New York University

Master's Thesis

Implementation of an L-BFGS-B Optimizer applied to NonSmooth Functions and some experimentations

Author:
Wilmer Henao

Supervisor:
Michael L. Overton

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

in the

Courant Institute of Mathematical Sciences
Department of Mathematics

March 2014

Declaration of Authorship

I, Wilmer Henao, declare that this thesis titled, 'Implementation of an L-BFGS-B Optimizer applied to NonSmooth Functions and some experimentations' 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

Faculty Name
Department of Mathematics

Master of Science in Scientific Computing

Implementation of an L-BFGS-B Optimizer applied to NonSmooth Functions and some experimentations

by Wilmer Henao

The Thesis Abstract is written here ...

Acknowledgements

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

I also would like to thank Allan Kaku and Anders Skaaja for earlier contributions to other versions of the code.

Finally, I would like to thank High Performance Computing at NYU.

Contents

LAH **L**ist **A**bbreviations **H**ere

For/Dedicated to/To my...

Chapter 1

Introduction

The goal in this thesis is to find a solution of the nonsmooth 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 $f \colon \mathbb{R}^n \to \mathbb{R}$, n is a very large but finite number. And l_i and $u_i \in \mathbb{R}$

Larger problems not only mean that their solution will take a longer time to solve. But storing and calculating a the necessary matrices depends on the capabilities of the machine used to solve the problem and this might be prohibitively expensive. There a few large scale optimization techniques that have already been developed for the case when n is very large. Also, several techniques have already been developed to handle this type of problems as long as the function f is smooth. But there is not much out there about large scale problems with nonsmooth f

In this thesis $f(x)$ is a nonsmooth function. This small change will require a few changes in the solution algorithm.

For the particular case when n is a small number, several methods that solve optimization problems of nondifferentiable functions in lower dimensions [1] have been developed. This thesis will try to see if it is possible to bring some of those concepts to large scale optimization.

In the case of smooth functions, it is possible to use Newton iteration algorithms and achieve quadratic convergence, the problem with Newton algorithms is that they require

second derivatives to be provided[1]. In the 1950's and several years after that, several "quasi-newton" methods were proposed where the second derivative Hessian matrix is approximated step by step [2]. These approximations or "updates" are calculated after every iteration of the original algorithm and the way in which this update is found defines a new method depending on the particular needs. This thesis will only be concerned with the $BFGS$. [2] which can achieve super linear convergence, has proven to work in most practical purposes and posseses very nice self correcting features [3]. In $BFGS$, it doesn't matter that one update incorrectly estimates the curvature in the objective function, $BFGS$ will always correct itself in just a few steps. This self-correcting property is very desired in the nonsmooth case, since changes in curvature could be abrupt near the optimal point.

$BFGS$ was originally developed for small to medium sized problems, and it is not the right tool for large scale optimization and therefore an $L-BFGS$ adaptation is needed to solve large scale problems1.1.

A final assumption in this thesis is that the Hessian matrix is not sparse. In this case, there are other algorithms that may be more suitable [4, 5], some of them have even been implemented in fortran [6].

This thesis builds upon the original $L-BFGS-B$ code [7] that solves smooth problems of f. There were three main changes in the code. The first one is the line search descent and curvature conditions which required a weaker version of the curvature in order to satisfy the different structure that a nonsmooth function requires. The second one is the line search methodology which was changed from a cubic interpolation to a bisection algorithm and last change in the thesis was the termination condition.

Nocedal's original algorithm consists of 2 steps. In the first step or gradient projection, most of the dimensions in the problem should be removed, making the problem a lot simpler. In the second step there is some fine tuning to guarantee better than just linear speed of convergence.

[1] the main issue with the second derivative is that it requires a total of $n \times n$ partial derivatives. Which is impractical for medium and for some small-size problems

[2] BFGS stands for the last names of its authors Broyden, Fletcher, Goldfarb and Shanno

Chapter 2

Original algorithm

The original algorithm [8] has an accompanying *FORTRAN* sortware [7] and this thesis builds upon that software by making sufficient changes to make it apply to the nonsmooth case.

2.1 Gradient Projection

The original algorithm was created for the case when n is large and f is nonsmooth. Its first step is a gradient projection similar to the one outlined in [9, 10] which is used to determine an active set corresponding to those variables that are bound at each step. The active set defined at point x^* is:

$$\mathcal{A}(x^*) = \{i \in \{1 \dots n\} | x_i^* = l_i \vee x_i^* = u_i\} \tag{2.1}$$

It seems like working on this active set is efficient in large scale problems and according to previous research [11] the gradient projection step is able to find most of the active set variables in a single stroke.

In fact, The reason why a projected gradient step is taken, is because a line search usually changes the active set by one variable at a time during the line search step [1]. So, if 1 million constraints are active at a nondegenerate solution, at least 1 million iterations will be needed just to get to that point. Gradient projection gets rid of that problem, diminishing the number of iterations, and at the same reducing the number of variables for the next step.

[1] the line search is cut short immediately after the first bound is hit, so only one active constraint will change at every step. Unless the line search hits several constraints at the same time by coincidence, which is very unlikely

Gradient projection works on the approximation model:

$$m_k(x) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{(x - x_k)^T B_k (x - x_k)}{2} \tag{2.2}$$

Where B_k will be a $L - BFGS - B$ approximation to the Hessian $\nabla^2 f$

In this first stage the algorithm starts on the current point x_k searching on the direction $-\nabla f(x_k)$. Whenever this search direction encounters one of the constraints, the search direction turns on the boundaries in order to remain feasible. The path is nothing but the feasible piecewise projection of the steepest descent search 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 $m_k(x)$ on this piecewise gradient projection path is known as the "Cauchy point" x^c.

The cauchy point will have effectively "killed" a good part of the dimensionality of the problem.

2.1.1 Subspace Minimization

The problem with gradient projection is that it eventually becomes steepest descent. In fact, gradient projection is exactly steepest descent if it were not for the existence of the constraints. The main problem with steepest descent is that it does not take advantage of the information given by the curvature of the function which causes it to have a slow (linear) speed of convergence. It is for this reason that a stage two is necessary. The idea is to use an $L - BFGS$ type of optimization only on the active set defined on the cauchy point $\mathcal{A}(x^c)$.

The idea at a higher level is to solve the constrained problem 2.2, but only on those dimensions that are free (not at bound). The starting point for this new problem will be the previously found cauchy point x^c, and the algorithm only moves in a direction that lives in the space of the free variables. In the end, the $L - BFGS$ approximation will provide a search direction $\hat{d}^u$

The algorithm will move in the direction set forth by $\hat{d}^* = \alpha^* \hat{d}^u$ where α^* is chosen so that the new point $\bar{x}_i$ satisfies some descent and curvature conditions, and stays within the constraints originally imposed. Once this step is finished, the next and final step will be the termination condition. If the termination condition fails, this algorithm updates the matrix and goes back to the first step of gradient projection.

Chapter 3

Modifications to the original algorithm

As it was mentioned earlier there were three main changes in the original code by Nocedal[7]. They were the line search Wolfe conditions, the line search methodology, and the termination condition.

3.1 Wolfe conditions

Probably the most important change made to the original code was the change in the curvature condition. Originally there are two Wolfe conditions, one of them is the Armijo condition, also known as the sufficient decrease requirement. The other one is the curvature condition, of which the most popular version is the strong wolfe curvature condition:

$$|p_k^T \nabla f(x_k + \alpha_k p_k)| \leq |p_k^T \nabla f(x_k)| \tag{3.1}$$

The strong wolfe is a more natural way to see and achieve convergence, but the problem is that it does not work well for the nonsmooth case. This is because near the minimal points, there may be abrupt changes in curvature. In these cases there is no other option but to relax the curvature condition as long as the sufficient decrease condition is satisfied. The suggested new decrease condition is this one:

$$p_k^T \nabla f(x_k + \alpha_k p_k) \geq p_k^T \nabla f(x_k) \tag{3.2}$$

It is noticeable that with this new condition the algorithm does not crash in the problems tested, as opposed to when the hard wolfe condition is used.

3.2 cubic interpolation replaced with line search

The original software by Nocedal[7], included a cubic line search. The idea of a cubic interpolation line search is to take advantage of the smooth properties of function f in order to find a more convenient point. But in the case of nonsmooth functions, this line search does not serve our purposes and therefore a more typical line search that implements a bisection has to be used.

In general, a step length is selected. If this step length does not satisfy the sufficient decrease and curvature conditions then a step length of half or double the size is selected. the algorithm is guaranteed to converge under a careful selection of the parameters. However, in case this does not happen, the software will stop with a warning.

3.3 Convex Hull and termination conditions

The most important requirement of a practical algorithm is that it ends in a finite time. For the case of smooth functions, the formal way to check whether the algorithm has finished, is to check whether the projected gradient has norm zero 0 wherever the constraints are not at bound. In the case of nonsmooth functions however, this is not necessarily true and the function at the minimum, may have a kink. In this kink the projected gradient may not vanish. Furthermore, if there is a sequence of points that approaches the optimum x from the right, 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 the left.

Given this set of conditions, there is the need for a special set of rules to establish the finalization of each optimization.

Since BFGS approximations typically converge to Clarke-Stationary points. The right methodology should be to calculate the subgradient and to see wheter zero 0 is part of this subgradient. One particular methodology that guarantees an end to the algorithm is suggested in [12]. In order to make sure that the gradient zero $\vec{0}$ is part of the subgradient calculated over a neighbourhood of the optimum. The algorithm keeps a record of the latest gradient vectors in a small neighbourhood of the point where it suspect that the optimum is located. This collection of gradients spans an associated

convex hull of gradients. If this convex hull contains at least one vector of norm smaller than a small number τ_k, the algorithm ends.

Of course the best way to find a vector with such properties is to find the vector with the minimal norm that resides in the convex hull.

3.3.1 minimization of the quadratic program

The termination condition is guaranteed to end up at a local optimum. However in order to find the vector with the minimal norm one subalgorithm needs to be solved. This subalgorithm is a practical primal-dual algorithm implemented in [13]. In this case in particular the best solution is to implement a variation of Mehrotra's Predictor-Corrector algorithm applied to quadratic programming. The primal dual method requires the solution of a system in order to calculate the search direction. The most expensive part of this solution is the calculation of the cholesky decomposition. Mehrotra's algorithm uses the same cholesky decomposition to calculate both directions. the predictor, and the corrector.

Currently there is not a theoretical calculation of the complexity of this algorithm but it is widely used in practice with great results. The implementation on [13] is is exactly the one on [11]. In this thesis it was implemented in $FORTRAN$ as part of the optimizer.

Chapter 4

the functions to be tested

In order to make some tests, a few functions will be evaluated. The most important function to test this non-smooth optimizer is a modified version of rosenbrock's:

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

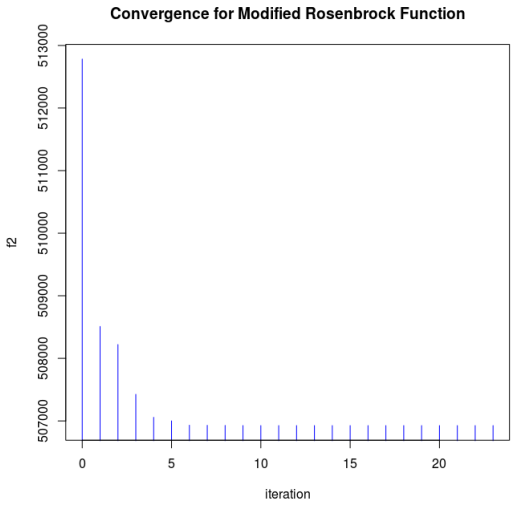
Where the value of p changes the behaviour of the optimizer. This function can be proven to be lipschitz continuous whenever $p > 1$ if restricted to the domain defined by

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

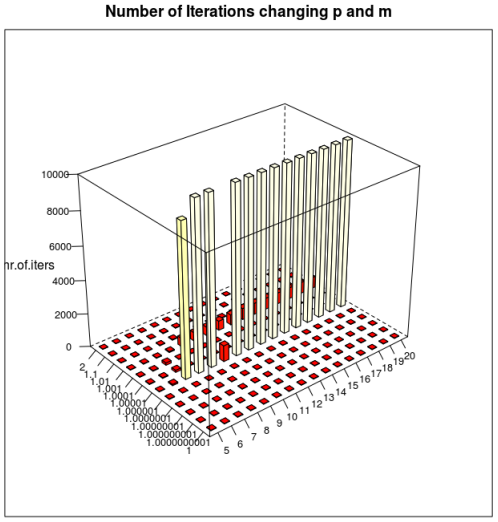
and in fact, whenever the function is restricted to a finite domain this function will be lipschitz continuous for $p > 1$. Whenever $p > 1$ the function $f(z) = |z|^p$ is zero 0 around zero because the derivative $p|z|^{p-1}$ is zero whenever z tends to zero from the right. (this is also the case from the left because it is an even function). However the second derivative will not be as nice.

For the case when $p \leq 1$ the second derivative tends to infinity. $\lim_{x \to 0^+} f' = \infty$. Which is already well known given the "heavyside" look of $f(z) = |z|$.

The convergence of the algorithm smoothly descends to the objective



The converge is adversely affected by the selection of p as one would expect. Values of p descending to 1 make the function less "smooth" and have the adverse effect of making the convergence much more difficult. In this exercise it is noticeable how slow the convergence becomes for a few specific values of p. In particular for 1.0001



$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k p_k^T \nabla f(x_k) \tag{4.3}$$

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] Krzysztof C. Kiwiel. *Methods of Descent for Nondiferentiable Optimization*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

[2] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, SIAM Publications, Englewood Cliffs, NJ, 1983, 1993.

[3] Jorge Nocedal. Theory of algorithms for unconstrained optimization. *Acta Numerica*, 1:199–242, January 1992.

[4] Roger Fletcher, Andreas Grothey, and Sven Leyffer. Computing sparse hessian and jacobian approximations with optimal hereditary properties. Technical report, Large-Scale Optimization with Applications, Part II: Optimal Design and Control, 1996.

[5] R. Fletcher. An optimal positive definite update for sparse hessian matrices. *SIAM Journal on Optimization*, 5:192–218, 1995.

[6] Prof. Dr. Ph. L. Toint Dr. A. R. Conn, Dr. N. I. M. Gould. *LANCELOT: A fortran package for large-scale nonlinear optimization (Release A)*. Springer Series in Computational Mathematics, Berlin Heidelberg, 1992.

[7] J. Nocedal; S. Wright. Lbfgsb 3.0. http://www.ece.northwestern.edu/~nocedal/lbfgsb.html, 2011.

[8] J. Nocedal R.H. Byrd, P. Lu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[9] PH.L.Toint A.R. Conn, N.I.M. Gould. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal of Numerical Analysis*, 25:433–460, 1988.

[10] G. Toraldo J.J. More. Algorithms for bound constrained quadratic programming problems. *Numerical Math.*, 55:377–400, 1989.

[11] Stephen J. Wright Jorge Nocedal. *Numerical Optimization*. Springer Series in Operations Research, 2nd edition, 2006.

[12] AdrianS. Lewis and MichaelL. Overton. Nonsmooth optimization via quasi-newton methods. *Mathematical Programming*, 141(1-2):135–163, 2013. ISSN 0025-5610. doi: 10.1007/s10107-012-0514-2. URL http://dx.doi.org/10.1007/s10107-012-0514-2.

[13] Anders Skaaja. 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.