerance 10^{-6}. In the case of `L-BFGS-B-NS`, *NSVCHPG* shows the norm of the smallest vector in the convex hull of projected gradients.

In table 4.1 we can see that since the test function is smooth when $p = 2$, `L-BFGS-B` has no problems minimizing it and that `L-BFGS-B-NS` reaches exactly the same final values, although because of the line search changes, the number of iterations and function and gradient evaluations usually increase.

On the other hand, in table 4.2 we see that the value of $p = 1$ leads to an abnormal line search termination for `L-BFGS-B` in all of the cases presented. This is to be expected as the function is non-smooth. In fact, for all cases `L-BFGS-B` terminates at the first iteration because of breakdown in the line search. `L-BFGS-B-NS` on the other hand is able to converge under most scenarios. In fact, it is possible to converge under all scenarios by tweaking the starting point of the optimization.

p	n	m	L-BFGS-B results				L-BFGS-B-NS results			
			iters.	#fg	f	NPG	iters.	#fg	f	NSVCHPG
1	100	5	1	21	151292.8	7.79E+02	15	130	4826.1066601788	1.34E-08
1	100	10	1	21	151292.8	7.79E+02	15	129	4826.1066352341	1.34E-08
1	100	20	1	21	151292.8	7.79E+02	15	129	4826.1066352341	1.34E-08
1	200	5	1	21	299792.8	1.10E+03	15	128	9668.0522943829	1.82E-08
1	200	10	1	21	299792.8	1.10E+03	15	128	9668.0522930362	1.82E-08
1	200	20	1	21	299792.8	1.10E+03	15	112	9667.9345180734	1.19E-07
1	1000	5	1	21	1487792.8	2.46E+03	23	193	48403.1390323475	5.72E-09
1	1000	10	1	21	1487792.8	2.46E+03	16	160	48403.3203939957	2.44E-08
1	1000	20	1	21	1487792.8	2.46E+03	16	160	48403.320394002	2.44E-08
1	5000	5	1	21	7427792.8	5.51E+03	20	127	242078.712084738	1.26E-08
1	5000	10	1	21	7427792.8	5.51E+03	56	339	242078.839910433	1.26E-08
1	5000	20	1	21	7427792.8	5.51E+03	45	249	242078.560631846	7.84E-08
1	10000	5	1	21	14852792.8	7.79E+03	18	148	484172.781463252	8.25E-08
1	10000	10	1	21	14852792.8	7.79E+03	10000	20019	484269.73074638832	3.76E+02
1	10000	20	1	21	14852792.8	7.79E+03	21	101	484172.918293261	1.77E-08

TABLE 4.2: Unsatisfactory results for the original algorithm `L-BFGS-B` applied to the Modified Rosenbrock function with $p = 1$, but converging results for `L-BFGS-B-NS`. NPG: Norm of Projected Gradient with tolerance $= 10^{-6}$. NSVCHPG: Norm of Smallest Vector in Convex Hull of Projected Gradients with $\tau_d = 10^{-6}, \tau_x = 10^{-3}$

p	n	m	L-BFGS-B results				L-BFGS-B-NS results			
			iters.	#fg	f	NPG	iters.	#fg	f	NSVCHPG
2	200	5	32	74	913376.515331672	1.97E-07	35	55	913376.515331676	3.19E-09
2	200	10	27	32	913376.515331672	5.26E-07	20	41	913376.515331677	3.98E-07
2	200	20	26	29	913376.515331672	5.80E-07	19	40	913376.515331672	8.03E-07
1.5	200	5	8	50	95144.1877450699	9.60E+02	29	68	94261.6310280216	7.52E-07
1.5	200	10	8	50	95095.5635531693	9.61E+02	30	59	94261.6310280212	9.69E-07
1.5	200	20	8	50	95095.5635531693	9.61E+02	26	66	94261.6310280211	9.95E-07
1.1	200	5	1	21	658485.96769483	1.10E+03	26	75	15226.525226329	4.24E-07
1.1	200	10	1	21	658485.96769483	1.10E+03	34	107	15226.5210644821	1.16E-07
1.1	200	20	1	21	658485.96769483	1.10E+03	38	99	15226.5209960549	1.73E-07
1.01	200	5	1	21	324235.017102379	1.10E+03	31	305	10218.0196721806	3.64E+01
1.01	200	10	1	21	324235.017102379	1.10E+03	47	151	10116.5275434197	7.29E-07
1.01	200	20	1	21	324235.017102379	1.10E+03	29	123	10116.5603888173	2.95E-09
1.001	200	5	1	21	302150.58179968	1.10E+03	36	111	9711.8763115237	5.70E-08
1.001	200	10	1	21	302150.58179968	1.10E+03	23	100	9711.8906439951	2.81E-09
1.001	200	20	1	21	302150.58179968	1.10E+03	39	164	9711.876311317	1.41E-07
1.0001	200	5	1	21	300027.736327598	1.10E+03	306	638	9672.3210642275	5.09E-07
1.0001	200	10	1	21	300027.736327598	1.10E+03	17	96	9672.3639815678	1.82E-08
1.0001	200	20	1	21	300027.736327598	1.10E+03	19	96	9672.3922445339	2.80E-09
1.00001	200	5	1	21	299816.285236336	1.10E+03	27	96	9668.3934739514	4.32E-07
1.00001	200	10	1	21	299816.285236336	1.10E+03	15	80	9668.373073478	2.80E-09
1.00001	200	20	1	21	299816.285236336	1.10E+03	15	80	9668.3730743134	2.80E-09
1	200	5	1	21	299792.8	1.10E+03	15	128	9668.0522943829	1.82E-08
1	200	10	1	21	299792.8	1.10E+03	15	128	9668.0522930362	1.82E-08
1	200	20	1	21	299792.8	1.10E+03	15	112	9667.9345180734	1.19E-07

TABLE 4.3: Results for different values of p, with $n = 200$.

Several other values of p were also tested. In Table 4.3, the parameter p is varied and for fixed $n = 200$. With a tolerance of 10^{-6} `L-BFGS-B` always fails as expected, and those values where p is closer to 1 are the most difficult for the original algorithm to handle. Values generated via `L-BFGS-B-NS` are comparatively better whenever $p < 2$, since the function is "less" smooth. Most runs of `L-BFGS-B-NS` converge using the termination condition from section 3.3.

Some runs with a value of $p = 0.999$ are shown in Table 4.4. As can be seen, both algorithms fail in the sense that the termination criteria are never met, but `L-BFGS-B-NS` reaches a better feasible solution in every scenario. The same thing can be said for $p = 0.99$ on table 4.5 and for $p = 0.9$ on table 4.6.

p	n	m	L-BFGS-B results				L-BFGS-B-NS results			
			iters.	#fg	f	NPG	iters.	#fg	f	NSVCHPG
0.999	100	5	1	21	150123.179035242	7.79E+02	10000	20003	4900.9128213197	3.86E+01
0.999	100	10	1	21	150123.179035242	7.79E+02	10000	19999	4900.9123782223	3.79E+01
0.999	100	20	1	21	150123.179035242	7.79E+02	10000	20000	4900.8873111184	3.78E+01
0.999	200	5	1	21	297453.671572579	1.10E+03	10000	29971	9720.7074076621	5.50E+01
0.999	200	10	1	21	297453.671572579	1.10E+03	10000	19999	9720.7073593488	5.41E+01
0.999	200	20	1	21	297453.671572579	1.10E+03	10000	20000	9720.7067337013	5.39E+01
0.999	1000	5	1	21	1476097.61187127	2.46E+03	10000	29961	48279.0637949643	9.94E-01
0.999	1000	10	1	21	1476097.61187127	2.46E+03	10000	20000	48279.0637881564	1.68E+02
0.999	1000	20	1	21	1476097.61187127	2.46E+03	10000	20000	48279.0637186514	1.66E+02
0.999	5000	5	1	21	7369317.31336543	5.51E+03	10000	29983	241070.845631957	9.94E-01
0.999	5000	10	1	21	7369317.31336543	5.51E+03	10000	29983	241070.845630635	9.94E-01
0.999	5000	20	1	21	7369317.31336543	5.51E+03	10000	20005	241070.845626631	2.73E+02
0.999	10000	5	1	21	14735841.9402302	7.79E+03	10000	29981	482060.572922137	3.89E+02
0.999	10000	10	1	21	14735841.9402302	7.79E+03	10000	29983	482060.572921515	9.94E-01
0.999	10000	20	1	21	14735841.9402302	7.79E+03	10000	20003	482060.572910768	5.28E+02

TABLE 4.4: Results for $p = 0.999$, and non-converging but better results for `L-BFGS-B-NS`; NPG: Norm of Projected Gradient with tolerance $= 10^{-6}$, never satisfied. NSVCHPG: Norm of Smallest Vector in Convex Hull of Projected Gradients with $\tau_d = 10^{-6}, \tau_x = 10^{-3}$, also, never satisfied

p	n	m	L-BFGS-B results				L-BFGS-B-NS results			
			iters.	#fg	f	NPG	iters.	#fg	f	NSVCHPG
0.99	100	5	1	21	140004.324489439	7.79E+02	10000	29999	4706.5690751224	9.46E-01
0.99	100	10	1	21	140004.324489439	7.79E+02	10000	29983	4706.5690446185	9.46E-01
0.99	100	20	1	21	140004.324489439	7.79E+02	10000	29989	4706.5690446185	9.46E-01
0.99	200	5	1	21	277216.896653442	1.10E+03	10000	29985	9332.020553172	9.46E-01
0.99	200	10	1	21	277216.896653442	1.10E+03	10000	20009	9332.0205286749	6.56E+01
0.99	200	20	1	21	277216.896653442	1.10E+03	10000	20007	9332.0182250141	8.21E+01
0.99	1000	5	1	21	1374917.47396547	2.46E+03	10000	29993	46335.6319951054	9.46E-01
0.99	1000	10	1	21	1374917.47396547	2.46E+03	10000	29993	46335.6319927555	9.46E-01
0.99	1000	20	1	21	1374917.47396547	2.46E+03	10000	20009	46335.6319815415	1.92E+02
0.99	5000	5	1	21	6863420.36052535	5.51E+03	10000	29995	231353.689126569	9.46E-01
0.99	5000	10	1	21	6863420.36052535	5.51E+03	10000	29995	231353.689125989	9.46E-01
0.99	5000	20	1	21	6863420.36052535	5.51E+03	10000	29995	231353.689125497	9.46E-01
0.99	10000	5	1	21	13724048.9687287	7.79E+03	10000	20013	462626.260534309	4.91E+02
0.99	10000	10	1	21	13724048.9687287	7.79E+03	10000	20013	462626.260533983	4.91E+02
0.99	10000	20	1	21	13724048.9687287	7.79E+03	10000	20013	462626.260533741	4.91E+02

TABLE 4.5: Results for $p = 0.99$; similar to table 4.4

p	n	m	L-BFGS-B results				L-BFGS-B-NS results			
			iters.	#fg	f	NPG	iters.	#fg	f	NSVCHPG
0.9	100	5	1	21	70247.1102599127	7.79E+02	10000	29985	3145.9378051899	7.82E+01
0.9	100	10	1	21	70247.1102599127	7.79E+02	10000	20005	3145.9378011472	4.17E+02
0.9	100	20	1	21	70247.1102599127	7.79E+02	10000	20007	3145.9375231332	2.66E+02
0.9	200	5	1	21	137705.665344048	1.10E+03	10000	29983	6210.7940850593	5.70E-01
0.9	200	10	1	21	137705.665344048	1.10E+03	10000	29987	6210.7940839115	5.70E-01
0.9	200	20	1	21	137705.665344048	1.10E+03	10000	20007	6210.793392882	3.72E+02
0.9	1000	5	1	21	677374.106017129	2.46E+03	10000	29997	30729.6443168733	2.49E+02
0.9	1000	10	1	21	677374.106017129	2.46E+03	10000	29999	30729.6443166765	5.70E-01
0.9	1000	20	1	21	677374.106017129	2.46E+03	10000	20013	30729.6443164162	1.58E+03
0.9	5000	5	1	21	3375716.30938256	5.51E+03	10000	29993	153323.895471387	5.70E-01
0.9	5000	10	1	21	3375716.30938256	5.51E+03	10000	29993	153323.895471342	5.70E-01
0.9	5000	20	1	21	3375716.30938256	5.51E+03	10000	29993	153323.895471246	5.70E-01
0.9	10000	5	1	21	6748644.0635896	7.79E+03	10000	29999	306566.709414405	5.70E-01
0.9	10000	10	1	21	6748644.0635896	7.79E+03	10000	29999	306566.709414387	5.70E-01
0.9	10000	20	1	21	6748644.0635896	7.79E+03	10000	29999	306566.709414375	5.70E-01

TABLE 4.6: Results for $p = 0.9$; similar to table 4.4

Conclusions

We conclude that the new code `L-BFGS-B-NS` works well on the modified Rosenbrock function, giving much better results than the original `L-BFGS-B` code for $p < 2$, when the function is not twice continuously differentiable, particularly for values of p close to or equal to 1. For $p \geq 1$, the algorithm typically terminates when the *NSVCHPG* holds: this is an approximate first-order non-smooth optimality condition following a suggestion of [LO13]. Even for $p < 1$, the computed function values seem reasonable, although we cannot verify this by the *NSVCHPG* condition since the gradients blow up when f is not Lipschitz. We hope the new code will be useful to users who wish to solve large-scale non-smooth bound-constrained optimization problems.

Appendix A

Running L-BFGS-B-NS

A.1 Running tests in local machines

In order to run the software, a copy of the files is required. The easiest way to get a copy is downloading directly from the repository [Hen14] and clicking on the download zip link. Users of git can also clone the repository by issuing either the Hypertext Transfer Protocol Secure command `git clone https://github.com/wilmerhenao/L-BFGS-B-NS.git` or the secure shell command `git clone git@github.com:wilmerhenao/L-BFGS-B-NS.git`. Other requirements on the machine are a `FORTRAN` compiler and `LAPACK`.

Once the user has obtained a local copy of the code, new executables need to be created. A simple `Makefile` has been provided; in order to "make" the executables and run a typical test with parameters $p = 1.1$, $n = 100$, $m = 5$ and $\tau_d = 10^{-6}$, the user should issue the following commands.

```
$ make
$ ./rosenbrockp 1.1 100 5 1d-6
```

Output by default goes directly to the screen. The best way to capture the results in a text file is using the redirection operator >.

```
$ make
$ ./rosenbrockp 1.1 100 5 1d-6 > mysampleresults.txt
```

If the user is running several tests, a bash script might be necessary.

```
#!/bin/bash
for ptol in 1d-6
do
    for p in 2 1 1.5 1.1 1.01 1.001 1.0001 1.00001 0.99 0.9
    do
        for n in 2 4 6 8 10 20 50 100 200 1000 5000 10000
```

```
        do
            for m in 5 10 20
            do
                echo $ptol $p $n $m
                ./rosenbrockp $p $n $m $ptol >> OUTPUTS/res1d6.txt
            done
        done
    done
done

exit 0;
```

This bash script can be made executable and run directly on the user's machine. All the results from the runs will be located on file `OUTPUTS/resid6.txt` in this case.

```
$ chmod +x runall.sh
$ ./runbatch.sh
```

A.2 Running on High Performance Computer Clusters

The requirements are different on the high performance computing cluster, but the standard is to use Portable Batch System *PBS* files. They allow the user to get detailed information of the tests via e-mails and provide the user with the ressources to run larger problems.

```
#!/bin/bash

#PBS -l nodes=1:ppn=8,walltime=48:00:00
#PBS -m abe
#PBS -M youremail@nyu.edu
#PBS -N rosenbrockHD9

module load gcc/4.7.3

cd /scratch/weh227/rosenbrock/
for ptol in 1d-6
do
  for p in 2 1 1.5 1.1 1.01 1.001 1.0001 1.00001 0.99 0.9
  do
      for n in 2 4 6 8 10 20 50 100 200 1000 5000 10000 100000 1000000
      do
          for m in 5 10 20
          do
              echo $ptol $p $n $m
              ./rosenbrockp $p $n $m $ptol >> res1d6.txt
          done
      done
  done
done
```

```
  exit 0;
```

Typically the user logs into the clusters, runs the tests using a software called *qsub* and picks up the results once the tests have finished. At NYU a user logs into the hpc access cluster, followed by the bowery computer cluster.

```
   $ ssh youremail@hpc.nyu.edu
  $ ssh bowery
  $ password: ********
  $ git clone https://www.github.com/wilmerhenao/L-BFGS-B-NS.git
  $ cd L-BFGS-B-NS
  $ make
  $ qsub -o precision1d6.log -j oe precision1d6.pbs
```

It is also possible to run several *PBS* jobs simultaneously. Your local High Performance Computer Cluster always has some documentation on how to create and use *PBS* files. https://wikis.nyu.edu/display/NYUHPC/Tutorial+-+Submitting+a+job+using+qsub

A.3 Specifying the Function and the Gradient

The user can specify a new function and its corresponding gradient. Just change lines 222 to 234 in file `DriverRosenbrockp.f90`

```
222    f=((x(1)-1d0)**2)
223   g(1)=2d0*(x(1)-1d0)
224
225   do 20 i=1, (n-1)
226     z=x(i+1)-x(i)**2
227     f=f+abs(z)**p
228     r1=p * abs(z)**(p - 1)
229     if (z < 0) then
230       r1 = -r1
231     endif
232     g(i+1)=r1
233     g(i)=g(i)-2d0*x(i)*r1
234 20 continue
```

Boundaries and starting points are also specified in this file in between lines 95 and 119

Bibliography

[BLNZ95] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ci You Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

[Bro70] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. II. The new algorithm. *J. Inst. Math. Appl.*, 6:222–231, 1970.

[CGT88] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM J. Numer. Anal.*, 25(2):433–460, 1988.

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

[Gol70] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Math. Comp.*, 24:23–26, 1970.

[Hen14] Wilmer Henao. L-BFGS-B-NS. https://github.com/wilmerhenao/L-BFGS-B-NS, 2014.

[LF01] Dong-Hui Li and Masao Fukushima. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM J. Optim.*, 11(4):1054–1064 (electronic), 2001.

[LO13] Adrian S. Lewis and Michael L. Overton. Nonsmooth optimization via quasi-Newton methods. *Math. Program.*, 141(1-2, Ser. A):135–163, 2013.

[MT89] Jorge J. Moré and Gerardo Toraldo. Algorithms for bound constrained quadratic programming problems. *Numer. Math.*, 55(4):377–400, 1989.

[Noc80] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comp.*, 35(151):773–782, 1980.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999.

[OS12] Michael Overton and Anders Skajaa. Hanso 2.02. http://www.cs.nyu.edu/faculty/overton/software/hanso/, 2012.

[Ros61] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comput. J.*, 3:175–184, 1960/1961.

[Sha70] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, 24:647–656, 1970.

[Ska10] Anders Skajaa. Limited memory BFGS for nonsmooth optimization. Master's thesis, New York University, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012, 2010.

[ZBLN97] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Software*, 23(4):550–560, 1997.

[ZBNM11] Ciyou Zhu, Richard Byrd, Jorge Nocedal, and Jose Luis Morales. L-BFGS-B 3.0. http://www.ece.northwestern.edu/~nocedal/lbfgsb.html, 2011.