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

April 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*

Michael Overton
Department of Mathematics

Master of Science in Scientific Computing

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

by Wilmer Henao

The Thesis Abstract is written here . . .

# *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 Skaaja 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 ressources and their helpful assistance.

# Contents

# List of Figures

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

The `L-BFGS-B` algorithm [2] is a standard method for solving large instances of (1.1) when $f$ is a smooth function tipically, twice differentiable. The original name of `BFGS` stands for Broyden, Fletcher, Goldfarb and Shanno, the authors of the original `BFGS` quasi-Newton algorithm for unconstrained optimization discovered and published independently by them in 1970 [1, 4, 5, 14]. 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 [10], where the `L` stands for "Limited-Memory" and also for "Large" problems, is based on `BFGS` but requires only $\mathcal{O}(mn)$ operations per iteration, and requires 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 [17].

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 [6] 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 [17, 18] 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 [11]. 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 [11].

In the case of Non-Smooth functions, `BFGS` typically succeeds in finding a local minimizer [7]. 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 single 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 the inverse of the Hessian of $f$. Instead of storing a dense $n \times n$ matrix, L-BFGS keeps an approximation to 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. For this reason it is the algorithm of choice in this thesis.

## 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 during the search step phase 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 original algorithm was created for the case when $n$ is large and $f$ is smooth. Its first step is the gradient projection similar to the one outlined in [3, 9] 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:

[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 in this thesis

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

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 $u_i$ and $l_i$. 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 segment 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 $\overrightarrow{l}$ and $\overrightarrow{u}$. 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 stage two is necessary. Stage 2 or 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 two 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 solve the constrained problem (2.3), but only on those dimensions that are free. 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. After an unconstrained search direction has been found, the constraints are taken into account and the search direction

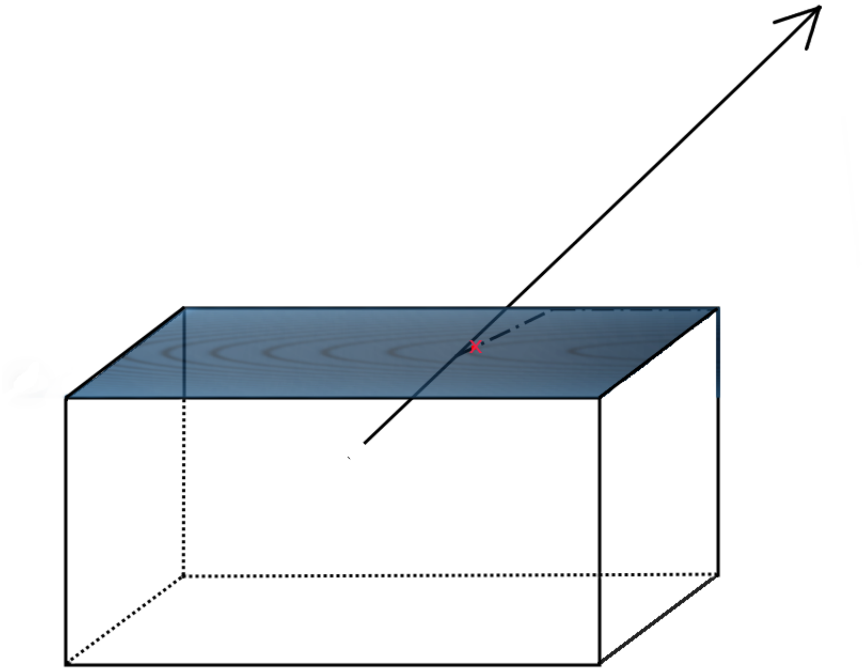


Figure 2.1: The arrow represents the direction of the gradient. The dotted path represents the projected gradient path from the gradient and onto the box. The region represents the level sets of the model. The optimal point (the 'x' in red) is the Cauchy point $x^c$

is restricted to the $l$, $u$ bounding box via a step length factor $\alpha^*$. The new direction and step length produced$\hat{d}^* = \alpha^* \hat{d}^u$ where $\alpha^*$ (the step length) is also chosen so that the new point $\bar{x}$ satisfies Armijo and weak Wolfe[3] conditions if that is possible, under some circumstances it may be only possible to satisfy the Armijo but not the Wolfe condition due to the bounded nature of the problem. A third restriction on the step length is added so that the next iteration stays feasible. Once this step is finished, the next and final step will be 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.

[3] Armijo and weak Wolfe conditions will be explained on section (3.1)

# 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 [8]. 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$c_1 = 10^{-4}$ [11]. This condition guarantees "sufficient decrease". 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 $c_2 = 0.9$ [11]. The strong Wolfe condition is a natural choice for optimization of

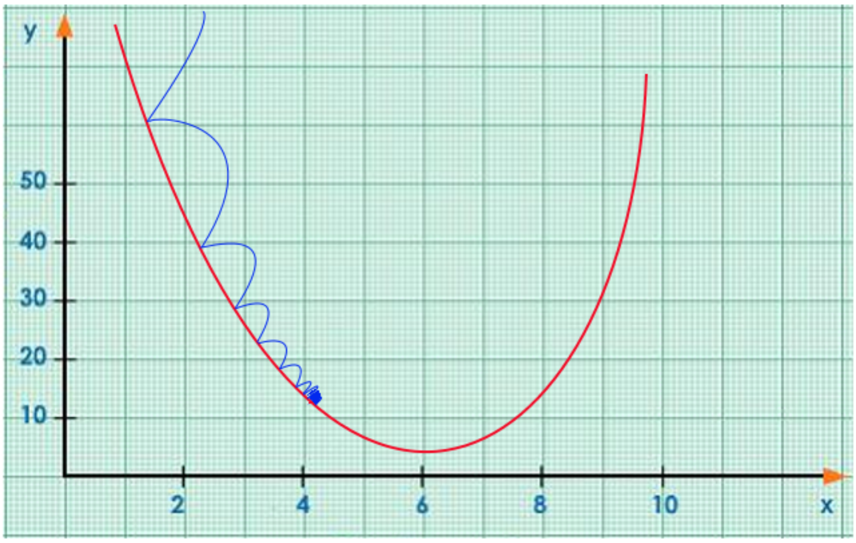


Figure 3.1: In this figure, the iterations always reduce the value of the function a little bit, but never enough to go below 12

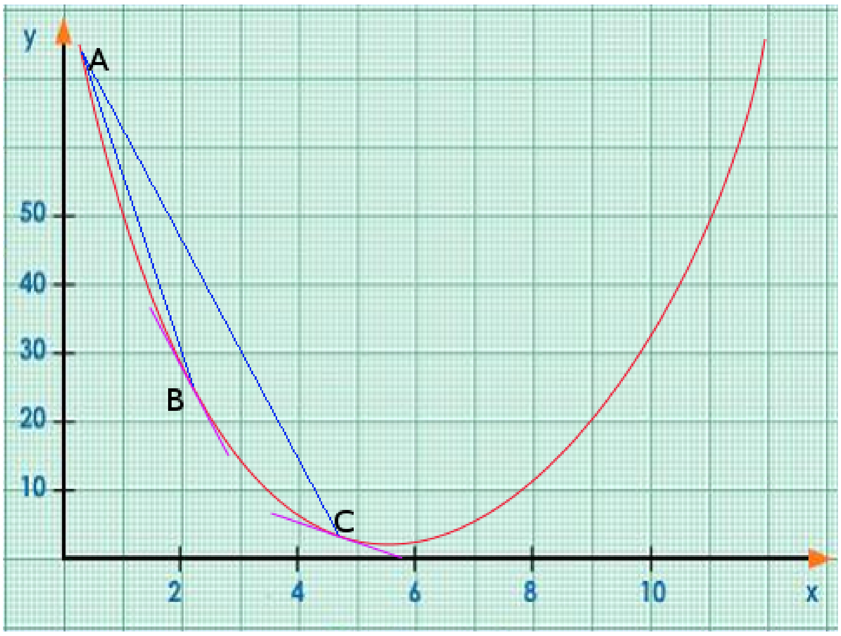


Figure 3.2: The logic of Wolfe conditions is this. Starting at point A. Point B is a step in the right direction, however, point C offers a "flatter" tangent and should be closer to the optimum which has a tangent of zero (Smooth case)

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 [7]. 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 [17] contains a line search subroutine. It was partially changed for the purpose of this thesis. The old version of the code was not removed, it was left commented out.

The old version has a problem with the function *dcstep* as it concerns this thesis. It turns out that function *dcstep* was designed to work only with smooth functions in mind. The algorithm takes advantage of quadratic and cubic approximations to the function in order to calculate step lengths that satisfy Armijo and Wolfe conditions. Unfortunately, these second and third order approximations do not work in the Non-Smooth case, and the optimizer crashes under the line search as it is. Function *dcstep* starts on line 3779 and a sample of the approximations is shown in between lines 3881 and 3902 of appendix (A.3) of `lbfgsbnomessages.f90`.

The solution to this particular issue is to use a line search similar to the one implemented in [12]. The HANSO 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). This version of the bisection and expansion is found in between lines 4425 and 4456 of `lbfgsbnomessages.f90` in appendix (A.4).

## 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*[1] which is nothing but the projection of the negative

[1] Not to be confused with “gradient projection” from subsection (2.3.1)

gradient onto the bounding box defined by $l$ and $u$. If this projected gradient has a small norm the algorithm converges. 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. 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 to a sequence of points that approach the optimum $x$ from an opposite direction.

Given this set of conditions, there is a need for particular rules to establish the finalization of the optimization, these rules have already been established in some works before [7, 15].

### 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 [7] In the case of unconstrained Non-Smooth optimization, the suggest computing the norm of the smallest vector that is part of the convex hull of projected 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 projected gradients is referred to as the set $\mathcal{G}$ [7].

With this list $\mathcal{G}$ of projected gradients at hand, the next step is to find the vector with the minimal norm contained in the convex hull of these projected gradients. If the norm of this vector has a norm smaller than $\tau_d$ , the algorithm ends with a message of convergence success "CONVERGENCE: ZERO_GRAD_IN_CONV_HULL". If the minimum such norm is larger than the tolerance, the algorithm must continue to the next iteration and not terminate.

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 projected gradients in $\mathcal{G}$ and $z$ is such that $\sum_{i=1}^{n} z_i = 1$ 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

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

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

### 3.3.2 The Solution of the Quadratic Program

The solution of the quadratic program (3.4) is obtained with a practical primal-dual methodology. This is the same methodology implemented by Skajaa [15] in his thesis. His code `qpspecial` was implemented in FORTRAN for this thesis and is part of `lbfgsbnomessages.f90`. His algorithm solution is the well known Mehrotra's Predictor-Corrector algorithm implementation applied to quadratic programming. This algorithm is explained in chapter 16 of [11]

# Chapter 4

# Solution Tests

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 the PBS generator script on appendix (B.1). This script generator created PBS files which in turn run bash shell scripts[2]. One of them is available in appendix (B.2). 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. The original `L-BFGS-B` optimizer displays different messages depending on the condition that triggered the exit.

## 4.1 Exit Messages

The following is a list of some of the other most common exit messages in the original `L-BFGS-B` optimizer.

- "ABNORMAL_TERMINATION_IN_LNSRCH" This message means that there was a problem and the program's exit was premature. It is typically found in Non-Smooth functions where the cubic interpolation is impossible. But the message is also symptomatic of other problems.
- "CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_LT_PGTOL": Means that convergence was achieved because the norm of the projected gradient is small

[1] PBS stands for Portable Batch System. This is a 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. In this case, the program execution of the original `L-BFGS-B` software and `L-BFGS-B-NS`

enough. 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 instead as explained in section (3.3).

- "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 machine $\epsilon$. This exit message does not apply to `L-BFGS-B-NS` either. It was disabled by setting the factor "FACTR" to zero.

on top of these messages, `L-BFGS-B-NS` introduced a new message for a successful exit from its termination condition. The following message replaces "CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_LT_PGTOL"

- "CONVERGENCE: ZERO_GRAD_IN_CONV_HULL" What this means is that the termination condition established on section (3.3) was satisfied[3].

.

Also, the limit to the number of iterations was set to 10.000.

## 4.2 Modified Rosenbrock Function

Consider a modified version of the Rosenbrock function problem [13]:

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

We can study the properties of function $f$ based on the properties of $\phi(x)$. Considering $\phi(t_i) = |t_i|^p$ where $t_i = x_i - x_{i-1}^2$. In this case, 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 $p \geq 1$. In particular, it is locally Lipschitz continuous if restricted to a finite domain of the type $l$, $u$, similar to the one defined on equation (1.1). However, its second derivative blows up whenever $p < 2$.

[3] This does not mean that the resulting vector is exactly equal to zero 0, but it is small enough to satisfy the termination condition

[4] The original Rosenbrock function had a value of $p = 2$ and the second term is multiplied by 100.

The results can be split into cases; Whenever $p > 1$ the derivative can be represented as,

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

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

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

. The conclusion from the existence of this limit is that the function $f$ has a smooth first derivative.

If $p = 1$, $\phi(t) = |t|$. In this case, $\phi(t)$ is Lipschitz continuous at $t = 0$ because the derivative's value is bounded. Its values are either $-1$ or 1. This function is not differentiable at $t = 0$ though.

The second derivative provides a bit more of information.

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

In this case, if $p \geq 2$ the function is smooth. 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 $|t| \to 0$.

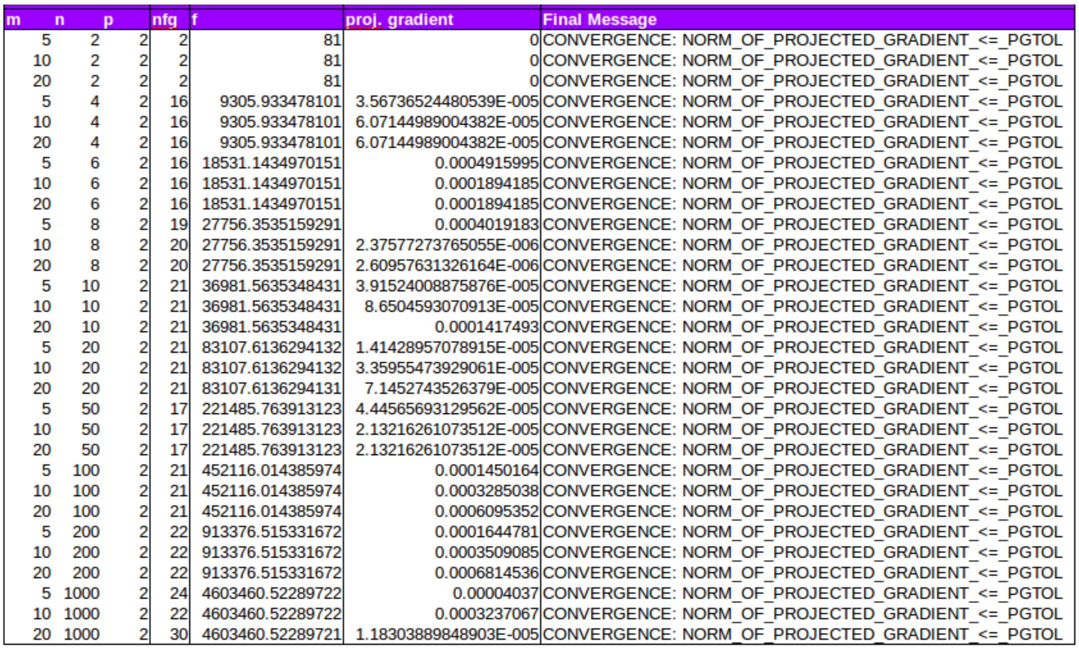
The next thing that needs to be defined is the region to be tested. In this case, the region was defined by the "box" with boundaries

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

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 stroke.

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

The problem is twice continuously differentiable for values of $p \geq 2$, but as the values of $p$ approach 1, the original `L-BFGS-B` optimizer should start to have problems. In order to check for that, we tested the original `L-BFGS-B` optimizer on the modified Rosenbrock function with $p$ varying between 2 and 1.



| m | n | p | nfg | f | proj. gradient | Final Message |
|---|---|---|---|---|---|---|
| 5 | 2 | 2 | 2 | 81 | 0 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 2 | 2 | 2 | 81 | 0 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 2 | 2 | 2 | 81 | 0 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 4 | 2 | 16 | 9305.933478101 | 3.56736524480539E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 4 | 2 | 16 | 9305.933478101 | 6.07144989004382E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 4 | 2 | 16 | 9305.933478101 | 6.07144989004382E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 6 | 2 | 16 | 18531.1434970151 | 0.0004915995 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 6 | 2 | 16 | 18531.1434970151 | 0.0001894185 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 6 | 2 | 16 | 18531.1434970151 | 0.0001894185 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 8 | 2 | 19 | 27756.3535159291 | 0.0004019183 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 8 | 2 | 20 | 27756.3535159291 | 2.37577273765055E-006 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 8 | 2 | 20 | 27756.3535159291 | 2.60957631326164E-006 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 10 | 2 | 21 | 36981.5635348431 | 3.91524008875876E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 10 | 2 | 21 | 36981.5635348431 | 8.6504593070913E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 10 | 2 | 21 | 36981.5635348431 | 0.0001417493 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 20 | 2 | 21 | 83107.6136294132 | 1.41428957078915E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 20 | 2 | 21 | 83107.6136294132 | 3.35955473929061E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 20 | 2 | 21 | 83107.6136294131 | 7.1452743526379E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 50 | 2 | 17 | 221485.763913123 | 4.44565693129562E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 50 | 2 | 17 | 221485.763913123 | 2.13216261073512E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 50 | 2 | 17 | 221485.763913123 | 2.13216261073512E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 100 | 2 | 21 | 452116.014385974 | 0.0001450164 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 100 | 2 | 21 | 452116.014385974 | 0.0003285038 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 100 | 2 | 21 | 452116.014385974 | 0.0006095352 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 200 | 2 | 22 | 913376.515331672 | 0.0001644781 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 200 | 2 | 22 | 913376.515331672 | 0.0003509085 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 200 | 2 | 22 | 913376.515331672 | 0.0006814536 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 1000 | 2 | 24 | 4603460.52289722 | 0.00004037 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 1000 | 2 | 22 | 4603460.52289722 | 0.0003237067 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 1000 | 2 | 30 | 4603460.52289721 | 1.18303889848903E-005 | CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |

Figure 4.1: Satisfactory results for the original algorithm L-BFGS-B applied to the Modified Rosenbrock function with $p = 2$

### 4.2.1 Performance of L-BFGS-B

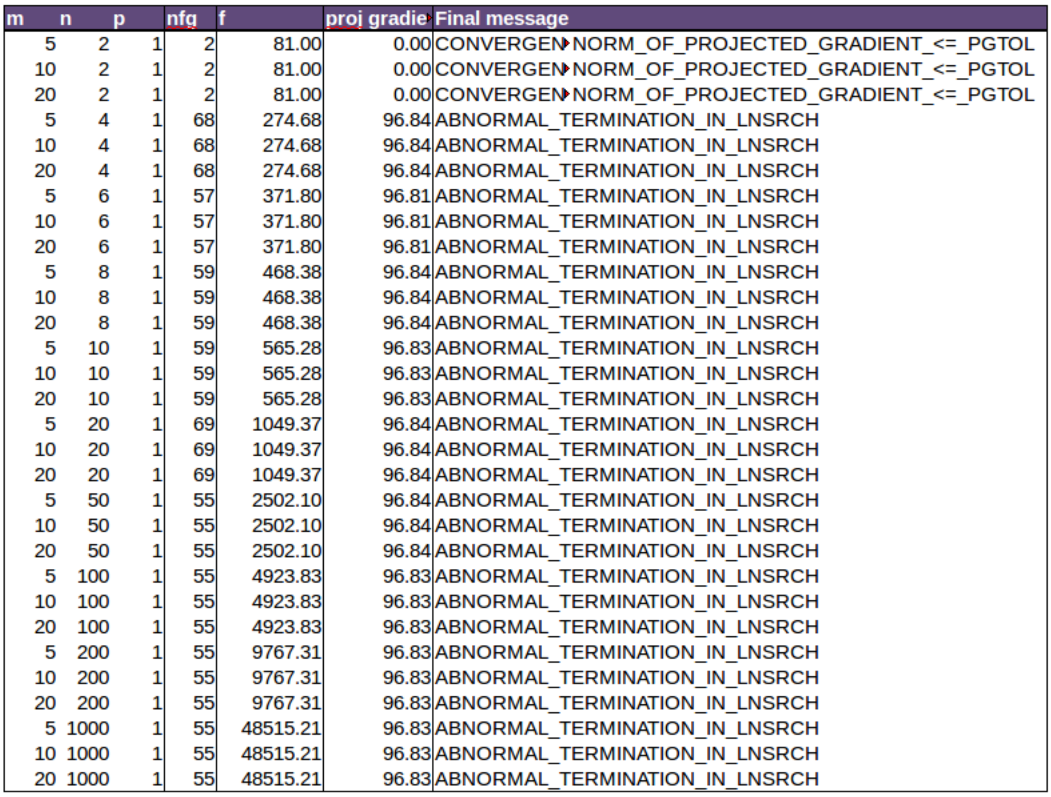
For a value of $p = 2$, the original L-BFGS-B yields good results as seen on the resulting table (4.1).

This exercise tested three different values of $m$, where $m$ stands for the memory of L-BFGS. The values that were tested are 5, 10 and 20. The number of dimensions in this exercise in particular ranges from 2 to 1000. The column $nfg$ stands for the number of function and gradient evaluations taken in order to finish the algorithm and $f$ stands for the optimal value that was achieved during the optimization. The two last columns show the norm of the final projected gradient and the final message when the algorithm finished.

In all cases the projected gradient has a very small norm. When this norm is sufficiently small the convergence is achieved because the norm of the projected gradient is smaller than the threshold. In other cases, the program exits successfully after a relative reduction in the size of the function. In other cases, an abnormal termination message is sent to the output.

The overall conclusion from this exercise is that the original L-BFGS-B optimizer works well, for the smooth modified Rosenbrock case[5].

[5] The original algorithm solves a modified version of this function (4.1) with parameter $p = 2$, so it is to be expected that the performance is good



| m | n | p | nfg | f | proj gradie | Final message |
|---|---|---|---|---|---|---|
| 5 | 2 | 1 | 2 | 81.00 | 0.00 | CONVERGEN▸NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 10 | 2 | 1 | 2 | 81.00 | 0.00 | CONVERGEN▸NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 20 | 2 | 1 | 2 | 81.00 | 0.00 | CONVERGEN▸NORM_OF_PROJECTED_GRADIENT_<=_PGTOL |
| 5 | 4 | 1 | 68 | 274.68 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 4 | 1 | 68 | 274.68 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 4 | 1 | 68 | 274.68 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 6 | 1 | 57 | 371.80 | 96.81 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 6 | 1 | 57 | 371.80 | 96.81 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 6 | 1 | 57 | 371.80 | 96.81 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 8 | 1 | 59 | 468.38 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 8 | 1 | 59 | 468.38 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 8 | 1 | 59 | 468.38 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 10 | 1 | 59 | 565.28 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 10 | 1 | 59 | 565.28 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 10 | 1 | 59 | 565.28 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 20 | 1 | 69 | 1049.37 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 20 | 1 | 69 | 1049.37 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 20 | 1 | 69 | 1049.37 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 50 | 1 | 55 | 2502.10 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 50 | 1 | 55 | 2502.10 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 50 | 1 | 55 | 2502.10 | 96.84 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 100 | 1 | 55 | 4923.83 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 100 | 1 | 55 | 4923.83 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 100 | 1 | 55 | 4923.83 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 200 | 1 | 55 | 9767.31 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 200 | 1 | 55 | 9767.31 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 200 | 1 | 55 | 9767.31 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 5 | 1000 | 1 | 55 | 48515.21 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 10 | 1000 | 1 | 55 | 48515.21 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |
| 20 | 1000 | 1 | 55 | 48515.21 | 96.83 | ABNORMAL_TERMINATION_IN_LNSRCH |

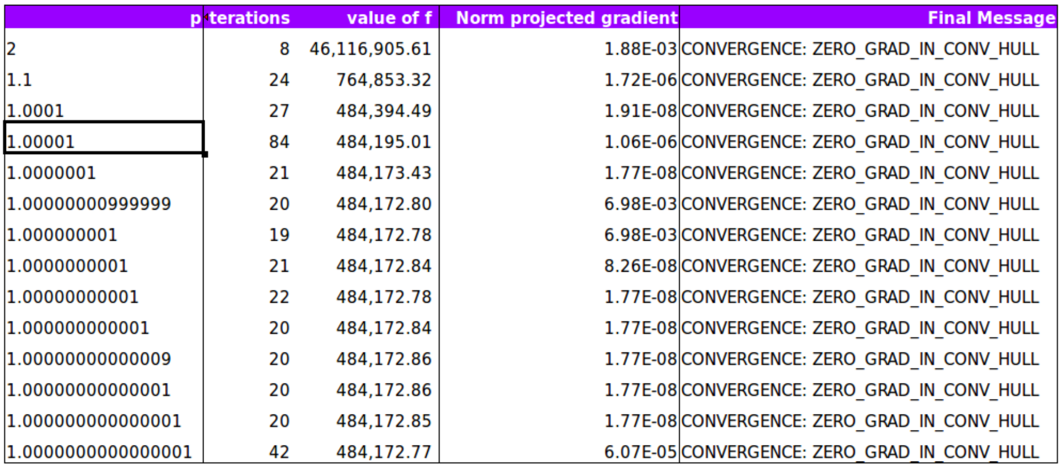
FIGURE 4.2: Unsatisfactory results for the original algorithm `L-BFGS-B` applied to the Modified Rosenbrock function with $p = 1$, notice however that the two-dimensional case is successful. This is because the function is smooth in this particular case.

On the other hand, the value of $p = 1$ has an abnormal line search termination in most of the cases presented. The projected gradient values are always far away from zero see table (4.2).

In this exercise, the memory length $m$ of `L-BFGS`, does not have an impact on the final value $f$ of the optimization, but this is because all cases crashed before the $5^{th}$ iteration and therefore all different cases of $m$ end up looking exactly the same in this table.

Several other values of $p$ were also tested, among others 1.5, 1.1, 1.01, 1.001, ... , 1.000000001, 1. As expected, those values where $p$ is closer to 1 are the most difficult to solve for the original algorithm. When $p = 1$ the algorithm does not work. See (4.2)[6]. It is important to point out that the two dimensional case is successful because in this particular case the function is smooth inside its bounding box.

[6] This is expected because the algorithm is originally designed to handle only smooth functions



| p | terations | value of f | Norm projected gradient | Final Message |
|---|---|---|---|---|
| 2 | 8 | 46,116,905.61 | 1.88E-03 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.1 | 24 | 764,853.32 | 1.72E-06 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.0001 | 27 | 484,394.49 | 1.91E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.00001 | 84 | 484,195.01 | 1.06E-06 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.0000001 | 21 | 484,173.43 | 1.77E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.00000000999999 | 20 | 484,172.80 | 6.98E-03 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.000000001 | 19 | 484,172.78 | 6.98E-03 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.0000000001 | 21 | 484,172.84 | 8.26E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.00000000001 | 22 | 484,172.78 | 1.77E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.000000000001 | 20 | 484,172.84 | 1.77E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.00000000000009 | 20 | 484,172.86 | 1.77E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.00000000000001 | 20 | 484,172.86 | 1.77E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.000000000000001 | 20 | 484,172.85 | 1.77E-08 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |
| 1.0000000000000001 | 42 | 484,172.77 | 6.07E-05 | CONVERGENCE: ZERO_GRAD_IN_CONV_HULL |

FIGURE 4.3: This is the number of algorithm iterations for different values of $p$. The value of the projected gradient is presented as well.

### 4.2.2 Performance of `L-BFGS-B-NS`

For intermediate values, the new changes seem to provide better values of $f$. Values generated via `L-BFGS-B-NS` are a little better for values of $p$ closer to 1, since the function is "less" smooth.

On table (4.3). It is possible to see that changing parameter $p$ all other things held constant, makes the problem more difficult to solve. Here the termination conditions is the one seen on (3.3). Under this condition and for the same values of $tau_d$ The number of iterations taken in order to finish seems to grow until a certain point.

# Chapter 5

# Conclusions

The need to build optimizers that work on a large number of variables quickly. Led us to the use of large scale quasi-Newton methodologies. These methodologies have been shown to work very well for smooth functions and several implementations have been suggested and made available already.

In the case of Non-Smooth functions, a lot of changes were proposed in this thesis. Implementing the changes proposed to the original `L-BFGS-B` software provides the capability to run optimizations on Non-Smooth functions on simply restricted domains. The most important changes were the Wolfe condition; which was changed from the strong to the weak version and using a line search algorithm that does not require smoothness of the function at critical points.

With the new tool, which is called `L-BFGS-B-NS`, it is possible to run optimizations of problems in large dimensions for some complicated tests. The software has been tested with very challenging functions and has performed well.

The new methodology offered some challenges to the way in which the algorithm terminates and for this reason it was also necessary to implement termination conditions that take care of the wedges natural to Non-Smooth functions. The new termination conditions work fine for the problems tested.

After running the algorithm it seems like there is not a good rule to choose the number of memory step terms $m$ to keep in memory, but this is something that also happened in the case of smooth functions. Also, the test functions Modified Rosenbrock and Modified NCR-NS1, show that the more “Non-Smooth” a function becomes, the more iterations and function evaluations will be required for the function to converge.

Another conclusion, quite obvious, is that larger problems involve a greater number of resources

Future steps suggest the investigation of limited-memory bundled methods $LMBM$, how it compares with `L-BFGS-B-NS` and how one could benefit from the other. Also, the study of the impact of the Lipschitz constant on the convergence of the `L-BFGS-B-NS` algorithm [16]. This line of research opened up as we were already finishing this thesis.

# Appendix A

# Samples of Code

## A.1 Old Parts of the Code (Commented Out)

All the code is available on [6] . The older version of the code was left untouched and simply was commented out. Here the old function dcsrch. lineww is replacing it instead

```

!     call dcsrch(f,gd,stp,ftol,gtol,xtol,zero,stpmx,csave,isave,dsave)
      call lineww(f,gd,stp,ftol,gtol,xtol,zero,stpmx,csave,isave,dsave)
```

## A.2 Stage 2 of the Line Search

Stage 2 of the line search[17] . This stage is not being run in lbfgsbNS. It is kept there for comparison. For the new line search that replaces it. You can look at appendix A.4

```
       if (stage .eq. 1 .and. f .le. fx .and. f .gt. ftest) then

!  Here we define the modified function and derivative values.
!  This line search will only aim to satisfy the condition in (3.3) modified Armijo
         fm = f - stp*gtest
         fxm = fx - stx*gtest
         fym = fy - sty*gtest
         gm = g - gtest
         gxm = gx - gtest
         gym = gy - gtest

!  Call dcstep to update stx, sty, and to compute the new step.
!
         call dcstep(stx,fxm,gxm,sty,fym,gym,stp,fm,gm,brackt,stmin,stmax)

!        Reset the function and derivative values for f

```

```
         fx = fxm + stx*gtest
         fy = fym + sty*gtest
         gx = gxm + gtest
         gy = gym + gtest

      else
```

## A.3 Cubic and Quadratic Line Search Sample

This part of the code does not work in the case when the function is Non-Smooth . For this reason it was eliminated from execution on lbfgsbNS and replaced with A.4. stx in this case is a variable that represents the step with the minimum value so far.

```
 !     First case: A higher function value. The minimum is bracketed.
!     If the cubic step is closer to stx than the quadratic step, the
!     cubic step is taken, otherwise the average of the cubic and
!     quadratic steps is taken.
  ! theta, three, gamma are parameters
      if (fp .gt. fx) then
         theta = three*(fx - fp)/(stp - stx) + dx + dp
         s = max(abs(theta),abs(dx),abs(dp))
         gamma = s*sqrt((theta/s)**2 - (dx/s)*(dp/s))
         if (stp .lt. stx) gamma = -gamma
         p = (gamma - dx) + theta
         q = ((gamma - dx) + gamma) + dp
         r = p/q
         stpc = stx + r*(stp - stx) !quadratic step
         stpq = stx + ((dx/((fx - fp)/(stp - stx) + dx))/two)* &
                                                            (stp - stx) !The
  cubic step
         if (abs(stpc-stx) .lt. abs(stpq-stx)) then
            stpf = stpc
         else
            stpf = stpc + (stpq - stpc)/two
         endif
         brackt = .true.
```

## A.4 Line Search Enforcing Weak Wolfe Conditions

This is the implementation of the line search that enforces the weak Wolfe conditions. Bisections and expansions (as long as it doesn't mean leaving the bounding box) of the step length. The purpose of this version is to be as similar as possible to the interior of the while loop in qpspecial.m at hanso [12]

```
       if (fp .ge. ftest) then
         sty = stp
         fy = fp
         dy = dp
         brackt = .true.
      else
!     if second condition is violated not gone far enough (Wolfe)
         if (-dp .ge. gtol*(-ginit)) then
            stx = stp
            fx = fp
            dx = dp
         else
            return
         endif
      endif

      !Bisection and expansion
      if (brackt) then
         stp = (sty + stx)/two
      else
         if (two * stp .le. stpmax) then !Remain within boundaries
            ! Still in expansion mode
            stp = two * stp
         else
            brackt = .true.
            sty = stp
            fy = fp
            dy = dp
         endif
      endif
      return
      end
```

# Appendix B

# Automation

This appendix includes some of the files that were used to run examples in parallel at the High Performance Clusters at NYU. All of these files can be found in the repository [6]

## B.1 Script Generator

This is a sample of the script generator that creates pbs files to be sent to the High Performance Machines. The value “i” creates a different set of files when it varies between 0 and 20, in this case one file for each value of the parameter $p$ from 4.1 that we want to test. To see one of the files resulting from running this script look at appendix B.2.

```
#!/bin/bash

for i in {0..20}
do
  cat > pbs.script.rosenbrock.$i <<EOF
#!/bin/bash

#PBS -l nodes=1:ppn=8,walltime=48:00:00
#PBS -m abe
#PBS -M weh227@nyu.edu
#PBS -N rosenbrockHD$((i))

module load gcc/4.7.3

cd /scratch/weh227/rosenbrock/
p=$(bc -l <<< "1+10 ^(-$((i)))")
for n in 1000 100000 1000000 10000000 100000000
do
   echo 10 \$n \$p
   ~/Documents/thesis/lbfgsfortran/./rosenbrockp 10 \$n \$p >> outputrosenbrock$((i)).txt
done

exit 0;

EOF
done
```

After they are created I could fire all of the runs at once by using the command

```
for i in 0..20; do qsub pbs.script.rosenbrock.$i ; done
```

## B.2 PBS File Sample

This is one of the sample files that were generated via the generator on appendix B.1. It runs a special value $p$. These are the ones that get started with the the qsub command.

Lines 4 to 5 provide the user with emails and critical information while running the process. Line 13 defines a run for different sizes of $n$. Line 16 is the actual run that will be sent to an output file. In this case “outputrosenbrock9.txt”

```
1
2  #!/bin/bash
3
4  #PBS -l nodes=1:ppn=8,walltime=48:00:00
5  #PBS -m abe
6  #PBS -M weh227@nyu.edu
7  #PBS -N rosenbrockHD9
8
9  module load gcc/4.7.3
10
11 cd /scratch/weh227/rosenbrock/
12 p=1.000000000100000000000
13 for n in 1000 100000 1000000 10000000 100000000
14 do
15    echo 10 $n $p
16    ~/Documents/thesis/lbfgsfortran/./rosenbrockp 10 $n $p >> outputrosenbrock9.txt
17 done
18
19 exit 0;
```

# Bibliography

[1] 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.

[2] 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.

[3] 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.

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

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

[6] Wilmer Henao. lbfgsbns. https://github.com/wilmerhenao/lbfgsbNS, 2014.

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

[8] 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.

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

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

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

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

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

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

[15] 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.

[16] J. L. Steward, I. M. Navon, M. Zupanski, and N. Karmitsa. Impact of non-smooth observation operators on variational and sequential data assimilation for a limited-area shallow-water equation model. *Quarterly Journal of the Royal Meteorological Society*, 138(663):323–339, 2012.

[17] Ciyou Zhu, Richard Byrd, Jorge Nocedal, and Jose Luis Morales. Lbfgsb 3.0. http://www.ece.northwestern.edu/~nocedal/lbfgsb.html, 2011.

[18] 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.