

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, Ciyu 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 tests and their helpful assistance.

Contents

Declaration of Authorship	i
Abstract	ii
Acknowledgements	iii
Contents	iv
List of Figures	vi
1 Introduction	1
2 L-BFGS-B	3
2.1 BFGS	3
2.2 L-BFGS	4
2.3 L-BFGS-B	4
2.3.1 Gradient Projection	4
2.3.2 Subspace Minimization	5
3 Modifications to the L-BFGS-B algorithm	7
3.1 The Armijo and Wolfe conditions	7
3.2 The line search Methodology	8
3.3 The Termination Condition	10
3.3.1 Termination Condition Sub-algorithm	10
3.3.2 The Solution of the Quadratic Program	11
4 Solution Tests	14
4.1 Exit Messages	14
4.2 Modified Rosenbrock Function	15
4.2.1 Performance of L-BFGS-B	16
4.2.2 Performance of L-BFGS-B-NS	18
4.3 Nesterov's Chebyshev-Rosenbrock Modified function	19
5 Conclusions	21

A	Samples of Code	23
A.1	Old Parts of the Code (Commented Out)	23
A.2	Stage 2 of the Line Search	23
A.3	Cubic and Quadratic Line Search Sample	24
A.4	Line Search Enforcing Weak Wolfe Conditions	24
B	Automation	26
B.1	Script Generator	26
B.2	PBS File Sample	27
	Bibliography	28

List of Figures

2.1	Graphical Representation of Gradient Projection	5
3.1	Representation of the Armijo Condition in a Nutshell	7
3.2	The Idea behind the Wolfe Condition	8
3.3	The Central Path and the Tangent Path	12
4.1	Modified Rosenbrock with $p = 2$	16
4.2	Original Rosenbrock Function Mesh	17
4.3	Contour Plot of the Original Rosenbrock Function	17
4.4	Modified Rosenbrock with $p = 1$	17
4.5	Comparison of selected values of the Modified Rosenbrock function	18
4.6	Number of function iterations for different values of p and m in the solution of Modified Rosenbrock	18
4.7	Number of function evaluations for different values of p and m in the solution of Modified Rosenbrock	19
4.8	Comparison of selected values of the Modified NCR-NS1 function pivoting by p and n parameters	19
4.9	Comparison of selected values of the Modified NCR-NS1 function pivoting by m and n parameters	20

Dedicated to my mother, my brother and Dr. Ian Malcolm

Chapter 1

Introduction

The problem addressed is to find a local minimizer of the Non-Smooth minimization problem

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

where $f: \mathbb{R}^n \rightarrow \mathbb{R}$, is continuous but not differentiable everywhere and n is a very large but finite number.

The L-BFGS-B algorithm [1] is a standard method for solving large instances of 1.1 when f is a smooth function. 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 [2–5]. 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 [6] where the L stands for “Limited-Memory” and also for “Large” problems, it is based on BFGS but requires only $\mathcal{O}(mn)$ operations per iteration. And not only it requires less FLOPS, but it also requires much 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 [7].

In this thesis, there is a brief description of the L-BFGS-B algorithm at a high level and there is an explanation of how the modified algorithm 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 [8] 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 at a very high level [7]. The original software is intended to work well with smooth functions. This thesis discusses how to modify the algorithm for Non-Smooth functions. The original FORTRAN code contains some documentation [9] about how the software was built.

2.1 BFGS

BFGS is a standard tool for optimization of smooth functions[10]. It is a line search method and its goal is to find a search direction starting from its current position x . The search direction is of type $d = -B_k \nabla f$ ¹ where B_k is the k^{th} approximation to the inverse Hessian.² This k^{th} step approximation is calculated via the BFGS formula

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

where $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$ and where the first approximation of B_0 is assumed to be the identity matrix I in this thesis³. BFGS exhibits super-linear convergence but it also requires $\mathcal{O}(n^2)$ operations per iteration[10].

In the case of Non-Smooth functions. BFGS typically succeeds in finding a local minimizer. However, this requires some modifications of the line search conditions. These conditions are known as the Armijo and weak Wolfe line search conditions and they are

¹Notice that when B is the identity, this is the same direction as steepest descent. Another common line search method of optimization

²When 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.

³A different starting matrix is suggested in Nocedal's book [10]

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

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 B . 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 according to the problem at hand⁴. For this reason the first m iterations of BFGS and L-BFGS produce exactly the same search directions.

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 comes naturally as 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 at bound 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 or *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 [11, 12] which is used

⁴In 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

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:

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

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. At least a number \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:

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

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 on the current point x_k in the direction $-\nabla f(x_k)$. Whenever this direction encounters one of the constraints, the line segment 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 \vec{l} and \vec{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 idea at a higher level is to solve the constrained problem 2.3, but only on those dimensions that are free. The starting point for this new stage will be the previously found Cauchy point x^c , the L-BFGS algorithm provides a new search direction \hat{d}^u that takes implicit advantage of second order approximations of the Hessian matrix.

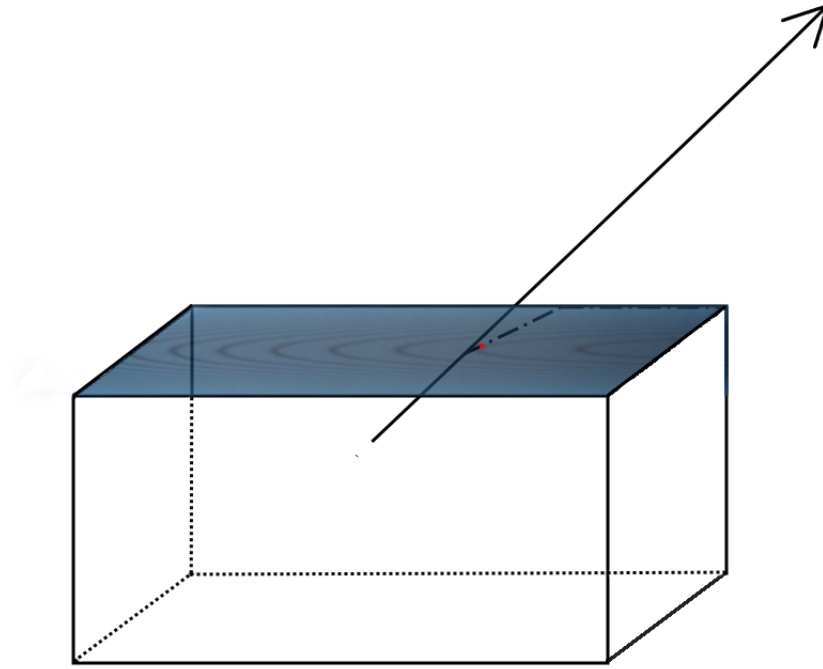


FIGURE 2.1: The arrow represents the direction of the gradient. The dotted path represents the projected gradient path from the gradient and onto the box. The region represents the level sets of the model. The optimal point (in red) is the Cauchy point x^c

The algorithm will move in the direction $\hat{d}^* = \alpha^* \hat{d}^u$ where α^* (the step length) is chosen so that the new point \bar{x}_i satisfies Armijo and weak Wolfe descent⁵ and curvature conditions. 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. If the termination condition is successful, the program exits with an appropriate exit message.

⁵Armijo 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

Probably the most important change made to the original code was the change in the curvature condition. It is accepted that the Armijo and Wolfe conditions work very well whenever the function f is smooth [13]. The Armijo condition, also known as the sufficient decrease requirement in the direction d_k is defined as

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k d_k^T \nabla f(x_k) \quad (3.1)$$

where $0 < c_1 < 1$ is a constant usually $c_1 = 10^{-4}$ [10]. This condition guarantees that the function continues decreasing and eventually reaches some optimum. 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 and the one that was actually changed, is the curvature condition, of which the most popular version is the “strong Wolfe” curvature condition:

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

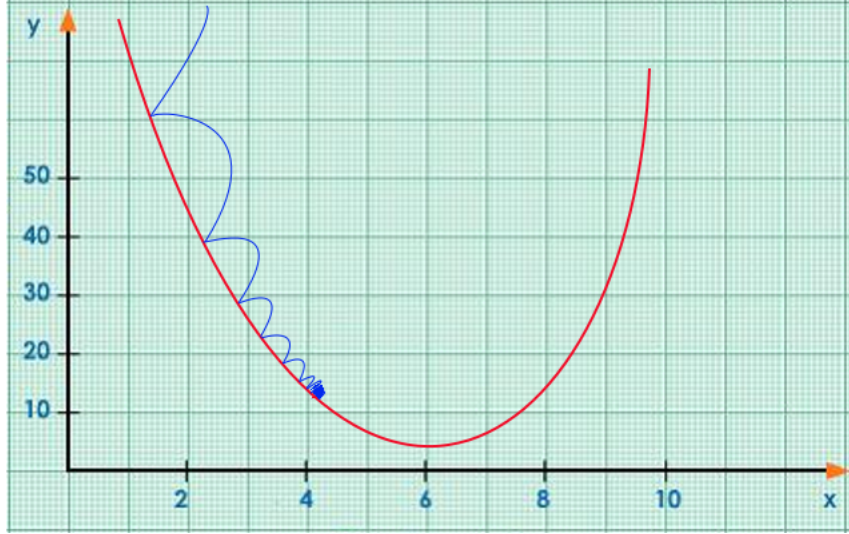


FIGURE 3.1: In this figure, the iterations always reduce the value of the function a little bit, but never enough to go below 12

Where d_k represents the search direction and c_2 is a constant such that $0 < c_1 < c_2 < 1$ and is usually $c_2 = 0.9$ [10]. The strong Wolfe condition is a natural choice for optimization 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

$$d_k^T \nabla f(x_k + \alpha_k d_k) \geq c_2 d_k^T \nabla f(x_k) \quad (3.3)$$

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 [14]. 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[7] contains a line search subroutine. This subroutine originally consisted of three steps. A maximum length, a sufficient decrease and a curvature condition (the Armijo and Wolfe conditions from the previous section). This

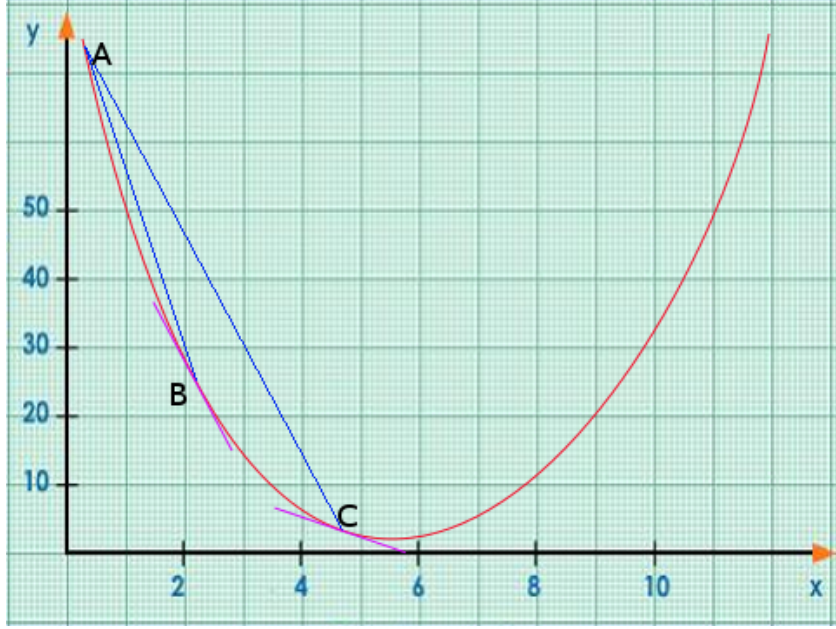


FIGURE 3.2: The logic of Wolfe conditions is this. Starting at point A. Point B is a step in the right direction, however, point C offers a “flatter” tangent and should be closer to the optimum which has a tangent of zero (Smooth case)

sub-procedure was partially changed for the purpose of this thesis. The old version of the code was not removed, it was left commented out and is only explained here with the purpose of showing why it needed to be modified. It was called on line 2662 [A.1](#) of `lbfgsbnomessages.f90` which is a file within the github repository[8].

1st The line search requires a maximum step. This maximum step is such that it guarantees that the step length stays within the bounding box delimited by l and u , as such, this is a unique feature of the bounded algorithm `L-BFGS-B`. In the original FORTRAN code, this part of the line search was not changed.

2nd An interval for α_k is chosen so that it contains a minimizer of the modified function,

$$\Psi(\alpha_k) = f(x_k + \alpha_k d_k) - f(x_k) - c_1 \alpha_k d_k^T \nabla f(x_k) \quad (3.4)$$

which is nothing but a modification of the Armijo condition [3.1](#). If $\Psi(\alpha_k) < 0$ and $\alpha_k d_k^T \nabla f(x_k) > 0$. The interval contains a minimizer of f and the subroutine can continue to the next step. This first stage took place in the subroutine `dcsrch`; specifically on lines 3687 and 3709 on [appendix A.2](#).

3rd This stage is very similar to the typical line search[10]. It was called on line 3713 of `lbfgsbnomessages.f90` and its mission was to find the step that satisfied Armijo and “Strong” Wolfe conditions on the original function f . Both steps 2

and step 3 are called on the subroutine *dcstep*. The subroutine still exists on file `lbfgsbnomessages.f90` for illustration. Although it is only commented out and never called in this thesis.

Those three steps make up the original subroutine. But there is a problem with the function *dcstep* on the third step 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 `hanso`[15]. The HANSO approach is to double the step length while the Armijo condition is violated. And once the interval has been bracketed, to do a bisection until both Armijo and Wolfe conditions are satisfied. The only difference with the HANSO approach in this thesis is that the line search in HANSO can double its step length up to 30 times¹. Whereas in this thesis, the step length can double only as long as the step length 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

One important requirement of an algorithm is that it ends in a finite time. In the case of smooth functions, L-BFGS-B checks whether the algorithm has converged, by means of the *projected gradient*² 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 norm³ the algorithm has converged. In the case of Non-Smooth functions however, this is not necessarily true and 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

¹for all practical purposes this is considered a good limit of iterations

²Not to be confused with “gradient projection” from subsection 2.3.1

³smaller than some tiny $\epsilon > 0$

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 [14, 16].

3.3.1 Termination Condition Sub-algorithm

Overton and Lewis formulate an algorithm that gives a practical solution to this problem on section 6.3 [14]. The best practical methodology should be to calculate whether zero 0 or a very small number is the norm of a vector that is part of the convex hull of gradients evaluated at points near the optimum candidate x . The neighborhood is defined as those points 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. In order to make sure that the gradient zero $\vec{0}$ or a vector with a very small norm smaller than a very tiny tolerance $\tau_d > 0$ is part of the convex hull calculated near a neighbourhood of the optimum, the algorithm keeps a record of the latest gradient vectors in this small neighbourhood of the point where the optimum is suspected to be located. This list of gradients is referred to as the set \mathcal{G} [14].

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 norm of this vector has a norm smaller than τ_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 following 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 non-negative linear 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 $\sum_{i=1}^n z_i = 1$ 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

$$\begin{aligned} \min \quad & q(z) = \|Gz\|_2^2 = z^T G^T G z \\ \text{s.t.} \quad & \sum_{i=1}^J z_i = 1 \\ & z_i \geq 0. \end{aligned} \tag{3.5}$$

The solution to this problem z^* has the associated vector Gz^* . And if $\|Gz^*\|_2 < \tau_d$ the algorithm converges.

3.3.2 The Solution of the Quadratic Program

The solution of equation 3.5 was implemented with a practical primal-dual methodology. This methodology is the same methodology implemented by Skajaa[16] 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 initially tries to solve the Karush-Kuhn-Tucker *KKT* conditions. The *KKT* is a system of equations whose solution characterizes the solution of the original optimization problem in equation 3.5, the Karush-Kuhn-Tucker equations are:

$$\begin{aligned} G^T Gz - e^T s &= 0 \\ \sum_{i=1}^J x - 1 &= 0 \\ z_i s_i &= 0; \quad i = 1, 2, \dots, J \\ (s, z) &\geq 0 \end{aligned} \tag{3.6}$$

Where s is the variable that represents the dual associated problem.

In order to solve this system of equations a variant of Newton's method is used. The method provides a search direction and as usual for this type of methods, it requires a corresponding step length. Since this is an interior point method, it approaches the solution following a path inside the convex interior of the feasible set. In the best of cases, it would be nice to approach the solution through a *central path* defined as third equation in 3.6 is allowed to take more relaxed values $z_i s_i = \tau$, where $\tau \geq 0$. If the solution is approached through this *central path*. The convergence will require fewer iterations⁴. The logic is that by being central, it is kept away from the borders.

The problem with a pure Newton's method is that it's solution is not close to this *central path*. The step length is usually very small because of the nature of the third and fourth equations. For this reason, if the step length is too large, either the condition of $z > 0$ or $s > 0$ will be violated. Unfortunately the pure method this way, does not allow reasonable step sizes in practice. The method approaches the solution very close to the

⁴The reason why the central path is so convenient is more thoroughly explained in Nocedal's text chapter 14[10]

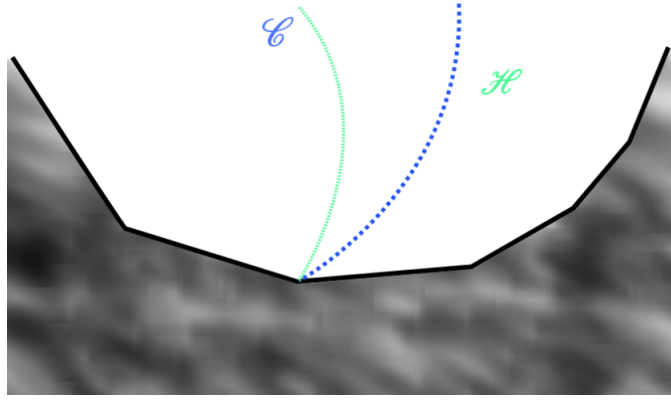


FIGURE 3.3: The central path \mathcal{C} is approached from a current and non-central point on the tangent path \mathcal{H}

boundaries of the feasible set and far away from the *central path* where the step lengths could be larger and the convergence therefore would be faster. It is for this reason that a *centering* of the path is performed. This first step is known as the “predictor” step.

Besides the centering; Mehrotra proposed a “correction” stage which pushes the solution even closer to the *central path* by taking into account its curvature. This second stage is also used to calibrate some of the parameters used during the centering in the “predictor” stage⁵. The second stage is similar to the first stage in that it requires the solution of a new system of equations.

And this is very important, because both steps require a solution of a system of equations which in turn requires a matrix factorization. And this factorization can be recycled. Mehrotra’s algorithm requires only one Cholesky factorization for the first stage. This factorization is reused during the second stage. And reusing the factorization, the performance is improved.

⁵These are some parameters that calibrate the speed at which the tangent path is pushed into the central path. This is also explained in chapter 14[10]

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 files¹ using the PBS generator script on appendix B.1. This script generator created PBS files which in turn run bash shell scripts². One of them is available on 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 initial 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

¹PBS 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

²Bash 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 ϵ . This exit message does not apply to L-BFGS-B-NS either.
- “ERROR: N.LE. 0”: This is one of many error checks performed at the beginning of the execution. This one in particular tests positivity of the dimensionality of the problem.
- “ERROR: NO FEASIBLE SOLUTION”: An error condition for those cases when the solution is not feasible. This is because the original L-BFGS-B optimizer allows open boxes of type $[l, \infty[,]-\infty, u]$ and $] -\infty, \infty[$
- “WARNING: ROUNDING ERRORS PREVENT PROGRESS”: If there is a rounding error issue

on top of these messages, L-BFGS-B-NS introduced a new message for a successful exit from its termination condition.

- “CONVERGENCE: ZERO_GRAD_IN_CONV_HULL” What this means is that all termination conditions established on section 3.3 were satisfied³.

4.2 Modified Rosenbrock Function

With those tools in hand, it is possible to run a few tests of the algorithm. Several functions were evaluated. One of them is a modified version of the Rosenbrock function problem[17]:

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

Here, the properties of the function depend on the value of the p parameter⁴. This function can be proven to be locally lipschitz continuous whenever $p \geq 1$. In particular

³This does not mean that the resulting vector is exactly equal to zero 0, but it is small enough to finish the execution of the algorithm

⁴The original Rosenbrock function would have a value of $p = 2$ and the second term is multiplied by 100

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.

The region to be tested is defined by the “box” with boundaries

$$x_i = \begin{cases} [-100, 100] & \text{if } i \in \text{even numbers} \\ [10, 100] & \text{if } i \in \text{odd numbers} \end{cases} \quad (4.2)$$

And 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.

$$x_i = \frac{u_i + l_i}{2} - (1 - 2^{1-i}) \quad (4.3)$$

The problem is smooth for values of p close to 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 plug the Modified Rosenbrock function into the original L-BFGS-B optimizer with p parameter varying between 2 and 1.

For a value of $p = 2$, the original L-BFGS-B optimizer does not have too many problems and it yields good results as seen on the resulting table 4.1.

4.2.1 Performance of L-BFGS-B

This exercise tested three different values of m , where m stands for the steps in memory of L-BFGS. The steps that were tested are 5, 10 and 20, these values of m cover more or less the set of recommended values. The number of dimensions in this exercise in particular ranges from 2 to 1000 and covering many values in between. The column nfg stands for the number of iterations taken in order to finish the algorithm and f stands for the optimal value that was achieved during the optimization. The two last columns stand for the norm of the 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 tiniest 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.

m	n	p	nfg	f	proj gradient	Final message
5	2	2	2	81.00		0 CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<= PGTOL
10	2	2	2	81.00		0 CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<= PGTOL
20	2	2	2	81.00		0 CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<= PGTOL
5	4	2	56	9305.93	3.56737E-005	ABNORMAL_TERMINATION_IN_LNSRCH
10	4	2	17	9305.93	4.33747E-007	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	4	2	17	9305.93	4.33747E-007	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	6	2	18	18531.14	0.000000058	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	6	2	17	18531.14	3.57280E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	6	2	17	18531.14	3.57280E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	8	2	22	27756.35	2.14264E-008	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	8	2	21	27756.35	2.07683E-008	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	8	2	28	27756.35	2.60729E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	10	2	22	36981.56	5.36315E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	10	2	22	36981.56	1.70055E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	10	2	23	36981.56	3.42189E-008	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	20	2	26	83107.61	7.82706E-008	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	20	2	43	83107.61	3.26243E-005	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	20	2	22	83107.61	3.80408E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	50	2	18	221485.76	7.93769E-007	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	50	2	63	221485.76	2.12629E-005	ABNORMAL_TERMINATION_IN_LNSRCH
20	50	2	63	221485.76	2.12629E-005	ABNORMAL_TERMINATION_IN_LNSRCH
5	100	2	22	452116.01	2.51616E-005	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	100	2	38	452116.01	4.04381E-006	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	100	2	22	452116.01	1.96281E-005	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	200	2	63	913376.52	2.85871E-005	ABNORMAL_TERMINATION_IN_LNSRCH
10	200	2	32	913376.52	0.0003494976	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	200	2	54	913376.52	0.0006327763	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
5	1000	2	48	4603460.52	3.01364E-005	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
10	1000	2	36	4603460.52	0.0003236833	CONVERGENCE: REL_REDUCTION_OF_F_<= FACTR*EPSMCH
20	1000	2	70	4603460.52	1.18304E-005	ABNORMAL_TERMINATION_IN_LNSRCH

* Sometimes the message output reads "ABNORMAL_TERMINATION_IN_LNSRCH". However, the function converges to the right value

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

Notice that in 4.1 the program sometimes sends an "ABNORMAL_TERMINATION_IN_LNSRCH" warning message even though the function has converged to a value that is very close to the optimal expected value. This is because the original Rosenbrock function is designed to be very difficult to solve. Its minimum is located in the middle of a banana shaped valley. This valley is very flat and is designed to be a test problem that causes trouble to even the best optimizers as. The shape of the original Rosenbrock function can be seen in figure 4.2. The shape of this region is perhaps easier to see on the contour plot 4.3. The minimum is located at the point (1, 1) on one of the corners.

The overall conclusion from this exercise is that the original L-BFGS-B optimizer works well, for the smooth modified Rosenbrock case⁵.

⁵The 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

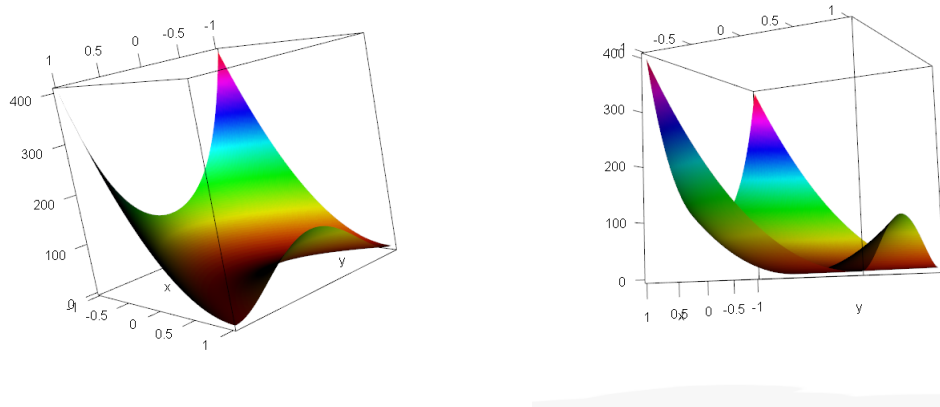


FIGURE 4.2: The original Rosenbrock function $f(x,y) = (1-x)^2 + 100(y-x^2)^2$ is designed to be a very difficult function to optimize. The global minimum is located along the very narrow, flat banana shaped valley[17]

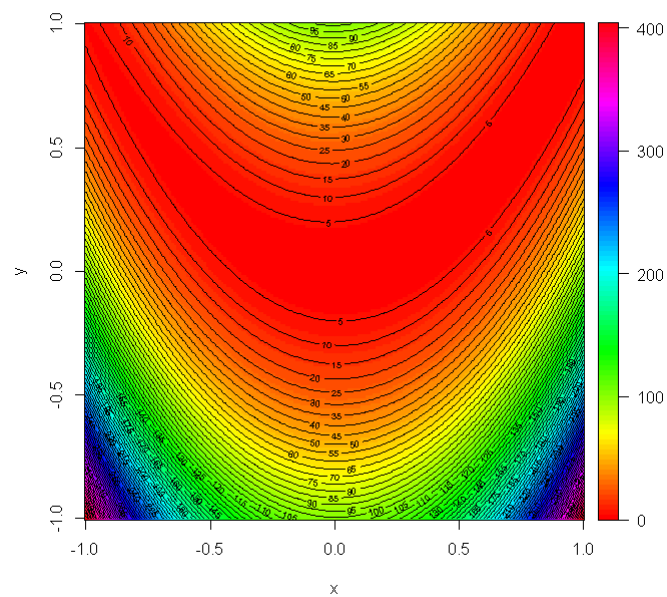


FIGURE 4.3: The contour plot shows the valley in red where the global optimum is located. It is located at the corner on (1, 1)

m	n	p	nfg	f	proj gradie	Final message
5	2	1	2	81.00	0.00	CONVERGENCE_NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
10	2	1	2	81.00	0.00	CONVERGENCE_NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
20	2	1	2	81.00	0.00	CONVERGENCE_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 4.4: 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 4.4.

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 5th iteration and therefore all different cases of m end up looking exactly the same in this table.

But an analysis of only the values $p = 1$ and $p = 2$ would not be enough. And therefore 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 as seen in 4.4⁶. 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.

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

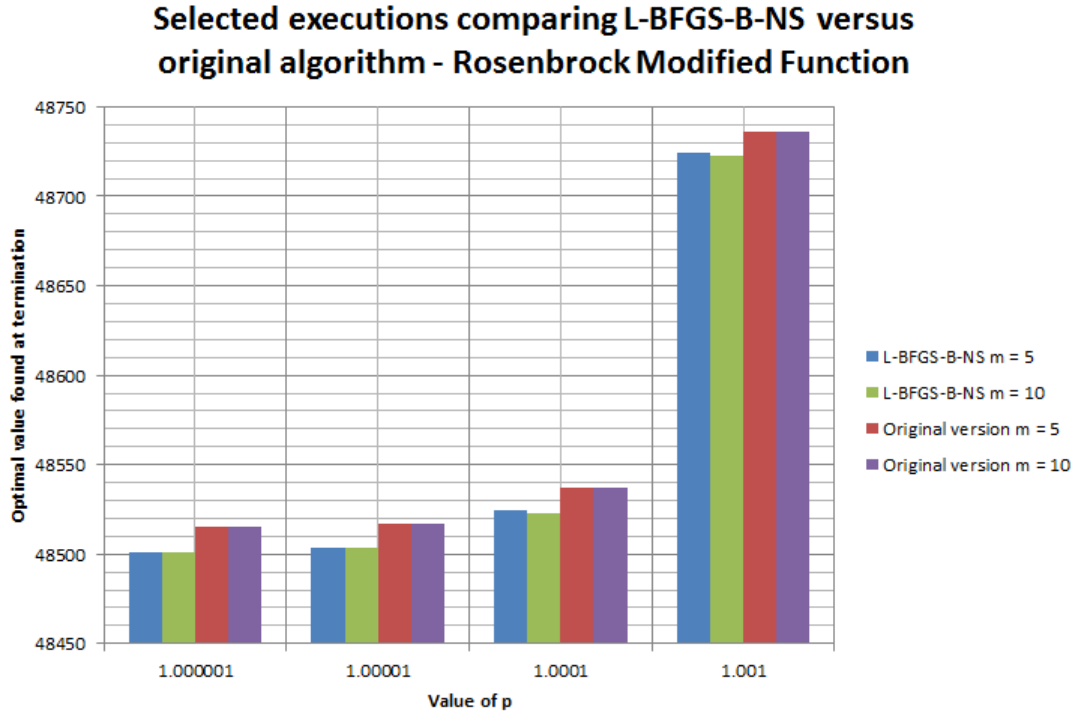


FIGURE 4.5: Comparison of the performance of L-BFGS-B-NS vs. the Original version of L-BFGS-B applied to the Modified Rosenbrock function with selected values of p in 1.000 dimensions

4.2.2 Performance of L-BFGS-B-NS

For intermediate values, the new changes seem to provide better values of f . Here is a simple comparison of the algorithm running on $m = 5, 10$ for selected values of p [4.5](#)

These are typical values of optimizations for intermediate values of p . In the bar graph it is possible to see that values generated via L-BFGS-B-NS are a little better for values of p closer to 1.

The test performed is represented in a table with the number of iterations performed sliced by p and m [4.6](#). And just as it would be expected, the number of evaluations required is higher for values of p close to 1.

The same table is reported for the number of function evaluations (as opposed to number of iterations) required to achieve convergence on table [4.7](#)

The values on these tables should be compared with the much more difficult problem in the next section.

		Values of m (L-BFGS lags)																		
Values of P	Iterations	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20			
	1.0000000000000000	19	19	23	34	23	42	23	21	21	21	21	21	21	21	21	21	21		
	1.0000000000000100	19	19	20	21	20	20	23	20	20	20	20	20	20	20	20	20	20		
	1.0000000000000900	19	19	20	21	20	20	23	20	20	20	20	20	20	20	20	20	20		
	1.0000000000001000	19	18	20	21	20	20	23	20	20	20	20	20	20	20	20	20	20		
	1.0000000000100000	18	19	20	20	20	22	21	20	20	20	20	20	20	20	20	20	20		
	1.0000000001000000	18	19	22	24	20	21	21	23	23	23	23	23	23	23	23	23	23		
	1.0000000010000000	32	25	27	23	20	19	20	20	20	20	20	20	20	20	20	20	20		
	1.0000000099999900	39	23	24	25	21	20	20	20	21	21	21	21	21	21	21	21	21		
	1.0000001000000000	25	27	5253	24	21	21	21	22	22	22	22	22	22	22	22	22	22		
	1.0000009999999900	24	343	3919				2678	945	945	945	945	945	945	945	945	945	945		
	1.0000100000000000				29	24	84	28	29	29	27	59	32	32	32	32	32	32		
	1.0001000000000000			26	25	27	27	25	24	27	24	24	28	28	28	28	28	28		
	1.1000000000000000	26	31	27	23	37	24	24	24	24	24	24	24	24	24	24	24	24		
	2.0000000000000000	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8		

FIGURE 4.6: This is the number of iterations for different values of p and m . Blank values converged using other stopping criteria besides the termination condition on 3.3

		Values of m (L-BFGS lags)																		
Values of P	Function Evaluations	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20			
	1.0000000000000000	148	147	106	174	101	267	118	101	101	101	101	101	101	101	101	101			
	1.0000000000000100	136	135	96	96	96	98	118	99	99	99	99	99	99	99	99	99			
	1.0000000000000900	136	135	96	96	96	98	118	99	99	99	99	99	99	99	99	99			
	1.0000000000001000	136	140	96	96	96	98	116	99	99	99	99	99	99	99	99	99			
	1.0000000000100000	141	135	96	96	96	96	96	97	97	97	97	97	97	97	97	97			
	1.0000000001000000	138	141	96	145	98	118	115	124	124	124	124	124	124	124	124	124			
	1.0000000010000000	144	229	124	125	152	111	152	152	152	152	152	152	152	152	152	152			
	1.0000000099999900	185	215	118	118	175	139	178	178	177	177	177	177	177	177	177	177			
	1.0000001000000000	138	276	10523	126	106	106	106	126	126	126	126	126	126	126	126	126			
	1.0000009999999900	106	738	7903				5400	1934	1934	1934	1934	1934	1934	1934	1934	1934			
	1.0000100000000000				140	143	241	105	128	130	115	181	111	111	111	111	111			
	1.0001000000000000			110	121	122	122	121	122	129	108	108	141	141	141	141	141			
	1.1000000000000000	67	76	71	61	92	62	62	62	62	62	62	62	62	62	62	62			
	2.0000000000000000	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16			

FIGURE 4.7: This is the number of function evaluations for different values of p and m . The number of function evaluations is a few times higher than the number of iterations because each iteration requires to evaluate the function a couple of times for the line search. Blank values converged using other stopping criteria besides the termination condition on 3.3

4.3 Nesterov's Chebyshev-Rosenbrock Modified function

The original NCR-NS1 function was first introduced in the thesis by Kaku[18]. It is a Non-Smooth variation of the NCR function. In this thesis there is yet a newer modification to this function and it is defined very similar to the function 4.1. with the inclusion of a p parameter.

$$f(x) = \frac{(x_1 - 1)^2}{4} + \sum_{i=1}^{n-1} |x_{i+1} - 2x_i^2 + 1|^p \quad (4.4)$$

And just like in equation 4.1 p is also assigned as a control variable for the Lipschitz Continuity of the function. The problem as stated in Kaku's thesis, is that the manifold defined by the term inside the sum,

Average number of Iterations given p and n (yuriosen function)												
n	p	1	1.000000001	1.00000001	1.0000001	1.000001	1.00001	1.0001	1.001	1.01	1.1	2
		2	4	6	8	10	20	50	2	4	6	8
2		2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	1.00
4		19.33	19.33	26.33	24.00	29.67	40.00	32.67	31.67	33.00	38.33	17.00
6		812.67	830.67	965.00	2480.67	2884.67	977.00	14.33	68.33	653.67	52.33	17.67
8		60.00	63.67	60.00	79.00	216.67	1072.00	97.33	74.33	121.33	67.33	22.67
10		80.00	107.33	79.33	61.67	51.33	242.67	108.33	111.00	90.67	83.33	24.00
20		129.00	130.33	127.00	124.33	124.67	125.00	116.67	113.67	158.67	137.67	18.00
50		156.00	185.67	199.33	189.33	201.33	181.00	197.00	152.33	151.00	195.00	19.67

*Average of m = 5; 10; 20

FIGURE 4.8: Comparison of the number of iterations in order to converge for the Modified NCR-NS1 functions sliced by the number of dimensions n and the p parameter that controls the Smoothness of the function

Average number of Iterations given m and n (yuriosen function)			
n	m	5	10
		20	50
2		1.92	1.92
4		33.67	25.42
6		502.50	1234.08
8		64.75	94.67
10		75.58	120.25
20		110.50	109.42
50		215.00	139.75

* For values of p = 2; 1.1; 1.01; 1.001; 1.0001; 1.00001; 1.000001; 1

FIGURE 4.9: Comparison of the number of iterations in order to converge for the Modified NCR-NS1 functions sliced by the number of dimensions n and the memory lag m

$$M = \{x : x_{i+1} - 2x_i^2 + 1 = 0, i = 1, \dots, n-1\} \quad (4.5)$$

is highly oscillatory in nature, and therefore BFGS type of methods require many iterations to converge. In particular, table 4.8 tries to slice the number of iterations required to finish the algorithm given different combinations of p and n , the number of dimensions. See table 4.8.

The results show that the number of iterations increases significantly with smaller values of p and with larger values of the dimension n . Also, the number of iterations is a lot larger than for the previous case on section 4.2.

This table was also pivoted with parameters n and m in table 4.9. And again you can see that the complexity of the problems grows with variable n , but there is no clear trend on the influence of m in the speed of convergence.

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[\[19\]](#). 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 [\[8\]](#) . The older version of the code was left untouched and simply was commented out. Here the old function `dcsrch`. `lineww` is replacing it instead

```
2607
2608 !      call dcsrch(f,gd,stp,ftol,gtol,xtol,zero,stpmx,csave,isave,dsave)
2609      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[\[7\]](#) . 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](#)

```
3687      if (stage .eq. 1 .and. f .le. fx .and. f .gt. ftest) then
3688
3689 ! Here we define the modified function and derivative values.
3690 ! This line search will only aim to satisfy the condition in (3.3) modified
      Armijo
3691      fm = f - stp*gtest
3692      fxm = fx - stx*gtest
3693      fym = fy - sty*gtest
3694      gm = g - gtest
3695      gxm = gx - gtest
3696      gym = gy - gtest
3697
3698 ! Call dcstep to update stx, sty, and to compute the new step.
3699 !
3700      call dcstep(stx,fxm,gxm,sty,fym,gym,stp,fm,gm,brackt,stmin,stmax)
3701
3702 !      Reset the function and derivative values for f
3703
```

```

3704         fx = fxm + stx*gtest
3705         fy = fym + sty*gtest
3706         gx = gxm + gtest
3707         gy = gym + gtest
3708
3709         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.

```

3881 !      First case: A higher function value. The minimum is bracketed.
3882 !      If the cubic step is closer to stx than the quadratic step, the
3883 !      cubic step is taken, otherwise the average of the cubic and
3884 !      quadratic steps is taken.
3885 !      theta, three, gamma are parameters
3886         if (fp .gt. fx) then
3887             theta = three*(fx - fp)/(stp - stx) + dx + dp
3888             s = max(abs(theta),abs(dx),abs(dp))
3889             gamma = s*sqrt((theta/s)**2 - (dx/s)*(dp/s))
3890             if (stp .lt. stx) gamma = -gamma
3891             p = (gamma - dx) + theta
3892             q = ((gamma - dx) + gamma) + dp
3893             r = p/q
3894             stpc = stx + r*(stp - stx) !quadratic step
3895             stpq = stx + ((dx/((fx - fp)/(stp - stx) + dx))/two)* &
3896                                     (stp - stx) !The
3897
3898         cubic step
3899         if (abs(stpc-stx) .lt. abs(stpq-stx)) then
3900             stpf = stpc
3901         else
3902             stpf = stpc + (stp - stpc)/two
3903         endif
3904         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 [15]

```

4425         if (fp .ge. ftest) then
4426             sty = stp

```

```
4427         fy = fp
4428         dy = dp
4429         brackt = .true.
4430     else
4431 !       if second condition is violated not gone far enough (Wolfe)
4432         if (-dp .ge. gtol*(-ginit)) then
4433             stx = stp
4434             fx = fp
4435             dx = dp
4436         else
4437             return
4438         endif
4439     endif
4440
4441 !Bisection and expansion
4442 if (brackt) then
4443     stp = (sty + stx)/two
4444 else
4445     if (two * stp .le. stpmax) then !Remain within boundaries
4446         ! Still in expansion mode
4447         stp = two * stp
4448     else
4449         brackt = .true.
4450         sty = stp
4451         fy = fp
4452         dy = dp
4453     endif
4454 endif
4455 return
4456 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 [\[8\]](#)

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](#).

```
1 #!/bin/bash
2
3 for i in {0..20}
4 do
5     cat > pbs.script.rosenbrock.$i <<EOF
6     #!/bin/bash
7
8     #PBS -l nodes=1:ppn=8,walltime=48:00:00
9     #PBS -m abe
10    #PBS -M weh227@nyu.edu
11    #PBS -N rosenbrockHD$((i))
12
13    module load gcc/4.7.3
14
15    cd /scratch/weh227/rosenbrock/
16    p=$(bc -l <<< "1+10 ^(-$((i)))")
17    for n in 1000 100000 1000000 10000000 100000000
18    do
19        echo 10 \$n \$p
```

```

20      ~/Documents/thesis/lbfgsfortran/./rosenbrockp 10 $n $p >> outputrosenbrock$
      ((i)).txt
21 done
22
23 exit 0;
24
25 EOF
26 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=1.0000000001000000000000
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] 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. ISSN 1064-8275. doi: 10.1137/0916069. URL <http://dx.doi.org/10.1137/0916069>.
- [2] 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. ISSN 0020-2932.
- [3] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970. doi: 10.1093/comjnl/13.3.317. URL <http://comjnl.oxfordjournals.org/content/13/3/317.abstract>.
- [4] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Math. Comp.*, 24:23–26, 1970. ISSN 0025-5718.
- [5] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, 24:647–656, 1970. ISSN 0025-5718.
- [6] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comp.*, 35(151):773–782, 1980. ISSN 0025-5718. doi: 10.2307/2006193. URL <http://dx.doi.org/10.2307/2006193>.
- [7] Ciyu Zhu, Richard Byrd, Jorge Nocedal, and Jose Luis Morales. Lbfgsb 3.0. <http://www.ece.northwestern.edu/~nocedal/lbfgsb.html>, 2011.
- [8] Wilmer Henao. lbfgsbns. <https://github.com/wilmerhenao/lbfgsbNS>, 2014.
- [9] Ciyu 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. ISSN 0098-3500. doi: 10.1145/279232.279236. URL <http://dx.doi.org/10.1145/279232.279236>.
- [10] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999. ISBN 0-387-98793-2. doi: 10.1007/b98874. URL <http://dx.doi.org/10.1007/b98874>.

- [11] 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. ISSN 0036-1429. doi: 10.1137/0725029. URL <http://dx.doi.org/10.1137/0725029>.
- [12] Jorge J. Moré and Gerardo Toraldo. Algorithms for bound constrained quadratic programming problems. *Numer. Math.*, 55(4):377–400, 1989. ISSN 0029-599X. doi: 10.1007/BF01396045. URL <http://dx.doi.org/10.1007/BF01396045>.
- [13] 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. ISSN 1052-6234. doi: 10.1137/S1052623499354242. URL <http://dx.doi.org/10.1137/S1052623499354242>.
- [14] Adrian S. Lewis and Michael L. Overton. Nonsmooth optimization via quasi-Newton methods. *Math. Program.*, 141(1-2, Ser. A):135–163, 2013. ISSN 0025-5610. doi: 10.1007/s10107-012-0514-2. URL <http://dx.doi.org/10.1007/s10107-012-0514-2>.
- [15] Michael Overton, James Burke, and Anders Skajaa. Hanso 2.02. <http://www.cs.nyu.edu/faculty/overton/software/hanso/>, 2012.
- [16] 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.
- [17] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comput. J.*, 3:175–184, 1960/1961. ISSN 0010-4620.
- [18] Allan Kaku. Implementation of high-precision arithmetic in the bfgs method for nonsmooth optimization. Master’s thesis, New York University, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012, 2011.
- [19] 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. ISSN 1477-870X. doi: 10.1002/qj.935. URL <http://dx.doi.org/10.1002/qj.935>.