A Derivative-Free Approximate Gradient Sampling Algorithm for Finite Minimax Problems

by

Julie Ann Nutini

B.Sc. Hons., The University of British Columbia, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The College of Graduate Studies

(Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

April 2012

Abstract

Mathematical optimization is the process of minimizing (or maximizing) a function. An algorithm is used to optimize a function when the minimum cannot be found by hand, or finding the minimum by hand is inefficient. The minimum of a function is a critical point and corresponds to a gradient (derivative) of 0. Thus, optimization algorithms commonly require gradient calculations. When gradient information of the objective function is unavailable, unreliable or 'expensive' in terms of computation time, a derivative-free optimization algorithm is ideal. As the name suggests, derivative-free optimization algorithms do not require gradient calculations. In this thesis, we present a derivative-free optimization algorithm for finite minimax problems. Structurally, a finite minimax problem minimizes the maximum taken over a finite set of functions. We focus on the finite minimax problem due to its frequent appearance in real-world applications. We present convergence results for a regular and a robust version of our algorithm, showing in both cases that either the function is unbounded below (the minimum is $-\infty$) or we have found a critical point. Theoretical results are explored for stopping conditions. Additionally, theoretical and numerical results are presented for three examples of approximate gradients that can be used in our algorithm: the simplex gradient, the centered simplex gradient and the Gupal estimate of the gradient of the Steklov averaged function. A performance comparison is made between the regular and robust algorithm, the three approximate gradients, and the regular and robust stopping conditions. Finally, an application in seismic retrofitting is discussed.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First, and foremost, I would like thank my supervisor, Dr. Warren Hare, whose guidance, patience and encouragement have been unfailing. I am incredibly grateful for the numerous occasions when his confidence in my ability far exceeded my own, and feel truly fortunate to have worked under his supervision. Without him, this thesis would not have come to its fruition, my passport would still be without stamps and the idea of 'proof by fairy dust' would still be a logical option.

I would like to thank my committee members, Dr. Heinz Bauschke, Dr. Shawn Wang and Dr. Erik Rosolowsky, for their time and helpful suggestions throughout this challenging, yet rewarding journey.

I would like to express my sincere gratitude to Dr. Claudia Sagastizábal for our conversations in Chile, especially with respect to the Goldstein approximate subdifferential.

I would like to thank Kasra Bigdeli and Dr. Solomon Tesfamariam for taking an interest in my algorithm and for furthering my passion for algorithm design.

I would like to thank Dr. Rebecca Tyson and Dr. Blair Spearman for encouraging me to pursue studies in optimization, and Dr. Yves Lucet for showing an invested interest in my research through his helpful, thought provoking questions.

I would like to thank UBC Okanagan and the many donors who have provided me with incredible financial support over the years.

Last, but not least, I would like to thank my family and friends for their understanding, faith and endless love throughout my studies.

Many of the results in this thesis are currently in submission to Computational Optimization and Applications (see [HN12]).

To my dad, who shares my love of mathematics,
and to my mom, who is my constant reminder...*va piano*.

Chapter 1

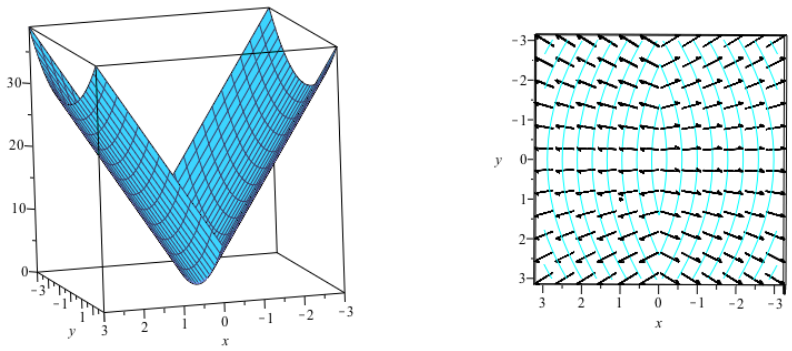
Introduction

Mathematical optimization is the process of minimizing (or maximizing) a function. We use an optimization algorithm when either the minimum of a function cannot be found by hand, or finding the minimum by hand is inefficient.

In the first year calculus classroom, optimization algorithms are simple: take the derivative of a function, set the derivative equal to 0 and solve for x. For the functions explored in first year calculus, this process generally works fine. However, many real-world problems result in nonsmooth functions.

For example, we consider the following nonsmooth finite max function:

$$f(x, y) = 10|x| + y^2 = \max\{10x + y^2, -10x + y^2\}.$$



(a) Graph of $f(x, y)$.

(b) Selected gradients of $f(x, y)$.

Figure 1.1: Graph of $f(x, y)$ and its gradients.

We can see in Figure 1.1(a) that $f(x, y)$ has a ridge along the line $x = 0$ where the function is not differentiable. Figure 1.1(b) shows the pattern of gradients for $f(x, y)$ as if looking down onto the bottom ridge. Notice

that for any point far from the ridge, the negative of the gradient points roughly towards the minimum. However, for any point near the ridge, the negative of the gradient points roughly perpendicular to the ridge, which is not necessarily the direction of the minimum.

In this thesis we develop a novel optimization algorithm specifically designed for functions with this finite max structure.

1.1 Notations

In this thesis, we use the following notations:

1. $|\cdot|$ denotes the Euclidean norm and $\|\cdot\|$ denotes the corresponding matrix norm.

2. As defined in [Cla90], the function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz at a point $x \in \mathbb{R}^n$ if there exists a scalar L and $\delta > 0$ such that

$$|f(y_1) - f(y_2)| \leq L|y_1 - y_2|$$

   for all y_1 and y_2 in the open ball of radius δ about x.

3. If f_1 and f_2 are continuous, then $\max\{f_1, f_2\}$ is continuous. If f_1 and f_2 are Lipschitz, then $\max\{f_1, f_2\}$ is Lipschitz [RW98, Prop 9.10]. However, we note that even if f_1 and f_2 are differentiable, this does not imply that $\max\{f_1, f_2\}$ is differentiable.

1.2 The Problem

We consider the finite minimax problem

$$\min_x f(x) \text{ where } f(x) = \max\{f_i(x) : i = 1, \ldots, N\},$$

where each individual f_i is continuously differentiable.

Finite minimax problems occur in numerous applications, such as portfolio optimization [CTYZ00], control system design [IOKK04], engineering design [Pol87], and determining the cosine measure of a positive spanning set [CSV09, Def 2.7]. In a finite max function, although each individual f_i may be smooth, taking the maximum forms a nonsmooth function with 'nondifferentiable ridges'. For this reason, most algorithms designed to solve finite minimax problems employ some form of smoothing technique;

[PGL93], [PRW03], [Pol88], and [Xu01] (among many others). In general, these smoothing techniques require gradient calculations.

However, in many applications gradient information is not available or can be difficult to compute accurately (see [BJF+98], [DV04], [Har10], [MFT08] and [CSV09, Chpt 1] for some examples of such situations). Thus, for the purpose of this thesis, we further restrict ourselves to the field of derivative-free optimization, where we are only permitted to compute function values, i.e., we cannot compute gradient values ∇f_i directly.

1.3 Derivative-Free Optimization

The research area of derivative-free optimization (DFO) has blossomed in recent years. As previously stated, DFO algorithms are useful in situations when gradient information is not available, difficult to compute accurately or 'expensive' to compute in relation to computation time. For example, if a function is given by a simulation, then gradient information may not be available. For a thorough introduction to several basic DFO frameworks and convergence results for each, see [CSV09].

As there are no gradient calculations required in DFO algorithms, it is assumed that function evaluations are the most 'expensive' computations in terms of CPU time. Thus, the performance of a DFO algorithm is based on the number of function evaluations required to solve the problem.

1.4 DFO and Finite Max Functions

In relation to our problem, research on optimizing finite max functions without calculating derivatives can be seen as early as 1975 [Mad75], while more recently we have seen a resurface in this area, [LLS06] and [HM11].

In 2006, Liuzzi, Lucidi and Sciandrone used a smoothing technique based on an exponential penalty function in a directional direct-search framework to form a derivative-free optimization method for finite minimax problems [LLS06]. This method is shown to globally converge towards a standard stationary point of the original finite minimax problem.

We want to take a step away from the prevalent smoothing techniques used to solve finite minimax problems. Instead of altering the function with a smoothing technique, we look to solve the finite minimax problem with a DFO method that exploits its smooth substructure. To understand how we do this, we first need to define the following terms.

Definition 1.1. Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz at a point $\bar{x}$. Then by Rademacher's Theorem ([RW98, Thm 9.60]), f is differentiable almost everywhere on $\mathbb{R}^n$. Let Ω_f be the set of points at which f fails to be differentiable. Then the **Clarke subdifferential**, as defined in [Cla90], is given by the set

$$\partial f(\bar{x}) = \text{conv}(\lim_j \nabla f(y^j) : y^j \to \bar{x},\ y^j \notin \Omega_f). \tag{1.1}$$

Basically, the subdifferential of a function at $\bar{x}$ is the set of all possible gradients near $\bar{x}$. As an example, we consider the absolute value function, $f(x) = |x|$ in $\mathbb{R}$. Clearly, $f(x)$ is not differentiable at 0. The subdifferential of f at the point 0 is the set

$$\partial f(0) = \text{conv}(1, -1),$$

as

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}.$$

Definition 1.2. A **descent direction** of a continuously differentiable function f at a given point $\bar{x} \in \text{dom}(f)$ as defined in [CSV09] is any vector d such that

$$\langle d, v \rangle < 0$$

for all $v \in \partial f(\bar{x})$.

To explain how subdifferentials can be used to find a descent direction, we first define the projection.

Definition 1.3 (Definition 3.7, [BC11])**.** Let $C \subseteq \mathbb{R}^n$ be a nonempty closed convex set and let $x \in \mathbb{R}^n$. Let $p \in C$. Then p is the unique projection of x onto C, denoted by $\text{Proj}(x|C)$, if

$$p \in \arg\min_y \{|y - x| : y \in C\}.$$

A point $\bar{x}$ is a (Clarke) stationary point of a function f when $0 \in \partial f(\bar{x})$. Thus, we define the direction of steepest descent as follows.

Definition 1.4. The **direction of steepest descent** as defined in [CSV09] is given by

$$d = -\text{Proj}(0|\partial f(\bar{x})).$$

In 2011, Hare and Macklem presented a derivative-free method that exploits the smooth substructure of the finite minimax problem. It combines the frameworks of a directional direct search method [CSV09, Chpt 7] and the gradient sampling algorithm (GS algorithm) presented in [BLO02] and [BLO05]. Loosely speaking, the GS algorithm uses a collection of local gradients to build a 'robust subdifferential' of the objective function and uses this to determine a 'robust descent direction'. In [HM11], these ideas are used to develop several methods to find an approximate descent direction that moves close to parallel to an 'active manifold'. During each iteration, points are sampled from around the current iterate and the simplex gradient is calculated for each of the active functions of the objective function. The calculated simplex gradients are then used to form an approximate subdifferential, which is then used to determine a likely descent direction.

Ideas from the GS algorithm have appeared in two other recent DFO methodologies: [BKS08] and [Kiw10].

In 2008, Bagirov, Karasözen and Sezer presented a discrete gradient derivative-free method for unconstrained nonsmooth optimization problems [BKS08]. Described as a derivative-free version of the bundle method presented in [Wol75], the method uses discrete gradients to approximate subgradients of the function and build an approximate subdifferential. The analysis of this method provides proof of convergence to a Clarke stationary point for an extensive class of nonsmooth problems. In this thesis, we focus on the finite minimax problem. This allows us to require few (other) assumptions on our function while maintaining strong convergence analysis. It is worth noting that we use the same set of test problems as in [BKS08]. Specifically, we use the [LV00] test set and exclude one problem as its subfunctions are complex-valued. (The numerics in [BKS08] exclude the same problem, and several others, without explanation.)

Using approximate gradient calculations instead of gradient calculations, the GS algorithm is made derivative free by Kiwiel in [Kiw10]. Specifically, Kiwiel employs the *Gupal estimate* of the gradient of the *Steklov averaged function* (see [Gup77] or Section 4.3 herein) as an approximate gradient. It is shown that, with probability 1, this derivative-free algorithm satisfies the same convergence results as the GS algorithm – it either drives the f-values to $-\infty$ or each cluster point is found to be Clarke stationary [Kiw10, Theorem 3.8]. No numerical results are presented for Kiwiel's derivative-free algorithm.

1.5 Method of Steepest Descent

To obtain a general understanding of the framework of the algorithm presented in this thesis, we recall the classical method of steepest descent.

1. Search Direction: Set $d^k = -\operatorname{Proj}(0|\partial f(x^k))$.
2. Step Length: Find t_k by solving $\min_{t_k>0}\{f(x^k + t_k d^k)\}$.
3. Update: Set $x^{k+1} = x^k + t_k d^k$. Increase $k = k + 1$. Loop.

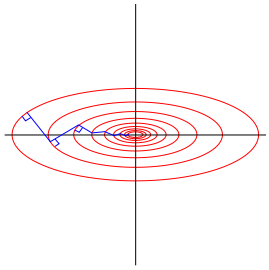


Figure 1.2: An example of the method of steepest descent.

In each iteration, the method of steepest descent calculates the direction of steepest descent, i.e., $-\operatorname{Proj}(0|\partial f(x^k))$, does a line search in this direction and then updates the iterate. This allows the algorithm to move perpendicular to the contours of the function, towards the minimum. The method of steepest descent works well for smooth functions. However, for nonsmooth functions, the optimal solution often occurs at a point of nondifferentiability, which results in a very slow convergence rate for the method of steepest descent.

In general, our algorithm takes the method of steepest descent and makes it derivative free; instead of calculating the subdifferential, we calculate an approximate subdifferential; instead of finding the direction of steepest descent, we project 0 onto our approximate subdifferential and find an approximate direction of steepest descent. With these alterations, our algorithm is able to navigate along nondifferentiable ridges, more rapidly minimizing the function.

Specifically, we use the GS algorithm framework to form a derivative-free approximate gradient sampling algorithm. As we are dealing with finite max functions, instead of calculating an approximate gradient of f at each

of the sampled points, we calculate an approximate gradient of each of the active functions at the current iterate. We say a function f_i is active if its index is an element of the active set.

Definition 1.5. We define the **active set** of f at $\bar{x}$ to be the set of indices

$$A(\bar{x}) = \{i : f(\bar{x}) = f_i(\bar{x})\}. \tag{1.2}$$

We denote the set of **active gradients** of f at $\bar{x}$ by

$$\{\nabla f_i(\bar{x})\}_{i \in A(\bar{x})}.$$

Expanding the active set to include 'almost' active functions, we also present a robust version of our algorithm, which is more akin to the GS algorithm. In this robust version, when our iterate is close to a point of nondifferentiability, the size and shape of our approximate subdifferential will reflect the presence of 'almost active' functions. Hence, when we project 0 onto our approximate subdifferential, the descent direction will direct minimization parallel to a 'nondifferentiable ridge', rather than straight at this ridge. It can be seen in our numerical results that these robust changes greatly influence the performance of our algorithm.

The algorithm presented in this thesis differs from those mentioned above in a few key manners. Unlike in [LLS06] we do not employ a smoothing technique. Unlike in [HM11], which uses the directional direct-search framework to imply convergence, we employ an approximate steepest descent framework. Using this framework, we are able to analyze convergence directly and develop stopping analysis based on our stopping conditions for the algorithm. Unlike in [BKS08] and [Kiw10], where convergence is proven for a specific approximate gradient, we prove convergence for *any* approximate gradient that satisfies a simple error bound dependent on the sampling radius. As examples, we present the simplex gradient, the centered simplex gradient and the Gupal estimate of the gradient of the Steklov averaged function. (As a side-note, Section 4.3 also provides, to the best our knowledge, a novel error analysis of the Gupal estimate of the gradient of the Steklov averaged function.) Furthermore, unlike in [Kiw10], where convergence to a stationary point is proved with probability 1, we prove convergence to a critical point for every cluster point of a sequence generated by our algorithm.

Focusing on the finite minimax problem provides us with an advantage over the methods of [BKS08] and [Kiw10]. In particular, we only require order n function calls per iteration (where n is the dimension of the problem),

while both [BKS08] and [Kiw10] require order mn function calls per iteration (where m is the number of gradients they approximate to build their approximate subdifferential). (The original GS algorithm suggests that $m \approx 2n$ provides a good value for m.)

1.6 Basic Definitions and Results

We denote by $\mathcal{C}^1$ the class of differentiable functions whose gradient mapping ∇ is continuous. As stated previously, we assume that each of the individual f_i in the finite max function is continuously differentiable, i.e., $f_i \in \mathcal{C}^1$.

We denote by $\mathcal{C}^{1+}$ the class of continuously differentiable functions whose gradient mapping ∇ is locally Lipschitz. We say $f \in \mathcal{C}^{1+}$ with constant L if its **gradient mapping** ∇ has a Lipschitz constant L.

We denote by $\mathcal{C}^{2+}$ the class of twice continuously differentiable functions whose gradient mapping ∇^2 is locally Lipschitz. We say $f \in \mathcal{C}^{2+}$ with constant L if its **gradient mapping** ∇^2 has a Lipschitz constant L.

The following result reveals that the subdifferential for a finite max function at a point $\bar{x}$ is equal to the convex hull of the active gradients of f at $\bar{x}$.

Proposition 1.6. *Let $f = \max\{f_i : i = 1, \ldots, N\}$. If $f_i \in \mathcal{C}^1$ for each $i \in A(\bar{x})$, then $\partial f(\bar{x}) = \operatorname{conv}(\nabla f_i(\bar{x}))_{i \in A(\bar{x})}$.*

Proof. See Proposition 2.3.12, [Cla90]. □

It is important to note that the subdifferential as defined in Proposition 1.6 is a compact set. We formally state and prove this result in the following corollary.

Corollary 1.7. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where $f_i \in \mathcal{C}^1$ for each $i \in A(\bar{x})$. The subdifferential of f at $\bar{x}$ is a compact set.*

Proof. By Proposition 1.6,

$$\partial f(\bar{x}) = \operatorname{conv}(\nabla f_i(\bar{x}))_{i \in A(\bar{x})}.$$

As there are a finite number of functions, $A(\bar{x})$ is a finite set, which implies $\{\nabla f_i(\bar{x})\}$ is a finite set of points. The convex hull of a finite set of points is closed and bounded. Hence, $\partial f(\bar{x})$ is a compact set. □

For our proof of convergence in Section 2.2, we require that ∂f is outer semicontinuous. First, we define the limit superior for a mapping.

Definition 1.8. For a mapping $S : \mathbb{R}^n \to \mathbb{R}^n$, we define the *limit superior* as

$$\limsup_{x \to \bar{x}} S(x) = \{y : \text{ there exists } x^k \to \bar{x} \text{ and } y^k \to y \text{ with } y^k \in S(x^k)\}.$$

Definition 1.9. As defined in [RW98, Def 5.4], S is *outer semicontinuous* (osc) at $\bar{x}$ if

$$\limsup_{x \to \bar{x}} S(x) = S(\bar{x}).$$

To see that the subdifferential is outer semicontinuous, we have the following proposition from [RW98].

Proposition 1.10. *For a continuous function $f : \mathbb{R}^n \to \mathbb{R}$, the mapping ∂f is outer semicontinuous everywhere.*

Proof. See Proposition 8.7, [RW98]. □

Chapter 2

Approximate Gradient Sampling Algorithm

In Chapter 2, we state the approximate gradient sampling algorithm (AGS algorithm), present the details as to when and why the AGS algorithm necessarily finds function value decrease using an arbitrary approximate gradient of each of the active f_i, denoted by $\nabla_A f_i$, and then show that the AGS algorithm converges to a local minimum. We note that no specific approximate gradient is discussed in this section. Furthermore, we emphasize that the error bound requirement used in the convergence results (Section 2.2) for the arbitrary approximate gradient is achievable. Examples of approximate gradients, their structures and their error bounds are discussed in Section 4.

2.1 Algorithm

We first provide a partial glossary of the notation used in the statement of the AGS algorithm.

Table 2.1: Glossary of notation used in AGS algorithm.

Glossary of Notation	
k: Iteration counter	x^k : Current iterate
μ_k: Accuracy measure	Δ_k: Sampling radius
m: Sample size	θ: Sampling radius reduction factor
y^j: Sampling points	Y: Sampled set of points
η: Armijo-like parameter	d^k: Search direction
t_k: Step length	t_{min}: Minimum step length
$\nabla_A f_i$: Approximate gradient of f_i	$A(x^k)$: Active set at x^k
G^k: Approximate subdifferential	ε_{tol}: Stopping tolerance

We state the theoretical AGS algorithm and then provide a detailed description of the algorithm.

Conceptual Algorithm: [AGS Algorithm]

0. Initialize: Set $k = 0$ and input

   x^0 - starting point
   $\mu_0 > 0$ - accuracy measure
   $\Delta_0 > 0$ - initial search radius
   $\theta \in (0, 1]$ - search radius reduction factor
   $0 < \eta < 1$ - Armijo-like parameter
   t_{min} - minimum step length
   $\varepsilon_{tol} > 0$ - stopping tolerance

1. Generate Approximate Subdifferential G^k:
   Generate a set $Y = [x^k, y^1, \dots y^m]$ around the current iterate x^k such that
   $$\max_{j=1,\dots,m} |y^j - x^k| \leq \Delta_k.$$
   Use Y to calculate the approximate gradient of f_i, denoted $\nabla_A f_i$, at x^k for each $i \in A(x^k)$. Set
   $$G^k = \operatorname{conv}(\nabla_A f_i(x^k))_{i \in A(x^k)}.$$

2. Generate Search Direction:
   Let
   $$d^k = -\operatorname{Proj}(0|G^k).$$
   Check if
   $$\Delta_k \leq \mu_k |d^k|. \tag{2.1}$$
   If (2.1) does not hold, then set $x^{k+1} = x^k$, set
   $$\Delta_{k+1} = \begin{cases} \theta \mu_k |d^k| & \text{if } |d^k| \neq 0 \\ \theta \Delta_k & \text{if } |d^k| = 0 \end{cases}, \tag{2.2}$$
   and go to Step 4. If (2.1) holds and $|d^k| < \varepsilon_{tol}$, then STOP. Else, continue to the line search.

3. Line Search:
   Attempt to find $t_k > 0$ such that

$$f(x^k + t_k d^k) < f(x^k) - \eta t_k |d^k|^2.$$

   Line Search Failure:
   Set $\mu_k = \dfrac{\mu_k}{2}$, $x^{k+1} = x^k$ and go to Step 4.

   Line Search Success:
   Let x^{k+1} be any point such that

$$f(x^{k+1}) \leq f(x^k + t_k d^k).$$

4. Update and Loop:
   Set $\Delta_{k+1} = \max\limits_{j=1,\ldots,m} |y^j - x^k|$, $k = k + 1$ and return to Step 1.

■

In Step 0 of the AGS algorithm, we set the iterate counter to 0, provide an initial starting point x^0, and initialize the parameter values.

In Step 1, we create the approximate subdifferential. First, we select a set of points around x^k within a search radius of Δ_k. In implementation, the points are randomly and uniformly sampled from the interior of an n-dimensional hypersphere of radius Δ_k centered at the origin (using the MATLAB `randsphere.m` function [Sta05]). Using this set Y, we then calculate an approximate gradient of each of the active functions at x^k and set the approximate subdifferential G^k equal to the convex hull of these active approximate gradients.

In Step 2, we generate a search direction by solving the projection of 0 onto the approximate subdifferential: $\text{Proj}(0|G^k) \in \arg\min_{g \in G^k} |g|^2$. The search direction d^k is set equal to the negative of the solution, i.e., $d^k = -\text{Proj}(0|G^k)$.

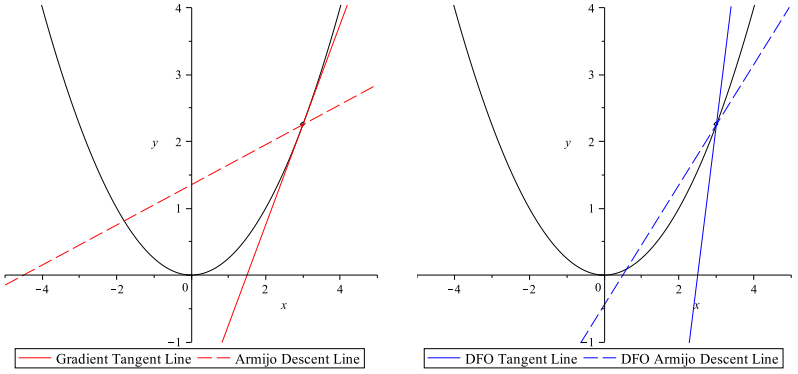
After finding a search direction, we check the condition $\Delta_k \leq \mu_k |d^k|$ (equation (2.1)). This condition determines if the current search radius is sufficiently small relative to the distance from 0 to the approximate subdifferential. If equation (2.1) holds and $|d^k| < \varepsilon_{tol}$, then we terminate the algorithm, as 0 is within ε_{tol} of the approximate subdifferential and the search radius is small enough to reason that the approximate subdifferential is accurate. If equation (2.1) does not hold, then the approximate subdifferential is not sufficiently accurate to warrant a line search, so we decrease

the search radius according to equation (2.2) and loop (Step 4). If equation (2.1) holds, but $|d^k| \geq \varepsilon_{tol}$, then we proceed to a line search.

In Step 3, we carry out a line search. We attempt to find a step length $t_k > 0$ such that the Armijo-like condition holds

$$f(x^k + t_k d^k) < f(x^k) - \eta t_k |d^k|^2. \tag{2.3}$$

The following figures illustrate the Armijo(-like) line search for the curve $f(x) = \frac{1}{4}x^2$. Figure 2.1(a) is representative of a gradient based method (Armijo line search), while Figure 2.1(b) is representative of a DFO method (Armijo-like line search).



(a) A gradient based line search. (b) A derivative-free line search.

Figure 2.1: A graphical depiction of the Armijo and Armijo-like condition.

To satisfy the condition in equation (2.3), the algorithm must find a point on the curve that lies on or below the Armijo descent line for both methods. Any point found below the Armijo descent line will result in a successful line search. Thus, the Armijo(-like) condition prevents the algorithm from taking multiple steps with minimal function value decrease. In other words, it ensures sufficient function value decrease is found when a line search success is declared.

Comparing Figures 2.1(a) and 2.1(b), we can see that the DFO Armijo descent line does not allow the algorithm to accept a point $x \leq 0.5$. This prevents the algorithm from jumping back and forth across the y-axis, thus, resulting in a faster convergence rate.

In implementation, we use a back-tracking line search (described in [NW99]) with an initial step-length of $t_{ini} = 1$. Basically, we test to see if equation (2.3) holds for $t_k = t_{ini}$. If it holds, then we declare a line search

success. If it does not hold, we reduce t_k by a factor of 0.5 and we test equation (2.3) again. If we have $t_k < t_{min}$ without declaring a line search success, then we declare a line search failure. If we find a t_k such that equation (2.3) holds, then we declare a line search success.

If a line search success occurs, then we let x^{k+1} be any point such that

$$f(x^{k+1}) \leq f(x^k + t_k d^k), \tag{2.4}$$

thus, forming a non-increasing sequence of function values $\{f(x^k)\}_{k=0}$. In implementation, we do this by searching through the function values used in the calculation of our approximate gradients ($\{f(y^i)\}_{y^i \in Y}$). As this set of function values corresponds to points distributed around our current iterate, there is a good possibility of finding further function value decrease without having to carry out additional function evaluations. We find the minimum function value in our set of evaluations and if equation (2.4) holds for this minimum value, then we set x^{k+1} equal to the corresponding input point. Otherwise, we set $x^{k+1} = x^k + t_k d^k$.

If a line search failure occurs, then we reduce the accuracy measure μ_k by a factor of 0.5 and set $x^{k+1} = x^k$. (Notice that as μ_k is decreased each time a line search failure occurs, $\{\mu_k\}_{k=0}$ is a non-increasing sequence.)

Finally, in Step 4, we set $\Delta_{k+1} = \max_{j=1,\ldots,m} |y^j - x^k|$, update the iterate counter and loop to Step 1 to resample.

2.2 Convergence

In order to prove convergence for the AGS algorithm, we must show that the direction d^k is a descent direction, that the algorithm is well-defined (eventually finds function value decrease) and that the stopping conditions are sufficient, i.e., when the algorithm satisfies the stopping conditions, the distance to a critical point is controlled by ε_{tol}.

Remark 2.1. For the following results, we denote the approximate subdifferential of f at $\bar{x}$ as

$$G(\bar{x}) = \text{conv}(\nabla_A f_i(\bar{x}))_{i \in A(\bar{x})},$$

where $\nabla_A f_i(\bar{x})$ is the approximate gradient of f_i at $\bar{x}$.

Our first result shows that the approximate subdifferential generated by the AGS algorithm is a good approximate of the exact subdifferential. We use part 1 of the following lemma to establish that our stopping conditions are sufficient. We use part 2 of the following lemma to establish that d^k is a descent direction.

Lemma 2.2. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose there exists an $\varepsilon > 0$ such that $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \varepsilon$ for all i. Then*

1. *for all $w \in G(\bar{x})$, there exists a $v \in \partial f(\bar{x})$ such that $|w - v| \leq \varepsilon$, and*
2. *for all $v \in \partial f(\bar{x})$, there exists a $w \in G(\bar{x})$ such that $|w - v| \leq \varepsilon$.*

Proof. 1. By definition, for all $w \in G(\bar{x})$ there exists a set of α_i such that

$$w = \sum_{i \in A(\bar{x})} \alpha_i \nabla_A f_i(\bar{x}), \qquad \text{where } \alpha_i \geq 0, \sum_{i \in A(\bar{x})} \alpha_i = 1.$$

By Proposition 1.6, as each $f_i \in \mathcal{C}^1$ we have $\partial f(\bar{x}) = \operatorname{conv}(\nabla f_i(\bar{x}))_{i \in A(\bar{x})}$. Using the same α_i as above, we see that

$$v = \sum_{i \in A(\bar{x})} \alpha_i \nabla f_i(\bar{x}) \in \partial f(\bar{x}).$$

Then

$$\begin{aligned} |w - v| &= |\sum_{i \in A(\bar{x})} \alpha_i \nabla_A f_i(\bar{x}) - \sum_{i \in A(\bar{x})} \alpha_i \nabla f_i(\bar{x})| \\ &\leq \sum_{i \in A(\bar{x})} \alpha_i |\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \\ &\leq \sum_{i \in A(\bar{x})} \alpha_i \varepsilon \qquad \text{(by assumption)} \\ &= \varepsilon \qquad \text{(as } \sum_{i \in A(\bar{x})} \alpha_i = 1\text{)}. \end{aligned}$$

Hence, for all $w \in G(\bar{x})$, there exists a $v \in \partial f(\bar{x})$ such that

$$|w - v| \leq \varepsilon. \tag{2.5}$$

2. We have $\partial f(\bar{x}) = \operatorname{conv}(\nabla f_i(\bar{x}))_{i \in A(\bar{x})}$. So for all $v \in \partial f(\bar{x})$, there exist α_i such that

$$v = \sum_{i \in A(\bar{x})} \alpha_i \nabla f_i(\bar{x}) \qquad \text{where } \alpha_i \geq 0, \sum_{i \in A(\bar{x})} \alpha_i = 1.$$

Using the same α_i as above, we see that

$$w = \sum_{i \in A(\bar{x})} \alpha_i \nabla_A f_i(\bar{x}) \in G(\bar{x}).$$

Using the same argument as in part 1, we can conclude that for all $v \in \partial f(\bar{x})$, there exists a $w \in G(\bar{x})$ such that $|w - v| \leq \varepsilon$. □

Our next goal (in Theorem 2.6) is to show that eventually a line search success will occur in the AGS algorithm. To achieve this, we first state the Projection Theorem.

Theorem 2.3. *(The Projection Theorem) Let $C \subseteq \mathbb{R}^n$ be a nonempty closed convex set. Then for every x and p in $\mathbb{R}^n$*

$$p = \text{Proj}(x|C) \iff p \in C \text{ and } \langle x - p, y - p\rangle \leq 0 \text{ for all } y \in C.$$

Proof. See Theorem 3.14, [BC11]. □

We use the Projection Theorem to prove $d = -\text{Proj}(0|G(\bar{x}))$ is a descent direction.

Lemma 2.4. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose there exists an $\varepsilon > 0$ such that $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \varepsilon$ for all i. Define $d = -\text{Proj}(0|G(\bar{x}))$ and suppose $|d| \neq 0$. Let $\beta \in (0, 1)$. If $\varepsilon < (1 - \beta)|d|$, then for all $v \in \partial f(\bar{x})$ we have*

$$\langle d, v\rangle < -\beta|d|^2.$$

Proof. Notice that, by Theorem 2.3, $d = -\text{Proj}(0|G(\bar{x}))$ implies that

$$\langle 0 - (-d), w - (-d)\rangle \leq 0 \text{ for all } w \in G(\bar{x}).$$

Hence,

$$\langle d, w + d\rangle \leq 0 \text{ for all } w \in G(\bar{x}). \tag{2.6}$$

So we have for all $v \in \partial f(\bar{x})$

$$\begin{aligned}
\langle d, v\rangle &= \langle d, v - w + w - d + d\rangle && \text{for all } w \in G(\bar{x}) \\
&= \langle d, v - w\rangle + \langle d, w + d\rangle + \langle d, -d\rangle && \text{for all } w \in G(\bar{x}) \\
&\leq \langle d, v - w\rangle - |d|^2 && \text{for all } w \in G(\bar{x}) \quad \text{(by (2.6))} \\
&\leq |d||v - w| - |d|^2 && \text{for all } w \in G(\bar{x}),
\end{aligned}$$

which follows by using Cauchy Schwarz. For any $v \in \partial f(\bar{x})$, using w as constructed in Lemma 2.2(2), we see that

$$\begin{aligned}
\langle d, v\rangle &\leq |d|\varepsilon - |d|^2 \\
&< |d|^2(1 - \beta) - |d|^2 && (\text{as } \varepsilon < (1 - \beta)|d|) \\
&= -\beta|d|^2.
\end{aligned}$$

□

Using Lemma 2.4, we can easily show that d is a descent direction for f at $\bar{x}$ by setting $\beta = 0$ and bounding ε above by $|d|$. We notice that this condition on ε requires the algorithm to reach a point where the error between the exact gradient and the approximate gradient is smaller than the distance from 0 to our approximate subdifferential.

Corollary 2.5. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose there exists an $\varepsilon > 0$ such that $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \varepsilon$ for all i. Define $d = -\operatorname{Proj}(0|G(\bar{x}))$ and suppose $|d| \neq 0$. If $\varepsilon < |d|$, then $\langle d, v\rangle < 0$ for all $v \in \partial f(\bar{x})$.*

To guarantee convergence, we must show that, except in the case of $0 \in \partial f(x^k)$, the algorithm will always be able to find a search radius that satisfies the requirements in Step 2. In Section 4, we show that (for three different approximate gradients) the value ε (in Lemma 2.4) is linked to the search radius Δ. As unsuccessful line searches will drive Δ to 0, this implies that eventually the requirements of Lemma 2.4 will be satisfied. We formalize this in the next two theorems.

Theorem 2.6. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose $0 \notin \partial f(x^k)$ for each iteration k. Suppose there exists $\bar{K} > 0$ such that given any set of points generated in Step 1 of the AGS algorithm, the approximate gradient satisfies $|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k$ for all i. Let $d^k = -\operatorname{Proj}(0|G(x^k))$. Then for any $\mu > 0$, there exists $\bar{\Delta} > 0$ such that,*

$$\Delta \leq \mu|d^k| + \bar{K}\mu(\Delta_k - \Delta) \qquad \text{for all } 0 < \Delta < \bar{\Delta},$$

Moreover, if $\Delta_k < \bar{\Delta}$, then the following inequality holds

$$\Delta_k \leq \mu|d^k|.$$

Proof. Let $\bar{v} = \operatorname{Proj}(0|\partial f(x^k))$ (by assumption, $\bar{v} \neq 0$).
Given $\mu > 0$, let

$$\bar{\Delta} = \frac{1}{\bar{K} + \frac{1}{\mu}}|\bar{v}|, \tag{2.7}$$

and consider $0 < \Delta < \bar{\Delta}$. Now create $G(x^k)$ and $d^k = -\operatorname{Proj}(0|G(x^k))$. As $-d^k \in G(x^k)$, by Lemma 2.2(1), there exists a $v^k \in \partial f(x^k)$ such that

$$|-d^k - v^k| \leq \bar{K}\Delta_k.$$

Then

$$\begin{aligned} & \bar{K}\Delta_k \geq |-d^k - v^k| \\ \Rightarrow \quad & \bar{K}\Delta_k \geq |v^k| - |d^k| \\ \Rightarrow \quad & \bar{K}\Delta_k \geq |\bar{v}| - |d^k| \qquad (\text{as } |v| \geq |\bar{v}| \text{ for all } v \in \partial f(x^k)\ ). \end{aligned}$$

Thus, for $0 < \Delta < \bar{\Delta}$, we apply equation (2.7) to $|\bar{v}|$ in the above inequality to get

$$\bar{K}\Delta_k \geq (\bar{K} + \frac{1}{\mu})\Delta - |d^k|,$$

which rearranges to

$$\Delta \leq \mu(|d^k| + \bar{K}\mu(\Delta_k - \Delta).$$

Hence, $\Delta \leq \mu|d^k| + \bar{K}\mu(\Delta_k - \Delta)$ for all $0 < \Delta < \bar{\Delta}$.
Finally, if $\Delta_k < \bar{\Delta}$, then

$$\Delta_k \leq \mu|d^k|.$$

□

Remark 2.7. In Theorem 2.6, it is important to note that eventually the condition $\Delta_k < \bar{\Delta}$ will hold. Examine $\bar{\Delta}$ as constructed above: $\bar{K}$ is a constant and $\bar{v}$ is associated with the current iterate. However, the current iterate is only updated when a line search success occurs, which will not occur unless the condition $\Delta_k \leq \mu_k|d^k|$ is satisfied. As a result, if $\Delta_k \geq \bar{\Delta}$, the AGS algorithm will reduce Δ_k, with $\bar{\Delta}$ remaining constant, until $\Delta_k < \bar{\Delta}$.

Recall in Step 3 of the AGS algorithm, for a given $\eta \in (0, 1)$, we attempt to find a step length $t_k > 0$ such that

$$f(x^k + t_k d^k) < f(x^k) - \eta t_k |d^k|^2$$

The following result shows that eventually the above inequality will hold in the AGS algorithm. Recall Corollary 1.7, which states that the exact subdifferential for a finite max function is a compact set. Thus, we know that in the following theorem $\tilde{v}$ is well-defined.

Theorem 2.8. *Fix $0 < \eta < 1$. Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $\tilde{v} \in \arg\max\{\langle d, v\rangle : v \in \partial f(\bar{x})\}$. Suppose there exists an $\varepsilon > 0$ such that $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \varepsilon$ for all i. Define $d = -\operatorname{Proj}(0|G(\bar{x}))$ and suppose $|d| \neq 0$. Let $\beta = \frac{2\eta}{1+\eta}$. If $\varepsilon < (1-\beta)|d|$, then there exists $\bar{t} > 0$ such that*

$$f(\bar{x} + td) - f(\bar{x}) < -\eta t|d|^2 \qquad \text{for all } 0 < t < \bar{t}.$$

Proof. Note that $\beta \in (0,1)$. Recall, from Lemma 2.4, we have for all $v \in \partial f(\bar{x})$

$$\langle d, v\rangle < -\beta|d|^2. \tag{2.8}$$

Using $\beta = \frac{2\eta}{1+\eta}$, equation (2.8) becomes

$$\langle d, v\rangle < -\tfrac{2\eta}{1+\eta}|d|^2 \quad \text{for all } v \in \partial f(\bar{x}). \tag{2.9}$$

From equation (2.8) we can conclude that for all $v \in \partial f(\bar{x})$

$$\langle d, v\rangle < 0.$$

Using Theorem 8.30 in [RW98], we have

$$\lim_{\tau \searrow 0} \frac{f(\bar{x}+\tau d) - f(\bar{x})}{\tau} = \max\{\langle d, v\rangle : v \in \partial f(\bar{x})\} = \langle d, \tilde{v}\rangle < 0.$$

Therefore, as $\frac{\eta+1}{2} < 1$ (since $\eta < 1$) there exists $\bar{t} > 0$ such that

$$\frac{f(\bar{x}+td) - f(\bar{x})}{t} < \frac{\eta+1}{2}\langle d, \tilde{v}\rangle \quad \text{for all } 0 < t < \bar{t}.$$

For such a t, we have

$$\begin{aligned} f(\bar{x}+td) - f(\bar{x}) &< \frac{\eta+1}{2} t\langle d, \tilde{v}\rangle \\ &< -\frac{\eta+1}{2}\frac{2\eta}{\eta+1} t|d|^2 \qquad \text{(by (2.9))} \\ &< -\eta t|d|^2. \end{aligned}$$

Hence,

$$f(\bar{x}+td) - f(\bar{x}) < -\eta t|d|^2 \qquad \text{for all } 0 < t < \bar{t}.$$

□

Combining the previous results, we show that the AGS algorithm is guaranteed to find function value decrease (provided $0 \notin \partial f(x^k)$). We summarize with the following corollary.

Corollary 2.9. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose $0 \notin \partial f(x^k)$ for each iteration k. Suppose there exists a $\bar{K} > 0$ such that given any set of points generated in Step 1 of the AGS algorithm, the approximate gradient satisfies $|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k$ for all i. Then after a finite number of iterations, the algorithm will find a new iterate with a lower function value.*

Proof. Consider x^k, where $0 \notin \partial f(x^k)$.
To find function value decrease with the AGS algorithm, we must declare a line search success in Step 3. The AGS algorithm will only carry out a line search if the stopping condition below is satisfied, i.e.,

$$\Delta_k \leq \mu_k |d^k|. \tag{2.10}$$

In Theorem 2.6, we showed that for any $\mu_k > 0$, there exists a $\bar{\Delta}$ such that if $\Delta_k < \bar{\Delta}$, then equation (2.10) is satisfied. If $\Delta_k > \bar{\Delta}$ in Step 2, i.e., equation (2.10) is not satisfied, then Δ_k is updated according to equation (2.2). Whether $|d^k| \neq 0$ or $|d^k| = 0$, we can see that $\Delta_{k+1} \leq \theta \Delta_k$, so eventually $\Delta_k < \bar{\Delta}$. i.e., equation (2.10) will be satisfied and the AGS algorithm will carry out a line search. Now, in order to have a line search success, we must be able to find a step length t_k such that the Armijo-like condition holds,

$$f(x^k + t_k d^k) < f(x^k) - \eta t_k |d^k|^2.$$

In Theorem 2.8, we showed that there exists $\bar{t} > 0$ such that

$$f(x^k + t_k d^k) - f(x^k) < -\eta t_k |d^k|^2 \qquad \text{for all } 0 < t_k < \bar{t},$$

provided that for $\beta \in (0, 1)$,

$$\varepsilon < (1 - \beta)|d^k|. \tag{2.11}$$

Set $\varepsilon = \bar{K}\Delta_k$. If equation (2.11) does not hold, then a line search failure will occur, resulting in $\mu_{k+1} = 0.5\mu_k$. Thus, eventually we will have $\mu_k < \frac{(1-\beta)}{\bar{K}}$ and

$$\Delta_k \leq \mu_k |d^k| < \frac{(1 - \beta)}{\bar{K}} |d^k|,$$

which means equation (2.11) will hold. Thus, after a finite number of iterations, the AGS algorithm will declare a line search success and find a new iterate with a lower function value. □

Now we are ready to prove convergence. In the following result, assuming that the step length t_k is bounded away from 0 means that there exists a $\bar{t} > 0$ such that $t_k > \bar{t}$.

Remark 2.10. The following theorem does not ensure the existence of a cluster point. In order to ensure the existence of a cluster point, an additional assumption would have to be made, such as compact level sets.

Theorem 2.11. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $\{x^k\}_{k=0}^{\infty}$ be an infinite sequence generated by the AGS algorithm. Suppose there exists a $\bar{K} > 0$ such that given any set of points generated in Step 1 of the AGS algorithm, the approximate gradient satisfies the error bound $|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k$ for all i. Suppose t_k is bounded away from 0. Then either*

1. *$f(x^k) \downarrow -\infty$, or*
2. *$|d^k| \to 0$, $\Delta_k \downarrow 0$ and every cluster point $\bar{x}$ of the sequence $\{x^k\}_{k=0}^{\infty}$ satisfies $0 \in \partial f(\bar{x})$.*

Proof. If $f(x^k) \downarrow -\infty$, then we are done.
Conversely, if $f(x^k)$ is bounded below, then $f(x^k)$ is non-increasing and bounded below, therefore $f(x^k)$ converges. We consider two cases.

Case 1: An infinite number of line search successes occur.
Let $\bar{x}$ be a cluster point of $\{x^k\}_{k=0}^{\infty}$. Notice that x^k only changes for line search successes, so there exists a subsequence $\{x^{k_j}\}_{j=0}^{\infty}$ of line search successes such that $x^{k_j} \to \bar{x}$. Then for each corresponding step length t_{k_j} and direction d^{k_j}, the following condition holds

$$f(x^{k_j+1}) \leq f(x^{k_j} + t_{k_j} d^{k_j}) < f(x^{k_j}) - \eta t_{k_j} |d^{k_j}|^2.$$

Note that

$$0 \leq \eta t_{k_j} |d^{k_j}|^2 < f(x^{k_j}) - f(x^{k_j+1}).$$

Since $f(x^k)$ converges we know that $f(x^{k_j}) - f(x^{k_j+1}) \to 0$. Since t_{k_j} is bounded away from 0, we see that

$$\lim_{j \to \infty} |d^{k_j}| = 0.$$

Recall from the AGS algorithm, we check the condition

$$\Delta_{k_j} \leq \mu_{k_j} |d^{k_j}|.$$

As $\Delta_{k_j} > 0$, $\mu_{k_j} \leq \mu_0$, and $|d^{k_j}| \to 0$, we can conclude that $\Delta_{k_j} \downarrow 0$.
Finally, from Lemma 2.2(1), as $-d^{k_j} \in G(x^{k_j})$, there exists a $v^{k_j} \in \partial f(x^{k_j})$ such that

$$\begin{aligned} & | - v^{k_j} - d^{k_j}| \leq \bar{K}\Delta_{k_j} \\ \Rightarrow \quad & | - v^{k_j}| - |d^{k_j}| \leq \bar{K}\Delta_{k_j} \\ \Rightarrow \quad & |v^{k_j}| \leq \bar{K}\Delta_{k_j} + |d^{k_j}|, \end{aligned}$$

which implies that

$$0 \leq |v^{k_j}| \leq \bar{K}\Delta_{k_j} + |d^{k_j}| \to 0.$$

So,

$$\lim_{j\to\infty} |v^{k_j}| = 0,$$

where $|v^{k_j}| \geq \text{dist}(0|\partial f(x^{k_j})) \geq 0$, which implies $\text{dist}(0|\partial f(x^{k_j})) \to 0$. We have $x^{k_j} \to \bar{x}$. As f is a finite max function, it is continuous and therefore, by Proposition 1.10, ∂f is outer semicontinuous. Hence, every cluster point $\bar{x}$ of a convergent subsequence of $\{x^k\}_{k=0}^{\infty}$ satisfies $0 \in \partial f(\bar{x})$.

Case 2: A finite number of line search successes occur.
This means there exists a $\bar{k}$ such that $x^k = x^{\bar{k}} = \bar{x}$ for all $k \geq \bar{k}$. However, by Corollary 2.9, if $0 \notin \partial f(\bar{x})$, then after a finite number of iterations, the algorithm will find function value decrease (line search success). Hence, we have $0 \in \partial f(\bar{x})$. □

Our last result shows that if the algorithm terminates in Step 2, then the distance from 0 to the exact subdifferential is controlled by ε_{tol}.

Theorem 2.12. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose there exists a $\bar{K} > 0$ such that for each iteration k, the approximate gradient satisfies $|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k$ for all i. Suppose the AGS algorithm terminates at some iteration $\bar{k}$ in Step 2 for $\varepsilon_{tol} > 0$. Then*

$$\text{dist}(0|\partial f(x^{\bar{k}})) < (1 + \bar{K}\mu_0)\varepsilon_{tol}.$$

Proof. Let $\bar{w} = \text{Proj}(0|G(x^{\bar{k}}))$. We use $\bar{v} \in \partial f(x^{\bar{k}})$ as constructed in Lemma 2.2(1) to see that

$$\begin{aligned}
\text{dist}(0|\partial f(x^{\bar{k}})) &\leq \text{dist}(0|\bar{v}) \\
&\leq \text{dist}(0|\bar{w}) + \text{dist}(\bar{w}|\bar{v}) \quad (\text{as } \bar{w} \in G(x^{\bar{k}}),\ \bar{v} \in \partial f(x^{\bar{k}})) \\
&= |d^{\bar{k}}| + |\bar{w} - \bar{v}| \quad (\text{as } |d^{\bar{k}}| = |\,\text{Proj}(0|G(x^{\bar{k}}))|) \\
&\leq |d^{\bar{k}}| + \varepsilon \quad (\text{by Lemma 2.2}) \\
&< \varepsilon_{tol} + \varepsilon \quad (\text{as } |d^{\bar{k}}| < \varepsilon_{tol}).
\end{aligned}$$

The final statement now follows by the test $\Delta_{\bar{k}} \leq \mu_{\bar{k}}|d^{\bar{k}}|$ in Step 2 and the fact that $\mu_{\bar{k}} \leq \mu_0$ as $\{\mu_k\}_{k=0}$ is a non-increasing sequence. □

Chapter 3

Robust Approximate Gradient Sampling Algorithm

The AGS algorithm depends on the active set of functions at each iterate, $A(x^k)$. Of course, it is possible at various times in the algorithm for there to be functions that are inactive at the current iterate, but active within a small radius of the current iterate. Typically, such behaviour means that the current iterate is close to a 'nondifferentiable ridge' formed by the function. In [BLO02] and [BLO05], it is suggested that allowing an algorithm to take into account these 'almost active' functions will provide a better idea of what is happening at and around the current iterate, thus, making the algorithm more robust, i.e., performs for unusual or strict types of functions.

In this section, we present the robust approximate gradient sampling algorithm (RAGS algorithm). Specifically, we adapt the AGS algorithm by expanding our active set to include all functions that are active at x^k or active at one (or more) of the points in the set $Y = [x^k y^1, \ldots, y^m]$. Recall from the AGS algorithm that the set Y is sampled from within a ball of radius Δ_k. Thus, the points in Y are not far from the current iterate. We define the robust active set next.

Definition 3.1. Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $y^0 = x^k$ be the current iterate and $Y = [y^0, y^1, y^2, \ldots, y^m]$ be a set of randomly sampled points from a ball centered at x^k with radius Δ_k. The **robust active set** of f on Y is

$$A(Y) = \bigcup_{y^j \in Y} A(y^j), \tag{3.1}$$

where each set $A(y^j)$ is defined as in equation (1.2).

To shed some light on the motivation for our robust algorithm, we consider the following function (originally from [CC78], *Problem 2.1 CB2* in the test set [LV00]):

$$
\begin{aligned}
& f = \max\{f_1, f_2, f_3\}, \\
\text{where} \quad & f_1(x, y) = x^2 + y^4, \\
& f_2(x, y) = (2 - x)^2 + (2 - y)^2, \\
& f_3(x, y) = 2\exp(y - x).
\end{aligned}
$$

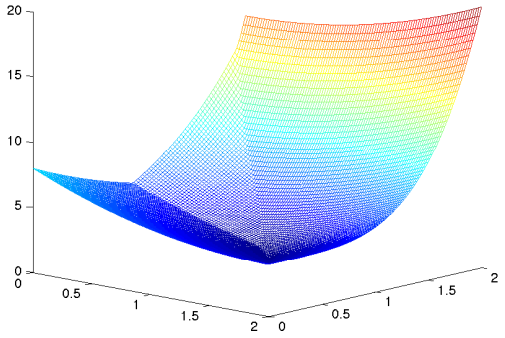
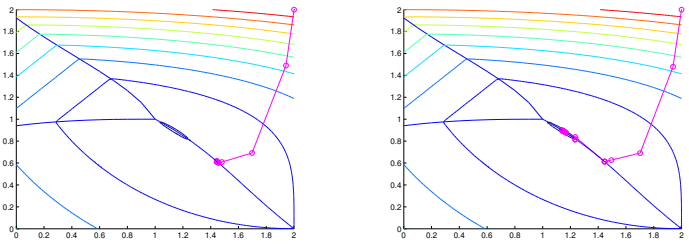


Figure 3.1: Surface plot for *Problem 2.1 CB2*.



(a) AGS algorithm (b) RAGS algorithm

Figure 3.2: Iterations taken using the simplex gradient for *Problem 2.1 CB2*.

These figures illustrate example iterations taken by the AGS algorithm and by the RAGS algorithm (presented in this section) for *Problem 2.1 CB2* using the simplex gradient as an approximate gradient (see Section 4.1 for details used in numerical results).

In Figure 3.2(a), we can see that the iterations move perpendicular to the contours of the functions towards the nondifferentiable ridge. However, eventually the iterations stall at the nondifferentiable ridge. These results

show that the AGS algorithm is performing similar to how we would expect the method of steepest descent to perform on this problem.

In Figure 3.2(b), we can see that the iterations proceed similar to the AGS algorithm until they reach the nondifferentiable ridge. Here, instead of stalling at the nondifferentiable ridge, the algorithm turns and minimizes along the nondifferentiable ridge, moving towards the optimal solution. Thus, by incorporating the 'almost active' functions into our active set, our algorithm is able to further minimize the function $f(x, y)$.

3.1 Algorithm

For the RAGS algorithm, we incorporate the robust active set by replacing Steps 1 and 2 of the AGS algorithm from Section 2.1 with the following.

1. Generate Approximate Subdifferential G_Y^k (Robust):
   Generate a set $Y = [x^k, y^1, \ldots, y^m]$ around the current iterate x^k such that
   $$\max_{j=1,\ldots,m} |y^j - x^k| \leq \Delta_k.$$
   Use Y to calculate the approximate gradient of f_i, denoted $\nabla_A f_i$, at x^k for each $i \in A(Y)$. Then set $G^k = \text{conv}(\nabla_A f_i(x^k))_{i \in A(x^k)}$ and $G_Y^k = \text{conv}(\nabla_A f_i(x^k))_{i \in A(Y)}$.

2. Generate Search Direction:
   Let
   $$d^k = -\operatorname{Proj}(0|G^k).$$
   Let
   $$d_Y^k = -\operatorname{Proj}(0|G_Y^k).$$
   Check if
   $$\Delta_k \leq \mu_k |d^k|. \tag{3.2}$$
   If (3.2) does not hold, then set $x^{k+1} = x^k$, set
   $$\Delta_{k+1} = \begin{cases} \theta\mu_k|d^k| & \text{if } |d^k| \neq 0 \\ \theta\Delta_k & \text{if } |d^k| = 0 \end{cases},$$
   and go to Step 4. If (3.2) holds and $|d^k| < \varepsilon_{tol}$, then STOP. Else, continue to the line search, using d_Y^k as a search direction.

Notice that in Step 2, we still use the stopping conditions from Section 2. Although this modification requires the calculation of two projections, it should be noted that neither of these projections are particularly difficult to calculate and that no additional function evaluations are required for this modification. In Section 3.3, we use the Goldstein approximate subdifferential to adapt Theorem 2.12 to work for stopping conditions based on d_Y^k, but we still do not have theoretical results for the exact subdifferential.

In the numerics section, we test each version of our algorithm using the robust descent direction to check the stopping conditions. For the RAGS algorithm, this alteration shows convincing results that the robust stopping conditions not only guarantee convergence, but significantly decrease the number of function evaluations required for the RAGS algorithm to converge.

3.2 Convergence

To show that the RAGS algorithm is well-defined we require that when Δ is small enough, the robust active set is in fact equal to the original active set.

Lemma 3.2. *Let $f = \max\{f_i : i = 1, \dots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $Y = [\bar{x}, y^1, \dots, y^m]$ be a randomly sampled set from a ball centered at $\bar{x}$ with radius Δ. Then there exists an $\tilde{\varepsilon} > 0$ such that if $Y \subseteq B_{\tilde{\varepsilon}}(\bar{x})$, then $A(\bar{x}) = A(Y)$.*

Proof. Clearly, if $i \in A(\bar{x})$, then $i \in A(Y)$, as $\bar{x} \in Y$.
Consider $i \notin A(\bar{x})$. Then by the definition of f, we have that

$$f_i(\bar{x}) < f(\bar{x}).$$

We know that f is continuous (see Section 1.1). Therefore, there exists an $\tilde{\varepsilon}_i > 0$ such that for all $z \in B_{\tilde{\varepsilon}_i}(\bar{x})$

$$f_i(z) < f(z).$$

If $\Delta < \tilde{\varepsilon}_i$, then we have $|y^j - \bar{x}| < \tilde{\varepsilon}_i$ for all $j = 1, \dots, m$. Therefore,

$$f_i(y^j) < f(y^j) \quad \text{for all } j = 1, \dots, m, \tag{3.3}$$

so $i \notin A(Y)$. Setting $\tilde{\varepsilon} = \min_{i \notin A(\bar{x})} \tilde{\varepsilon}_i$ completes the proof. □

Using Lemma 3.2, we can easily conclude that the AGS algorithm is still well-defined when using the robust active set.

Corollary 3.3. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose $0 \notin \partial f(x^k)$ for each iteration k. Suppose there exists a $\bar{K} > 0$ such that given any set of points generated in Step 1 of the RAGS algorithm, the approximate gradient satisfies $|\nabla_A f_i(x^k) - \nabla f(x^k)| \leq \bar{K}\Delta_k$ for all i. Then after a finite number of iterations, the RAGS algorithm will find function value decrease.*

Proof. Consider x^k, where $0 \notin \partial f(x^k)$.
For eventual contradiction, suppose we do not find function value decrease. In the RAGS algorithm, this corresponds to an infinite number of line search failures. If we have an infinite number of line search failures, then $\Delta_k \to 0$ and $x^{\bar{k}} = x^k$ for all $\bar{k} \geq k$. In Lemma 3.2, $\tilde{\varepsilon}$ depends only on x^k. Hence, we can conclude that eventually $\Delta_k < \tilde{\varepsilon}$ and therefore $Y^k \subseteq B_{\tilde{\varepsilon}}(x^k)$. Thus, eventually $A(x^k) = A(Y^k)$. Once the two active sets are equal, the results of Section 2.2 will hold. Thus, by Corollary 2.9, the algorithm will find a new iterate with a lower function value, a contradiction. □

To examine convergence of the RAGS algorithm we use the result that eventually the robust active set at the current iterate will be a subset of the regular active set at any cluster point of the algorithm.

Lemma 3.4. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $Y^k = [x^k, y^1, \ldots, y^m]$ be a randomly sampled set from a ball centered at x^k with radius Δ_k. Let $x^k \to \bar{x}$. Then there exists an $\tilde{\varepsilon} > 0$ such that if $Y^k \subseteq B_{\tilde{\varepsilon}}(\bar{x})$, then $A(Y^k) \subseteq A(\bar{x})$.*

Proof. Let $i \notin A(\bar{x})$. By definition of f, we have that

$$f_i(\bar{x}) < f(\bar{x}).$$

Since f is continuous, there exists an $\tilde{\varepsilon}_i > 0$ such that for all $z \in B_{\tilde{\varepsilon}_i}(\bar{x})$

$$f_i(z) < f(z).$$

If $Y^k \subseteq B_{\tilde{\varepsilon}_i}(\bar{x})$, then we have $|x^k - \bar{x}| < \tilde{\varepsilon}_i$ and $|y^j - \bar{x}| < \tilde{\varepsilon}_i$ for all $j = 1, \ldots, m$. Therefore

$$f_i(x^k) < f(x^k)$$

and

$$f_i(y^j) < f(y^j) \qquad \text{for all } j = 1, \ldots, m.$$

Therefore, $i \notin A(Y^k)$. Letting $\tilde{\varepsilon} = \min_{i \notin A(\bar{x})} \tilde{\varepsilon}_i$ completes the proof. □

Now we examine the convergence of the RAGS algorithm.

Theorem 3.5. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $\{x^k\}_{k=0}^{\infty}$ be an infinite sequence generated by the RAGS algorithm. Suppose there exists a $\bar{K} > 0$ such that given any set of points generated in Step 1 of the RAGS algorithm, the approximate gradient satisfies the error bound $|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k$ for all i. Suppose t_k is bounded away from 0. Then either*

1. *$f(x^k) \downarrow -\infty$, or*
2. *$|d^k| \to 0$, $\Delta_k \downarrow 0$ and every cluster point $\bar{x}$ of the sequence $\{x^k\}_{k=0}^{\infty}$ satisfies $0 \in \partial f(\bar{x})$.*

Proof. If $f(x^k) \downarrow -\infty$, then we are done.
Conversely, if $f(x^k)$ is bounded below, then $f(x^k)$ is non-increasing and bounded below, therefore $f(x^k)$ converges. We consider two cases.

Case 1: An infinite number of line search successes occur.
Let $\bar{x}$ be a cluster point of $\{x^k\}_{k=0}^{\infty}$. Notice that x^k only changes for line search successes, so there exists a subsequence $\{x^{k_j}\}_{k=0}^{\infty}$ of line search successes such that $x^{k_j} \to \bar{x}$. Following the arguments of Theorem 2.11, we have $|d^{k_j}| \to 0$ and $\Delta_{k_j} \downarrow 0$. Notice that if $\Delta_{k_j} \downarrow 0$, then eventually $Y^{k_j} \subseteq B_{\tilde{\varepsilon}}(\bar{x})$, where $x^k \to \bar{x}$ and $\tilde{\varepsilon}$ is defined as in Lemma 3.4. Thus, by Lemma 3.4, we have that $A(Y^{k_j}) \subseteq A(\bar{x})$. This means $G_{Y^{k_j}}(x^{k_j})$ is formed from a subset of the active gradients that make up $\partial f(\bar{x})$. Thus, by Lemma 2.2(1), as $-d^{k_j} \in G_{Y^{k_j}}(x^{k_j})$, we can construct a $v^{k_j} \in \partial f(\bar{x})$ from the same set of active gradients that make up $G_{Y^{k_j}}(x^{k_j})$ such that

$$
\begin{aligned}
& |-d^{k_j} - v^{k_j}| \\
= \; & \left| \sum_{i \in A(Y^{k_j})} \alpha_i \nabla_A f_i(x^{k_j}) - \sum_{i \in A(Y^{k_j})} \alpha_i \nabla f_i(\bar{x}) \right| \\
\leq \; & \sum_{i \in A(Y^{k_j})} \alpha_i |\nabla_A f_i(x^{k_j}) - \nabla f_i(\bar{x})| \\
= \; & |\nabla_A f_i(x^{k_j}) - \nabla f_i(\bar{x})| \qquad (\text{as } \textstyle\sum_{i \in A(Y^{k_j})} \alpha_i = 1) \\
\leq \; & |\nabla_A f_i(x^{k_j}) - \nabla f_i(x^{k_j})| + |\nabla f_i(x^{k_j}) - \nabla f_i(\bar{x})| \\
\leq \; & \bar{K}\Delta_{k_j} + |\nabla f_i(x^{k_j}) - \nabla f_i(\bar{x})| \qquad (\text{by error bound}).
\end{aligned}
$$

Using the Triangle Inequality, we have

$$| - d^{k_j} - v^{k_j}| = | - v^{k_j} - d^{k_j}| \geq |v^{k_j}| - |d^{k_j}|,$$

which implies that

$$|v^{k_j}| - |d^{k_j}| \leq \bar{K}\Delta_{k_j} + |\nabla f_i(x^{k_j}) - \nabla f_i(\bar{x})|.$$

We already showed that $|d^{k_j}| \to 0$ and $\Delta_{k_j} \downarrow 0$. Furthermore, since $\nabla f_i \in \mathcal{C}$ and $x^{k_j} \to \bar{x}$, we have $|\nabla f_i(x^{k_j}) - \nabla f_i(\bar{x})| \to 0$. So,

$$\lim_{j\to\infty} |v^{k_j}| = 0.$$

Using the same arguments as in Theorem 2.11, the result follows.

Case 2: A finite number of line search successes occur.
This means there exists a $\bar{k}$ such that $x^k = x^{\bar{k}} = \bar{x}$ for all $k \geq \bar{k}$. However, by Corollary 3.3, if $0 \notin \partial f(\bar{x})$, then after a finite number of iterations, the algorithm will find function value decrease (line search success). Hence, we have $0 \in \partial f(\bar{x})$. □

Remark 3.6. Using d^k to check our stopping conditions allows the result of Theorem 2.12 to still hold.

3.2.1 Descent Direction: d_Y

It is worth noting that we can prove Lemma 2.2(2) for $G(Y)$ and thus, prove d_Y is a descent direction without the requirement that $A(Y) \subseteq A(\bar{x})$. We show this in Lemmas 3.7 and 3.8.

Lemma 3.7. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $Y = [y^0, y^1, y^2, \ldots, y^m]$ be a randomly sampled set from a ball centered at $y^0 = \bar{x}$ with radius Δ. Suppose there exists a $\bar{K} > 0$ such that the approximate gradient satisfies $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \bar{K}\Delta$ for all i. Then for all $v \in \partial f(\bar{x})$, there exists a $w \in G(Y)$ such that*

$$|w - v| \leq \bar{K}\Delta.$$

Proof. By definition, for all $v \in \partial f(\bar{x})$ there exists a set of α_i such that

$$v = \sum_{i\in A(\bar{x})} \alpha_i \nabla f_i(\bar{x}) \qquad \text{where } \alpha_i \geq 0, \sum_{i\in A(\bar{x})} \alpha_i = 1.$$

We know that $A(\bar{x}) \subseteq A(Y)$, as $\bar{x} \in Y$. Thus, using the same α_i as above, we see that

$$w = \sum_{i \in A(\bar{x})} \alpha_i \nabla_A f_i(\bar{x}) \in G(Y).$$

Using the same argument as shown in Lemma 2.2(2), we can conclude that for all $v \in \partial f(\bar{x})$, there exists a $w \in G(Y)$ such that

$$|w - v| \leq \bar{K}\Delta.$$

□

Using this result, we can show that d_Y is a descent direction.

Lemma 3.8. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Let $Y = [y^0, y^1, y^2, \ldots, y^m]$ be a randomly sampled set from a ball centered at $y^0 = \bar{x}$ with radius Δ. Suppose there exists a $\bar{K} > 0$ such that the approximate gradient satisfies $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \bar{K}\Delta$ for all i. Define $d_Y = -\operatorname{Proj}(0|G(Y))$ and suppose $|d_Y| \neq 0$. If $\bar{K}\Delta < |d_Y|$, then $\langle d_Y, v\rangle < 0$ for all $v \in \partial f(\bar{x})$.*

Proof. Using Lemma 3.7 instead of Lemma 2.2(2) in the proof of Corollary 2.5, we can prove the result. □

3.3 Robust Stopping Conditions

We want to provide some insight as to how Theorem 2.12 can work for stopping conditions based on d_Y^k, that is, replacing the stopping conditions $\Delta_k \leq \mu_k |d^k|$ and $|d^k| < \varepsilon_{tol}$ in Step 2 with the robust stopping conditions

$$\Delta_k \leq \mu_k |d_Y^k| \text{ and } |d_Y^k| < \varepsilon_{tol}. \tag{3.4}$$

The following proposition does not theoretically justify why the robust stopping conditions in (3.4) are sufficient, but does help explain their logic. Theoretically, since we do not know what $\bar{x}$ is, we cannot tell when $A(Y^k) \subseteq A(\bar{x})$.

Proposition 3.9. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^1$. Suppose there exists a $\bar{K} > 0$ such that $|\nabla_A f_i(x) - \nabla f_i(x)| \leq \bar{K}\Delta_k$ for all i and for all $x \in B_{\Delta_k}(x^k)$. Suppose the RAGS algorithm terminates at some iteration $\bar{k}$ in Step 2 using the robust stopping conditions given in (3.4). Furthermore, suppose there exists $\bar{x} \in B_{\Delta_{\bar{k}}}(x^{\bar{k}})$ such that $A(Y^{\bar{k}}) \subseteq A(\bar{x})$. Then*

$$\operatorname{Proj}(0|\partial f(\bar{x})) < (1 + \bar{K}\mu_0)\varepsilon_{tol}.$$

Proof. If $A(Y^{\bar{k}}) \subseteq A(\bar{x})$, then the proofs of Lemma 2.2(1) and Theorem 2.12 still hold. □

Additionally, in the following results, we approach the theory for robust stopping conditions using the Goldstein approximate subdifferential. If the RAGS algorithm terminates in Step 2, then it is shown that the distance between 0 and the Goldstein approximate subdifferential is controlled by ε_{tol}. Again, this does not prove the robust stopping conditions are sufficient for the exact subdifferential.

Definition 3.10. The *Goldstein approximate subdifferential*, as defined in [Gol77], is given by the set

$$\partial^{\mathcal{G}}_{\Delta} f(\bar{x}) = \operatorname{conv}(\partial f(z) : z \in B_{\Delta}(\bar{x})), \tag{3.5}$$

where $B_{\Delta}(\bar{x})$ is the closed ball centered at $\bar{x}$ with radius Δ.

We now show that the Goldstein approximate subdifferential contains all of the gradients of the active functions in the robust active set.

Lemma 3.11. *Let $f = \max\{f_i : i = 1, \ldots, N\}$. Let $Y = [y^0, y^1, y^2, \ldots, y^m]$ be a randomly sampled set from a ball centered at $y^0 = \bar{x}$ with radius Δ. If $f_i \in \mathcal{C}^1$ for each i, then*

$$\partial^{\mathcal{G}}_{\Delta} f(\bar{x}) \supseteq \operatorname{conv}(\nabla f_i(y^j) : y^j \in Y, i \in A(y^j)).$$

Proof. If $f_i \in \mathcal{C}^1$ for each $i \in A(Y)$, then by Proposition 1.6, for each $y^j \in Y$ we have

$$\partial f(y^j) = \operatorname{conv}(\nabla f_i(y^j))_{i \in A(y^j)} = \operatorname{conv}(\nabla f_i(y^j) : i \in A(y^j)).$$

Using this in our definition of the Goldstein approximate subdifferential in (3.5) and knowing $B_{\Delta}(\bar{x}) \supseteq Y$, we have

$$\partial^{\mathcal{G}}_{\Delta} f(\bar{x}) \supseteq \operatorname{conv}(\operatorname{conv}(\nabla f_i(y^j) : i \in A(y^j)) : y^j \in Y),$$

which simplifies to

$$\partial^{\mathcal{G}}_{\Delta} f(\bar{x}) \supseteq \operatorname{conv}(\nabla f_i(y^j) : y^j \in Y, i \in A(y^j)). \tag{3.6}$$

□

Now we have a result similar to Lemma 2.2(1) for d^k_Y with respect to the Goldstein approximate subdifferential.

Remark 3.12. For the following two results, we assume each of the $f_i \in \mathcal{C}^{1+}$ with Lipschitz constant L. Note that this implies the Lipschitz constant L is independent of i. If each $f_i \in \mathcal{C}^{1+}$ has Lipschitz constant L_i for ∇f_i, then L is easily obtained by $L = \max\{L_i : i = 1, \ldots, N\}$.

Lemma 3.13. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^{1+}$ with Lipschitz constant L. Let $Y = [y^0, y^1, y^2, \ldots, y^m]$ be a randomly sampled set from a ball centered at $y^0 = \bar{x}$ with radius Δ. Suppose there exists a $\bar{K} > 0$ such that the approximate gradient satisfies $|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| \leq \bar{K}\Delta$ for all i. Then for all $w \in G(Y)$, there exists a $g \in \partial_\Delta^{\mathcal{G}} f(\bar{x})$ such that*

$$|w - g| \leq (\bar{K} + L)\Delta.$$

Proof. By definition, for all $w \in G(Y)$ there exists a set of α_i such that

$$w = \sum_{i \in A(Y)} \alpha_i \nabla_A f_i(\bar{x}), \qquad \text{where } \alpha_i \geq 0, \sum_{i \in A(Y)} \alpha_i = 1.$$

By our assumption that each $f_i \in \mathcal{C}^{1+}$, Lemma 3.11 holds. It is clear that for each $i \in A(Y)$, $i \in A(y^j)$ for some $y^j \in Y$. Let j_i be an index corresponding to this active index; i.e., $i \in A(y^{j_i})$. Thus, for each $i \in A(Y)$, there is a corresponding active gradient

$$\nabla f_i(y^{j_i}) \in \operatorname{conv}(\nabla f_i(y^{j_i}) : y^{j_i} \in Y, i \in A(y^{j_i})) \subseteq \partial_\Delta^{\mathcal{G}} f(\bar{x}).$$

Using the same α_i as above, we can construct

$$g = \sum_{i \in A(Y)} \alpha_i \nabla f_i(y^{j_i}) \in \operatorname{conv}(\nabla f_i(y^{j_i}) : y^{j_i} \in Y, i \in A(y^{j_i})) \subseteq \partial_\Delta^{\mathcal{G}} f(\bar{x}).$$

Then

$$\begin{aligned}
& |w - g| \\
= \; & |\sum_{i \in A(Y)} \alpha_i \nabla_A f_i(\bar{x}) - \sum_{i \in A(Y)} \alpha_i \nabla f_i(y^{j_i})| \\
\leq \; & \sum_{i \in A(Y)} \alpha_i |\nabla_A f_i(\bar{x}) - \nabla f_i(y^{j_i})| \\
= \; & \sum_{i \in A(Y)} \alpha_i |\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x}) + \nabla f_i(\bar{x}) - \nabla f_i(y^{j_i})| \\
\leq \; & \sum_{i \in A(Y)} \alpha_i \Big(|\nabla_A f_i(\bar{x}) - \nabla f_i(\bar{x})| + |\nabla f_i(\bar{x}) - \nabla f_i(y^{j_i})| \Big).
\end{aligned}$$

Applying our error bound assumption and the Lipschitz condition, we have

$$\begin{aligned} |w-g| &\leq \sum_{i\in A(Y)} \alpha_i \left(\bar{K}\Delta + L \max_{j_i} |\bar{x} - y^{j_i}| \right) \\ &\leq \bar{K}\Delta + L\Delta \qquad (\text{as } \sum_{i\in A(Y)} \alpha_i = 1) \\ &= (\bar{K}+L)\Delta. \end{aligned}$$

Hence, for all $w \in G(\bar{x})$, there exists a $g \in \partial^{\mathcal{G}}_{\Delta} f(\bar{x})$ such that

$$|w-g| \leq \varepsilon + L\Delta. \tag{3.7}$$

□

Thus, using Lemma 3.13, we can show that if the algorithm stops due to the robust stopping conditions, then the distance from 0 to the Goldstein approximate subdifferential is controlled by ε_{tol}.

Proposition 3.14. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^{1+}$ with Lipschitz constant L. Suppose there exists a $\bar{K} > 0$ such that for each iteration k, the approximate gradient satisfies $|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k$ for all i. Suppose the RAGS algorithm terminates at some iteration $\bar{k}$ in Step 2 using the robust stopping conditions given in* (3.4)*. Then*

$$\operatorname{Proj}(0|\partial^{G}_{\Delta_{\bar{k}}} f(x^{\bar{k}})) < [1 + \mu_0(\bar{K}+L)]\varepsilon_{tol}.$$

Proof. Let $\bar{w} = \operatorname{Proj}(0|G(Y^{\bar{k}}))$. We use $\bar{g} \in \partial^{\mathcal{G}}_{\Delta_{\bar{k}}} f(x^{\bar{k}})$ as constructed in Lemma 3.13 to see that

$$\begin{aligned} \operatorname{dist}(0|\partial^{\mathcal{G}}_{\Delta_{\bar{k}}) f(x^{\bar{k}})} &\leq \operatorname{dist}(0|\bar{g}) \\ &\leq \operatorname{dist}(0|\bar{w}) + \operatorname{dist}(\bar{w}|\bar{g}) \quad (\text{as } \bar{w} \in G(Y^{\bar{k}}),\ \bar{g} \in \partial^{\mathcal{G}}_{\Delta_{\bar{k}}} f(x^{\bar{k}})) \\ &= |d^{\bar{k}}_Y| + |\bar{w} - \bar{g}| \qquad (\text{as } |d^{\bar{k}}_Y| = |\operatorname{Proj}(0|G(Y^{\bar{k}}))|) \\ &\leq |d^{\bar{k}}_Y| + (\bar{K}+L)\Delta_{\bar{k}} \qquad (\text{by Lemma 3.13}) \\ &< \varepsilon_{tol} + (\bar{K}+L)\Delta_{\bar{k}} \qquad (\text{as } |d^{\bar{k}}_Y| < \varepsilon_{tol}). \end{aligned}$$

The statement now follows by the test $\Delta_{\bar{k}} \leq \mu_{\bar{k}} |d^{\bar{k}}_Y|$ in Step 2 and the fact that $\mu_{\bar{k}} \leq \mu_0$ as $\{\mu_k\}_{k=0}$ is a non-increasing sequence. □

Chapter 4

Approximate Gradients

As seen in the previous two sections, in order for convergence to be guaranteed in the AGS and RAGS algorithms, the approximate gradient used must satisfy an **error bound** for each of the active f_i. Specifically, for a randomly sampled set of points $Y = [x^k, y^1, \ldots, y^m]$ centered around x^k, there must exist a $\bar{K} > 0$ such that

$$|\nabla_A f_i(x^k) - \nabla f_i(x^k)| \leq \bar{K}\Delta_k,$$

where $\Delta_k = \max_j |y^j - x^k|$. In this section, we present three specific approximate gradients that satisfy the above requirement: the simplex gradient, the centered simplex gradient and the Gupal estimate of the gradient of the Steklov averaged function.

4.1 Simplex Gradient

The simplex gradient is a commonly used approximate gradient. In recent years, several derivative-free algorithms have been proposed that use the simplex gradient, ([BK98], [Kel99a], [CV07], [CJV08], and [HM11]) among others. It is geometrically defined as the gradient of the linear interpolation of f over a set of $n + 1$ points in $\mathbb{R}^n$. In the following section, we mathematically define the simplex gradient.

4.1.1 Definitions

First we present several basic definitions of terms used in the definition of the simplex gradient.

Definition 4.1. A set S is an **affine set** if given any two points $x_1 \in S$ and $x_2 \in S$ with $x_1 \neq x_2$, the line formed by x_1 and x_2 is a subset of S, i.e., $\overline{x_1 x_2} \subseteq S$.

Definition 4.2. The **affine hull** of a set $S \subseteq \mathbb{R}^n$ is the smallest affine set containing S.

Definition 4.3. A set of $m+1$ points $Y = [y^0, y^1, \ldots, y^m]$ is said to be **affinely independent** if its affine hull $\text{aff}(y^0, y^1, \ldots, y^m)$ has dimension m.

Equivalently, Y is affinely independent if the set $[y^1 - y^0, \ldots, y^m - y^0]$ is linearly independent, [CSV09].

Definition 4.4. Let $Y = [y^0, y^1, \ldots, y^n]$ be a set of affinely independent points in $\mathbb{R}^n$. We say that Y forms the **simplex**, S, where $S = \text{conv}(Y)$. Thus, S is a simplex if it can be written as the convex hull of an affinely independent set of $n+1$ points in $\mathbb{R}^n$.

With the previous terms defined, we are now ready to mathematically define the simplex gradient.

Definition 4.5. Let $Y = [y^0, y^1, \ldots, y^n]$ be a list of affinely independent points in $\mathbb{R}^n$. The **simplex gradient** of a function f over the set Y is given by

$$\nabla_s f(Y) = L^{-1}\delta f(Y),$$

where

$$L = \begin{bmatrix} y^1 - y^0 \cdots y^n - y^0 \end{bmatrix}^\top$$

and

$$\delta f(Y) = \begin{bmatrix} f(y^1) - f(y^0) \\ \vdots \\ f(y^n) - f(y^0) \end{bmatrix}.$$

The **condition number** of a simplex is given by $\|\hat{L}^{-1}\|$, where

$$\hat{L} = \frac{1}{\Delta}[y^1 - y^0 \;\; y^2 - y^0 \;\; \ldots \;\; y^n - y^0]^\top \text{ and } \Delta = \max_{j=1,\ldots n} |y^j - y^0|.$$

4.1.2 Convergence

The following result (from Kelley) shows that there exists an appropriate error bound between the simplex gradient and the exact gradient of our objective function. We note that the Lipschitz constant used in the following theorem corresponds to ∇f_i.

Theorem 4.6. *Consider $f_i \in \mathcal{C}^{1+}$ with Lipschitz constant K_i for ∇f_i. Let $Y = [y^0, y^1, \ldots, y^n]$ form a simplex. Let*

$$\hat{L} = \frac{1}{\Delta}[y^1 - y^0 \;\; y^2 - y^0 \;\; \ldots \;\; y^n - y^0]^\top, \quad \textit{where } \Delta = \max_{j=1,\ldots n} |y^j - y^0|.$$

Then the simplex gradient satisfies the error bound

$$|\nabla_s f_i(Y) - \nabla f_i(y^0)| \leq \bar{K}\Delta,$$

where $\bar{K} = \frac{1}{2} K_i \sqrt{n} \|\hat{L}^{-1}\|$.

Proof. See [Kel99b, Lemma 6.2.1]. □

With the above error bound result, we conclude that convergence holds when using the simplex gradient as an approximate gradient in the AGS and RAGS algorithms.

Corollary 4.7. *Let* $f = \max\{f_i : i = 1, \ldots, N\}$ *where each* $f_i \in \mathcal{C}^{1+}$ *with Lipschitz constant* K_i *for* ∇f_i. *If the approximate gradient used in the AGS or RAGS algorithm is the simplex gradient and* $\|\hat{L}^{-1}\|$ *is bounded above, then the results of Theorems 2.6, 2.8, 2.11 and 2.12 hold.*

4.1.3 Algorithm

Algorithm - Simplex Gradient

In order to calculate a simplex gradient in Step 1, we generate a set $Y = [x^k, y^1, \ldots, y^n]$ of points in $\mathbb{R}^n$ and then check to see if Y forms a well-poised simplex by calculating its condition number, $\|\hat{L}^{-1}\|$. A bounded condition number ($\|\hat{L}^{-1}\| < n$) ensures a 'good' error bound between the approximate gradient and the exact gradient.

If Y forms a well-poised simplex ($\|\hat{L}^{-1}\| < n$), then we calculate the simplex gradient of f_i over Y for each $i \in A(x^k)$ (or each $i \in A(Y^k)$) and then set the approximate subdifferential equal to the convex hull of the active simplex gradients. If the set Y does not form a well-poised simplex ($\|\hat{L}^{-1}\| \geq n$), then we resample. We note that the probability of generating a random matrix with a condition number greater than n is asymptotically constant, [Wsc04]. Thus, randomly generating simplices is a quick and practical option. Furthermore, notice that calculating the condition number does not require function evaluations; thus, resampling does not affect the number of function evaluations required by the algorithm.

4.2 Centered Simplex Gradient

The centered simplex gradient is the average of two simplex gradients. Although it requires more function evaluations, it contains an advantage that the error bound satisfied by the centered simplex gradient is in terms of Δ^2, rather than Δ.

4.2.1 Definitions

Definition 4.8. Let $Y = [y^0, y^1, \ldots, y^n]$ form a simplex. We define the sets

$$Y^+ = [x, x + \tilde{y}^1, \ldots, x + \tilde{y}^n]$$

and

$$Y^- = [x, x - \tilde{y}^1, \ldots, x - \tilde{y}^n],$$

where $x = y^0$ and $\tilde{y}^i = y^i - y^0$ for $i = 1, \ldots n$. The **centered simplex gradient** is the average of the two simplex gradients over the sets Y^+ and Y^-, i.e.,

$$\nabla_{CS} f(Y^+) = \frac{1}{2}(\nabla_S f(Y^+) + \nabla_S f(Y^-)).$$

4.2.2 Convergence

To show that the AGS and RAGS algorithms are well-defined when using the centered simplex gradient as an approximate gradient, we provide an error bound between the centered simplex gradient and the exact gradient (again from Kelley).

Theorem 4.9. *Consider $f_i \in \mathcal{C}^{2+}$ with Lipschitz constant K_i for $\nabla^2 f_i$. Let $Y = [y^0, y^1, \ldots, y^n]$ form a simplex. Let*

$$\hat{L} = \frac{1}{\Delta}[y^1 - y^0, \ldots, y^n - y^0] \quad \text{where } \Delta = \max_{j=1,\ldots n} |y^j - y^0|.$$

Then the centered simplex gradient satisfies the error bound

$$|\nabla_{CS} f_i(Y) - \nabla f_i(y^0)| \leq \bar{K}\Delta^2,$$

where $\bar{K} = K_i\sqrt{n}\|\hat{L}^{-1}\|$.

Proof. See [Kel99b, Lemma 6.2.5]. □

Notice that Theorem 4.9 requires $f_i \in \mathcal{C}^{2+}$. If $f_i \in \mathcal{C}^{1+}$, then the error bound is in terms of Δ, not Δ^2. With the above error bound result, we conclude that convergence holds when using the centered simplex gradient as an approximate gradient in the AGS and RAGS algorithms.

Corollary 4.10. *Let $f = \max\{f_i : i = 1, \ldots, N\}$ where each $f_i \in \mathcal{C}^{2+}$ with Lipschitz constant K_i for $\nabla^2 f_i$. If the approximate gradient used in the AGS or RAGS algorithm is the centered simplex gradient and $\Delta_0 \leq 1$, then the results of Theorems 2.6, 2.8, 2.11 and 2.12 hold.*

Proof. Since $\Delta_0 \leq 1$ and non-increasing, $(\Delta_k)^2 \leq \Delta_k$ and ergo, Theorems 2.6 and 2.11 hold. □

4.2.3 Algorithm

To adapt the AGS and RAGS algorithms to use the centered simplex gradient, in Step 1 we sample our set Y in the same manner as for the simplex gradient (resampling until a well-poised set is achieved). We then form the sets Y^+ and Y^- and proceed as expected.

4.3 Gupal Estimate of the Gradient of the Steklov Averaged Function

The nonderivative version of the gradient sampling algorithm presented by Kiwiel in [Kiw10] uses the Gupal estimate of the gradient of the Steklov averaged function as an approximate gradient. We see in Theorem 4.16 that an appropriate error bound exists for this approximate gradient. Surprisingly, unlike the error bounds for the simplex and centered simplex gradients, the error bound in Theorem 4.16 does not include a condition number term. Following the notation used by Kiwiel in [Kiw10], we define the Gupal estimate of the gradient of the Steklov averaged function as follows.

4.3.1 Definitions

Definition 4.11. For $\alpha > 0$, the **Steklov averaged** function f_α is defined by

$$f_\alpha(x) = \int_{\mathbb{R}^n} f(x-y)\psi_\alpha(y)dy,$$

where $\psi_\alpha : \mathbb{R}^n \to \mathbb{R}_+$ is the **Steklov mollifier** defined by

$$\psi_\alpha(y) = \begin{cases} 1/\alpha^n & \text{if } y \in [-\alpha/2, \alpha/2]^n, \\ 0 & \text{otherwise.} \end{cases}$$

We can equivalently define the Steklov averaged function by

$$f_\alpha(x) = \frac{1}{\alpha^n} \int_{x_1-\alpha/2}^{x_1+\alpha/2} \cdots \int_{x_n-\alpha/2}^{x_n+\alpha/2} f(y)dy_1 \ldots dy_n. \tag{4.1}$$

The partial derivatives of f_α are given by

$$\frac{\partial f_\alpha}{\partial x_i}(x) = \int_{\mathbb{B}_\infty} \gamma_i(x, \alpha, \zeta)d\zeta \tag{4.2}$$

for $i = 1, \ldots, n$, where $\mathbb{B}_\infty = [-1/2, 1/2]^n$ is the unit cube centred at 0 and

$$\begin{aligned}&\gamma_i(x, \alpha, \zeta) = \\ &\frac{1}{\alpha}\Bigg[f(x_1 + \alpha\zeta_1, \ldots, x_{i-1} + \alpha\zeta_{i-1}, x_i \ + \ \frac{1}{2}\alpha, x_{i+1} + \alpha\zeta_{i+1}, \ldots, x_n + \alpha\zeta_n) \\ &-f(x_1 + \alpha\zeta_1, \ldots, x_{i-1} + \alpha\zeta_{i-1}, x_i \ - \ \frac{1}{2}\alpha, x_{i+1} + \alpha\zeta_{i+1}, \ldots, x_n + \alpha\zeta_n)\Bigg].\end{aligned} \tag{4.3}$$

Definition 4.12. Given $\alpha > 0$ and $z = (\zeta^1, \ldots, \zeta^n) \in \prod_{i=1}^n \mathbb{B}_\infty$, the **Gupal estimate** of $\nabla f_\alpha(x)$ over z is given by

$$\gamma(x, \alpha, z) = (\gamma_1(x, \alpha, \zeta^1), \ldots, \gamma_n(x, \alpha, \zeta^n)). \tag{4.4}$$

Remark 4.13. Although we define $\gamma(x, \alpha, z)$ as the Gupal estimate of $\nabla f_\alpha(x)$, in Section 4.3.2, we will show that $\gamma(x, \alpha, z)$ provides a good approximate of the exact gradient, $\nabla f_i(x)$.

Remark 4.14. For the following results, we note that the α used in the above definitions is equivalent to our search radius Δ. Thus, we will be replacing α with Δ in the convergence results in Section 4.3.2.

4.3.2 Convergence

Convergence

As before, in order to show that the AGS and RAGS algorithms are well-defined when using the Gupal estimate as an approximate gradient, we must establish that it provides a good approximate of our exact gradient. To do this, we need the following result.

Lemma 4.15. *Let $f \in \mathcal{C}^{1+}$ with Lipschitz constant K for ∇f. Let $y^0 \in \mathbb{R}^n$. Then for any $y \in \mathbb{R}^n$*

$$|f(y) - f(y^0) - \nabla f(y^0)^\top (y - y^0)| \leq \frac{1}{2} K |y - y^0|^2.$$

Proof. For ease of notation, let $\delta = y - y^0$. By the Fundamental Theorem of Calculus, for any $y \in \mathbb{R}^n$ we have

$$f(y) - f(y^0) = \int_0^1 \nabla f(y^0 + \tau(\delta))^\top \delta \, d\tau. \tag{4.5}$$

Considering $\nabla f(y^0)^\top \delta$, notice that

$$\nabla f(y^0)^\top \delta = \int_0^1 \nabla f(y^0)^\top \delta \; d\tau. \tag{4.6}$$

Subtracting (4.6) from (4.5), we have

$$f(y) - f(y^0) - \nabla f(y^0)^\top \delta = \int_0^1 (\nabla f(y^0 + \tau\delta) - \nabla f(y^0))^\top \delta \; d\tau.$$

Taking the absolute value, we have

$$\begin{aligned}
& \left| f(y) - f(y^0) - \nabla f(y^0)^\top \delta \right| \\
= \; & \left| \int_0^1 (\nabla f(y^0 + \tau\delta) - \nabla f(y^0))^\top \delta \; d\tau \right| \\
\leq \; & \int_0^1 \left| (\nabla f(y^0 + \tau\delta) - \nabla f(y^0))^\top \delta \right| d\tau \qquad \text{(as } \nabla f \text{ is cont.)} \\
\leq \; & \int_0^1 \left| (\nabla f(y^0 + \tau\delta) - \nabla f(y^0)) \right| |\delta| \; d\tau \quad \text{(by Cauchy Schwarz)} \\
\leq \; & \int_0^1 K|y^0 + \tau\delta - y^0||\delta| d\tau \qquad \text{(as } \nabla f \text{ is Lip.)} \\
= \; & \int_0^1 K\tau|\delta|^2 d\tau \qquad \text{(as } \tau \in (0,1)) \\
= \; & \frac{1}{2} K|\delta|^2.
\end{aligned}$$

Therefore, with $\delta = y - y^0$, we have

$$|f(y) - f(y^0) - \nabla f(y^0)^\top (y - y^0)| \leq \frac{1}{2} K|y - y^0|^2.$$

□

Using Lemma 4.15, we establish an error bound between the Gupal estimate and the exact gradient of f_i.

Theorem 4.16. *Consider $f_i \in \mathcal{C}^{1+}$ with Lipschitz constant K_i for ∇f_i. Let $\varepsilon > 0$. Then for $\Delta > 0$, $z = (\zeta^1, \ldots, \zeta^n) \in Z = \prod_{i=1}^n \mathbb{B}_\infty$ and any point $x \in \mathbb{R}^n$, the Gupal estimate of $\nabla f_{i,\alpha}(x)$ satisfies the error bound*

$$|\gamma(x, \Delta, z) - \nabla f_i(x)| \leq \sqrt{n}\frac{1}{2}K_i\Delta(\sqrt{n} + 3).$$

Proof. For $\Delta > 0$, let

$$y^{j-} = [x_1 + \Delta\zeta_1, \ldots, x_{j-1} + \Delta\zeta_{j-1}, x_j - \frac{1}{2}\Delta, x_{j+1} + \Delta\zeta_{j+1}, \ldots, x_n + \Delta\zeta_n]^\top.$$

By Lemma 4.15, for

$$y^{j+} = [x_1 + \Delta\zeta_1, \ldots, x_{j-1} + \Delta\zeta_{j-1}, x_j + \frac{1}{2}\Delta, x_{j+1} + \Delta\zeta_{j+1}, \ldots, x_n + \Delta\zeta_n]^\top,$$

we have

$$|f_i(y^{j+}) - f_i(y^{j-}) - \nabla f_i(y^{j-})^\top(y^{j+} - y^{j-})| \leq \frac{1}{2}K_i|y^{j+} - y^{j-}|^2. \tag{4.7}$$

From equation (4.3) (with $\alpha = \Delta$), we can see that

$$f_i(y^{j+}) - f_i(y^{j-}) = \Delta\ \gamma_j(x, \Delta, \zeta^j).$$

Hence, equation (4.7) becomes

$$|\Delta\ \gamma_j(x, \Delta, \zeta^j) - \nabla f_i(y^{j-})^\top(y^{j+} - y^{j-})| \leq \frac{1}{2}K_i|y^{j+} - y^{j-}|^2. \tag{4.8}$$

From our definitions of y^{j-} and y^{j+}, we can see that

$$y^{j+} - y^{j-} = [0, \ldots, 0, \Delta, 0, \ldots, 0]^\top.$$

The inner product in equation (4.8) simplifies to

$$\nabla f_i(y^{j-})^\top(y^{j+} - y^{j-}) = \Delta\ \frac{\partial f_i}{\partial x_j}(y^{j-}).$$

Thus, we have

$$\left|\Delta\ \gamma_j(x, \Delta, \zeta^j) - \Delta\ \frac{\partial f_i}{\partial x_j}(y^{j-})\right| \leq \frac{1}{2}K_i\Delta^2.$$

Dividing out Δ gives

$$\left|\gamma_j(x, \Delta, \zeta^j) - \frac{\partial f_i}{\partial x_j}(y^{j-})\right| \leq \frac{1}{2}K_i\Delta. \tag{4.9}$$

We notice that

$$\begin{aligned} |y^{j-} - x| &= \left|(\Delta\zeta_1^j, \ldots, \Delta\zeta_{j-1}^j, -\frac{1}{2}\Delta, \Delta\zeta_{j+1}^j, \ldots, \Delta\zeta_n^j)\right| \\ &= \Delta\left|(\zeta_1^j, \ldots, \zeta_{j-1}^j, -\frac{1}{2}, \zeta_{j+1}^j, \ldots, \zeta_n^j)\right|. \end{aligned}$$

Using the standard basis vector e^j, we have

$$\begin{aligned} |y^{j-} - x| &= \Delta\left|\zeta^j - \zeta_j e^j - \frac{1}{2}e^j\right| \\ &\leq \Delta\left(|\zeta^j| + |\zeta_j| + \frac{1}{2}\right) \\ &\leq \Delta\left(\frac{1}{2}\sqrt{n} + \frac{1}{2} + \frac{1}{2}\right) \quad (\text{as } \zeta^j \in [-1/2, 1/2]^n) \\ &= \frac{1}{2}\Delta(\sqrt{n} + 2). \end{aligned}$$

Thus, since $f_i \in \mathcal{C}^{1+}$, we have

$$|\nabla f_i(y^{j-}) - \nabla f_i(x)| \leq K_i \frac{1}{2}\Delta(\sqrt{n} + 2). \tag{4.10}$$

Now, we notice that

$$\begin{aligned} &\left|\frac{\partial f_i}{\partial x_j}(y^{j-}) - \frac{\partial f_i}{\partial x_j}(x)\right| \\ = &\left|\nabla f_i(y^{j-})^\top e^j - \nabla f_i(x)^\top e^j\right| \\ \leq &|\nabla f_i(y^{j-}) - \nabla f_i(x)||e^j| \\ = &|\nabla f_i(y^{j-}) - \nabla f_i(x)|. \end{aligned}$$

Therefore,

$$\left|\frac{\partial f_i}{\partial x_j}(y^{j-}) - \frac{\partial f_i}{\partial x_j}(x)\right| \leq K_i \frac{1}{2}\Delta(\sqrt{n} + 2). \tag{4.11}$$

Using equations (4.9) and (4.11), we have

$$\begin{aligned}
& \left|\gamma_j(x,\Delta,\zeta^j) - \frac{\partial f_i}{\partial x_j}(x)\right| \\
= & \left|\gamma_j(x,\Delta,\zeta^j) - \frac{\partial f_i}{\partial x_j}(y^{j-}) + \frac{\partial f_i}{\partial x_j}(y^{j-}) - \frac{\partial f_i}{\partial x_j}(x)\right| \\
\leq & \left|\gamma_j(x,\Delta,\zeta^j) - \frac{\partial f_i}{\partial x_j}(y^{j-})\right| + \left|\frac{\partial f_i}{\partial x_j}(y^{j-}) - \frac{\partial f_i}{\partial x_j}(x)\right| \\
\leq & \frac{1}{2}K_i\Delta + K_i\frac{1}{2}\Delta(\sqrt{n}+2) \\
= & \frac{1}{2}K_i\Delta(\sqrt{n}+3).
\end{aligned}$$

Hence,

$$\begin{aligned}
& |\gamma(x,\Delta,z) - \nabla f_i(x)| \\
= & \sqrt{\left(\gamma_1(x,\Delta,\zeta^1) - \frac{\partial f_i}{\partial x_1}(x)\right)^2 + \cdots + \left(\gamma_n(x,\Delta,\zeta^n) - \frac{\partial f_i}{\partial x_n}(x)\right)^2} \\
\leq & \sqrt{\left(\frac{1}{2}K_i\Delta(\sqrt{n}+3)\right)^2 + \cdots + \left(\frac{1}{2}K_i\Delta(\sqrt{n}+3)\right)^2} \\
= & \sqrt{n\left(\frac{1}{2}K_i\Delta(\sqrt{n}+3)\right)^2} \\
= & \sqrt{n}\frac{1}{2}K_i\Delta(\sqrt{n}+3).
\end{aligned}$$

□

We conclude that convergence holds when using the Gupal estimate of the gradient of the Steklov averaged function of f as an approximate gradient in the AGS and RAGS algorithms.

Corollary 4.17. *Let $f = \max\{f_i : i = 1,\ldots,N\}$ where each $f_i \in \mathcal{C}^{1+}$ with Lipschitz constant K_i for ∇f_i. If the approximate gradient used in the AGS or RAGS algorithm is the Gupal estimate of the gradient of the Steklov averaged function, then the results of Theorems 2.6, 2.8, 2.11 and 2.12 hold.*

4.3.3 Algorithm

Algorithm

To use the Gupal estimate of the gradient of the Steklov averaged function in the AGS and RAGS algorithms, in Step 1, we sample independently and uniformly $\{z^{kl}\}_{l=1}^{m}$ from the unit cube in $\mathbb{R}^{n\times n}$, where m is the number of active functions. Then proceed as expected.

Chapter 5

Numerical Results

5.1 Versions of the AGS Algorithm

We implemented the AGS and RAGS algorithms using the simplex gradient, the centered simplex gradient and the Gupal estimate of the gradient of the Steklov averaged function as approximate gradients.

Additionally, we used the robust descent direction to create *robust stopping conditions*. That is, the AGS and RAGS algorithms terminate when

$$\Delta_k \leq \mu_k |d_Y^k| \quad \text{and} \quad |d_Y^k| < \varepsilon_{tol}, \tag{5.1}$$

where d_Y^k is the projection of 0 onto the approximate subdifferential generated using the *robust* active set. (See Lemmas 3.11 and 3.13, and Proposition 3.14 for results linking the robust stopping conditions with the Goldstein approximate subdifferential.) The implementation was done in MATLAB (v. 7.11.0.584, R2010b). Software is available upon request.

Let d^k denote the regular descent direction and let d_Y^k denote the robust descent direction. There are three scenarios that could occur when using the robust stopping conditions:

1. $|d^k| = |d_Y^k|$;
2. $|d^k| > |d_Y^k|$, but checking the stopping conditions leads to the same result (line search, radius decrease or termination); or
3. $|d^k| > |d_Y^k|$, but checking the stopping conditions leads to a different result.

In Scenarios 1 and 2, the robust stopping conditions have no influence on the algorithm. In Scenario 3, we have two cases:

1. $\Delta_k \leq \mu_k |d_Y^k| \leq \mu_k |d^k|$, but $|d^k| \geq \varepsilon_{tol}$ and $|d_Y^k| < \varepsilon_{tol}$ or
2. $\Delta_k \leq \mu_k |d^k|$ holds, but $\Delta_k > \mu_k |d_Y^k|$.

Thus, we hypothesize that the robust stopping conditions will cause the AGS and RAGS algorithms to do one of two things: to terminate early, providing a solution that has a smaller quality measure, but requires less function evaluations to find, or to reduce its search radius instead of carrying out a line search, reducing the number of function evaluations carried out during that iteration and calculating a more accurate approximate subdifferential at the next iteration.

Our goal in this testing is to determine if there are any notable numerical differences in the quality of the three approximate gradients (simplex, centered simplex, and Gupal estimate), the two search directions (non-robust and robust), and the two stopping conditions (non-robust and robust). This results in the following 12 versions:

AGS Simplex (1. non-robust /2. robust stopping)

RAGS Simplex (3. non-robust /4. robust stopping)

AGS Centered Simplex (5. non-robust /6. robust stopping)

RAGS Centered Simplex (7. non-robust /8. robust stopping)

AGS Gupal (9. non-robust /10. robust stopping)

RAGS Gupal (11. non-robust/12. robust stopping)

5.2 Test Sets and Software

Testing was performed on a 2.0 GHz Intel Core i7 Macbook Pro and a 2.2 GHz Intel Core 2 Duo Macbook Pro. We used the test set from Lukšan-Vlček, [LV00]. The first 25 problems presented are of the desired form

$$\min_x F(x) \text{ where } F(x) = \max\{f_i(x) : i = 1, 2, \ldots, N\}.$$

Of these 25 problems, we omit problem 2.17 because the sub-functions are complex-valued. Thus, our test set is a total of 24 finite minimax problems with dimensions from 2 to 20. There are several problems featuring functions f_i that have the form $f_i = |\mathbf{f_i}|$, where $\mathbf{f_i}$ is a smooth function. We rewrote these functions as $f_i = \max\{\mathbf{f_i}, -\mathbf{f_i}\}$. The resulting test problems have from 2 to 130 sub-functions. A summary of the test problems appears in Table 1 in Appendix A.

5.3 Initialization and Stopping Conditions

We first describe our choices for the initialization parameters used in the AGS and RAGS algorithms.

The initial starting points are given for each problem in [LV00]. We set the initial accuracy measure to 0.5 with a reduction factor of 0.5. We set the initial search radius to 0.1 with a reduction factor of 0.5. The Armijo-like parameter η was chosen to be 0.1 to ensure that a line search success resulted in a significant function value decrease. We set the minimum step length to 10^{-10}.

Next, we discuss the stopping tolerances used to ensure finite termination of the AGS and RAGS algorithms. We encoded four possible reasons for termination in our algorithm. The first reason corresponds to our theoretical stopping conditions, while the remaining three reasons are to ensure numerical stability of the algorithm.

1. Stopping conditions met: As stated in the theoretical algorithm, the algorithm terminates for this reason when $\Delta_k \leq \mu_k |d^k|$ and $|d^k| < \varepsilon_{tol}$, where d^k is either the regular or the robust descent direction.

2. Hessian matrix has *NaN / Inf* entries: For the solution of the quadratic program in Step 2, we use the *quadprog* command in MATLAB, which has certain numerical limitations. When these limitations result in *NaN* or *Inf* entries in the Hessian, the algorithm terminates.

3. Δ_k, μ_k, and $|d^k|$ are small: This stopping criterion bi-passes the test $\Delta_k \leq \mu_k |d^k|$ (in Step 2) and stops if $\Delta_k < \Delta_{tol}$, $\mu_k < \mu_{tol}$ and $|d^k| < \varepsilon_{tol}$. Examining Theorem 2.12 along with Theorems 4.6, 4.9 and 4.16, it is clear that this is also a valid stopping criterion. We used a bound of 10^{-6} in our implementation for both Δ_k and μ_k.

4. Max number of function evaluations reached - As a final failsafe, we added an upper bound of 10^6 on the number of function evaluations allowed. (This stopping condition only occurs once in our results.)

5.4 Computing a Descent Direction

In each iteration of the AGS algorithm, we must compute the projection of 0 onto the approximate subdifferential $G(x^k)$. Additionally, for the RAGS algorithm, we must compute the projection of 0 onto the robust approximate subdifferential $G(Y^k)$.

To explain how we solve this problem in MATLAB, we first define the projection of 0 onto the convex hull of a set of points

$$X = [x_1, x_2, \ldots, x_n].$$

By definition,

$$\text{conv}(X) = \{z : z = \sum_{i=1}^{n} \alpha_i x_i,\ \alpha_i \geq 0,\ \sum_{i=1}^{n} \alpha_i = 1,\ x_i \in X\},$$

where $x \in \mathbb{R}^n$ and $\alpha = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^n$. By Definition 1.3, we know that the projection p of a point $x \in \mathbb{R}^n$ onto a closed convex set C is in $\arg\min\{|y - x|^2 : y \in C\}$. Hence, we have

$$\text{Proj}(0|\,\text{conv}(X)) \in \underset{(z,\alpha)}{\arg\min}\{|z|^2 \colon z = \sum_{i=1}^{n} \alpha_i x_i,\ \alpha_i \geq 0,\ \sum_{i=1}^{n} \alpha_i = 1,\ x_i \in X\}.$$

Next we show that this minimization problem is a quadratic program that can be solved using a quadratic program solver. For the sake of notation, let $x = x^k$. We are given $G(x) = \text{conv}(\nabla_A f_i(x))_{i \in A(x)}$, i.e.,

$$G(x) = \{g : g = \sum_{i \in A(x)} \lambda_i \nabla_A f_i(x),\ \sum_{i \in A(x)} \lambda_i = 1, \lambda_i \geq 0\}.$$

Without loss of generality, assume the first m functions in our finite set of f_i functions are active ($m \leq n$). Let

$$\lambda = [\lambda_1\ \lambda_2\ \ldots\ \lambda_m]^\top$$

and

$$\Lambda = [\nabla_A f_1(x)\ \nabla_A f_2(x)\ \ldots\ \nabla_A f_m(x)].$$

Then for all $g \in G(x)$,

$$g = \Lambda\lambda \quad \text{for some } \lambda \geq 0,\ \sum_{i \in A(\bar{x})} \lambda_i = 1.$$

Define $A = [1\ 1\ \ldots\ 1]$. Then $A\lambda = 1$. Now

$$\|g\|^2 = (\Lambda\lambda)^\top(\Lambda\lambda) = \lambda^\top \Lambda^\top \Lambda\lambda.$$

Let

$$
\begin{aligned}
&H \\
&= \Lambda^\top \Lambda \\
&= \begin{bmatrix}
\nabla_A f_1(x)^2 & \nabla_A f_1(x)^\top \nabla_A f_2(x) & \cdots & \nabla_A f_1(x)^\top \nabla_A f_m(x) \\
\nabla_A f_2(x)^\top \nabla_A f_1(x) & \nabla_A f_2(x)^2 & \cdots & \nabla_A f_2(x)^\top \nabla_A f_m(x) \\
\vdots & \vdots & \ddots & \vdots \\
\nabla_A f_m(x)^\top \nabla_A f_1(x) & \nabla_A f_m(x)^\top \nabla_A f_2(x) & \cdots & \nabla_A f_m(x)^2
\end{bmatrix}.
\end{aligned}
$$

Then we have the problem

$$\min_{\lambda} \{\lambda^\top H \lambda : \lambda \geq 0, A\lambda = 1\}. \tag{5.2}$$

After solving this problem, we have

$$d = -\operatorname{Proj}(0|G(x)) = -\Lambda\lambda.$$

Lemma 5.1. *The matrix H defined above is positive semi-definite and therefore, the minimization problem in* (5.2) *is a convex quadratic program.*

Proof. Clearly, $H \in \mathbb{R}^{(m+1)\times(m+1)}$ is symmetric as $\nabla_A f_i(x)^\top \nabla_A f_j(x) = \nabla_A f_j(x)^\top \nabla_A f_i(x)$ for all $i \neq j$, $i, j = 1, \ldots, m$. Furthermore, we have for all $y \in \mathbb{R}^n$,

$$y^\top H y = y^\top \Lambda^\top \Lambda y = (\Lambda y)^\top (\Lambda y) = |\Lambda y|^2 \geq 0.$$

Thus, H is positive semi-definite. □

In implementation, we used the `quadprog.m` function in MATLAB to solve for λ. The default algorithm for the `quadprog.m` function is the interior point algorithm, which finds the projection in a limiting sense. As the quadratic programs solved in our algorithm are of small size, we used the active-set algorithm instead, thus, finding exact solutions.

5.5 Results

Due to the randomness in the AGS and RAGS algorithms, we carry out 25 trials for each version. For each of the 25 trials, we record the number of function evaluations, the number of iterations, the solution, the quality of the solution and the reason for termination. The quality was measured

by the improvement in the number of digits of accuracy, which is calculated using the formula

$$-\log\left(\frac{|F_{\min} - F^*|}{|F_0 - F^*|}\right),$$

where $F_{\min}$ is the function value at the final (best) iterate, F^* is the true minimum value (optimal value) of the problem (as given in [LV00]) and F_0 is the function value at the initial iterate. Results on function evaluations and solution qualities appear in Tables 6.2, 6.3 and 6.4 of the appendix.

To visually compare algorithmic versions, we use performance profiles. A performance profile is the (cumulative) distribution function for a performance metric [DM02]. For the AGS and RAGS algorithm, the performance metric is the ratio of the number of function evaluations taken by the current version to successfully solve each test problem versus the least number of function evaluations taken by any of the versions to successfully solve each test problem. Performance profiles eliminate the need to discard failures in numerical results and provide a visual representation of the performance difference between several solvers.

For each performance profile, we have a set $\mathcal{S}$ of n_s solvers and a set $\mathcal{P}$ of n_p problems. Our performance profiles define $t_{p,s}$ to be the number of function evaluations required to solve problem p by solver s. The baseline for comparison used is the following performance ratio

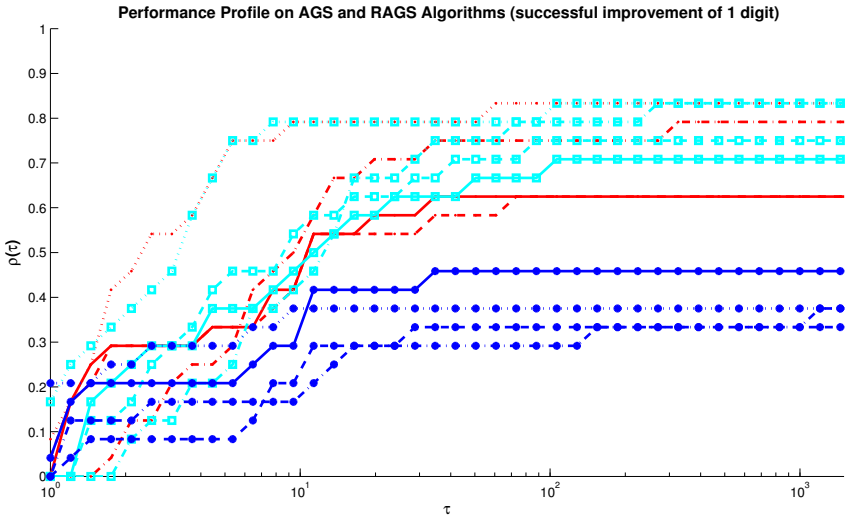
$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}.$$

As an overall assessment of the performance of the solver, we define

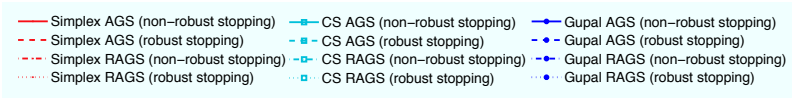
$$\rho_s(\tau) = \frac{1}{n_p}\text{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\}.$$

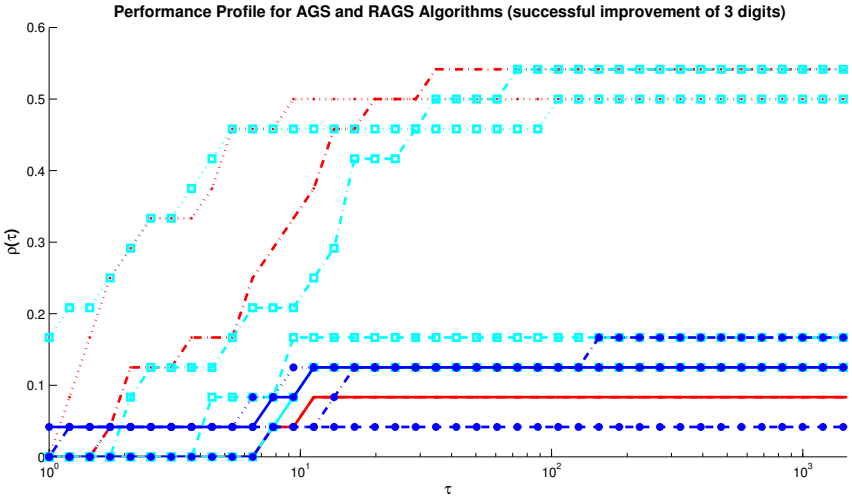
So, "$\rho_s(\tau)$ is the probability for solver $s \in \mathcal{S}$ that a performance ratio $r_{p,s}$ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio", [DM02].

In Figures 5.1(a) and 5.1(b) we include a performance profile showing all 12 versions our algorithm tested, declaring a success for 1 digit and 3 digits of improvement, respectively. We used `PProfile.m` in MATLAB, a function courtesy of Dr. Warren Hare and Valentine Koch, to generate the performance profiles.



(a) Accuracy improvement of 1 digit.





(b) Accuracy improvement of 3 digits.

Figure 5.1: Performance profiles for 12 versions of AGS/RAGS algorithm.

In general, we can see that using the Gupal estimate of the gradient of the Steklov averaged function as an approximate gradient does not produce the best results. It only produces 1 or more digits of accuracy for problems 2.1, 2.2, 2.4, 2.10, 2.18 and 2.23 (robust version). There is no significant difference between the performance of the AGS and RAGS algorithms using the simplex and centered simplex gradients as approximate gradients.

Looking at the results in Tables 6.2, 6.3 and 6.4, and our performance profiles, we can make the following two observations:

i) the versions of the RAGS algorithm generally outperform (converge faster than) the versions of the AGS algorithm, and

ii) the RAGS algorithm using the robust stopping conditions terminates faster and with lower (but still significant) accuracy.

Robust active set: From our results, it is clear that expanding the active set to include 'almost active' functions in the RAGS algorithm greatly improves performance for the simplex and centered simplex algorithm. This robust active set brings more local information into the approximate subdifferential and thereby allows for descent directions that are more parallel to any nondifferentiable ridges formed by the function.

Robust stopping conditions: We notice from the performance profiles that in terms of function evaluations, the robust stopping conditions improve the overall performance of the RAGS algorithm, although they decrease the average accuracy on some problems. These results correspond with our previously discussed hypothesis. Furthermore, upon studying the reasons for termination, it appears that the non-robust stopping conditions cause the AGS and RAGS algorithms to terminate mainly due to Δ_k and μ_k becoming too small. For the robust stopping conditions, the RAGS algorithm terminated often because the stopping conditions were satisfied. As our theory in Section 3.3 is not complete, we cannot make any theoretical statements about how the robust stopping conditions would perform in general (like those in Theorem 2.11). However, from our results, we conjecture that the alteration is beneficial for decreasing function evaluations.

In 23 of the 24 problems tested, for both robust and non-robust stopping conditions, the RAGS algorithm either matches or outperforms the AGS algorithm in average accuracy obtained over 25 trials using the simplex and centered simplex gradients. Knowing this, we conclude that the improvement of the accuracy is due to the choice of a robust search direction.

5.6 An Application in Seismic Retrofitting

This application deals with the potentially destructive inter-story drift that occurs during an earthquake. In high density metropolitan areas, the close proximity of buildings leaves little room for side-to-side movement. As a means of lessening the destructive effects of an earthquake on high rise buildings, dampers are placed between closely adjacent buildings. When the buildings sway, the dampers absorb the horizontal movement.

For this application, there are two optimization problems that need to be solved. The first is determining the optimal positions of the dampers. The second is determining the optimal damping coefficients of damper connectors. Details on the discrete optimization problem of optimal damper configurations can be found in [BHT12].

The problem of determining optimal damping coefficients is a continuous optimization problem. You can think of a damping coefficient like a friction coefficient; it represents the effect the damper has on the velocity of the i^{th} floor of the building.

In 2011, Bigdeli, Hare and Tesfamariam presented the first research on the application of mathematical optimization to the problem of optimal damping coefficients [BHT11]. As presented, a set of damping coefficients is run through a simulation that generates the 'inter-story drift' of each floor. The objective is to minimize the maximum inter-story drift. As inter-story drift is computed via simulation, no derivative information is available and DFO methods are indispensable. Applying the Nelder-Mead method, a well-known DFO algorithm, to this problem, it is shown in [BHT11] that non-uniform damping coefficients can greatly improve system performance.

This problem is a finite minimax problem. The details of the problem can be found in [BHT11]; simply put, we are trying to minimize the maximum inter-story drift between buildings. In this section, we explore the performance of two commercial solvers and the RAGS algorithm (using the simplex gradient and the robust stopping conditions) on this problem. The two commercial solvers we look at are the DFO pattern search and genetic algorithms defined in MATLAB.

The pattern search algorithm is categorized as a directional direct-search method. As in any optimization algorithm, we desire to find a new point x^{k+1} such that $f(x^{k+1}) < f(x^k)$. There are two phases to this algorithm as described in [CSV09]: the search step and the poll step. In the search step, the algorithm searches a finite set of points for function value decrease. If found, then the iterate is updated and the algorithm loops. Else, the algorithm carries out a poll step. A poll step is a localized search around

the current iterate using a given directional matrix.

The genetic algorithm follows the framework of initialization, selection and reproduction. In the initialization stage, the algorithm randomly generates an initial population, i.e., generates a random set of points. In the selection stage, the algorithm chooses the 'best' individuals in the population to reproduce, i.e., the set of points that result in the lowest function values. In the reproduction stage, the algorithm creates a new population from the 'best' individuals. The genetic algorithm requires a large number of function evaluations per iteration, so is best suited for problems with fast function call speed or when the dimension of a problem is high.

In Table 5.6, we present the summary results of our comparison of the genetic, pattern search and RAGS algorithms for 135 damper coefficient selection test problems.

Table 5.1: Summary results for 135 damping coefficient selection test problems.

	Best Time	Total Computation Time (seconds)	Best F_{opt}
GA	46	972357.5	5
PS	22	4689590.1	71
RAGS	67	864632.6	59

We can see that the RAGS algorithm has the shortest computation time for 67 of the 135 test problems and the shortest total computation time for all of the 135 test problems. If for optimal function values, we declare a tie between algorithms when

$$\frac{|F_{opt,GA/PS} - F_{opt,RAGS}|}{F_{opt,GA/PS}} < 10^{-2},$$

then the genetic algorithm tied all 5 of its best solutions with the RAGS algorithm, and the pattern search algorithm tied 40 of its 71 best solutions with the RAGS algorithm. In summary, for 104 of the 135 test problems, the RAGS algorithm had the best or tied for the best optimal solution. Thus, the RAGS algorithm is certainly comparable, if not superior, to the genetic and pattern search algorithms in terms of finding the optimal function value for these problems.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have presented a new derivative-free algorithm for finite minimax problems that exploits the smooth substructure of the problem. Convergence results are given for any arbitrary approximate gradient that satisfies an error bound dependent on the sampling radius. Three examples of such approximate gradients are given. Additionally, a robust version of the algorithm is presented and shown to have the same convergence results as the regular version.

Of the theory presented, the most influential result is Lemma 2.2, which says that our approximate subdifferential is a good approximate of our exact subdifferential. Lemma 2.2 (1) is essential in proving that our stopping conditions are sufficient and Lemma 2.2 (2) is essential in proving that our search direction (approximate direction of steepest descent) is a descent direction. Part 1 can be adapted to the Goldstein approximate subdifferential (see Lemma 3.7) and Part 2 can be adapted to the robust approximate subdifferential. The proof is elegant, relying only on the definitions of subdifferential and approximate subdifferential sets.

In Chapter 4, we see that the AGS and RAGS algorithms are flexible as to the approximate gradient used. The condition that an error bound in terms of Δ is satisfied is a reasonable assumption, as the error bounds from Kelley for the simplex and centered simplex gradients and the error bound we provided for the Gupal estimate of the gradient of the Steklov average function depend on Δ.

Through numerical testing, we found that the RAGS algorithm outperforms the AGS algorithm with respect to the accuracy of the solution obtained. The general framework of the RAGS algorithm is not so different from the method of steepest descent. However, by including the 'almost active' functions in the robust active set, the RAGS algorithm is shown to avoid the downfall of the method of steepest descent caused by nondifferentiable ridges when applied to nonsmooth functions. Additionally, we tested robust stopping conditions and found that they generally required less func-

tion evaluations before termination for the RAGS algorithm. Although the results presented in Section 3.3 do not provide a complete theory for the robust stopping conditions, they give insight into the superior performance of the robust stopping conditions in our numerical results. Of the visuals presented, Figures 3.2(a) and 3.2(b) capture the true essence of the RAGS algorithm, clearly showing that the robust stopping conditions paired with the robust version of the algorithm performed best.

6.2 Future Work

Considerable future work is available in this research direction. Most obvious is a further exploration of the theory behind the performance of the robust stopping conditions. Another direction lies in the theoretical requirement bounding the step length away from 0 (see Theorems 2.11 and 3.5). In gradient based methods, one common way to avoid this requirement is with the use of Wolfe-like conditions. We are unaware of any derivative-free variant on the Wolfe conditions.

The AGS and RAGS algorithms are hybrid algorithms; they combine the elements of several previously proposed algorithm frameworks to form new optimization algorithms. Combining the strengths of algorithms to create novel hybrid algorithms is an unbounded field of future work with great potential.

Finally, there is considerable future work in the seismic retrofitting problem, as well as numerous other real-world applications.

Bibliography

[BC11] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. CMS Books in Mathematics. Springer, New York, NY, 2011. → pages 4, 16

[BHT11] K. Bigdeli, W. Hare, and S. Tesfamariam. Determining optimal non-uniform damping coefficients for adjacent buildings via a nelder-mead method. *Proceeding of the 23rd Canadian Congress of Applied Mechanics*, 2011. 4 pages. → pages 53

[BHT12] K. Bigdeli, W. Hare, and S. Tesfamariam. Configuration optimization of dampers for adjacent buildings under seismic excitations. To appear in *Eng. Opt.*, 2012. 19 pages. → pages 53

[BJF+98] A. J. Booker, J. E. Dennis Jr., P. D. Frank, D. B. Serafini, and V. Torczon. Optimization using surrogate objectives on a helicopter test example. In *Computational methods for optimal design and control (Arlington, VA, 1997)*, volume 24 of *Progr. Systems Control Theory*, pages 49–58. Birkhäuser Boston, Boston, MA, 1998. → pages 3

[BK98] D. M. Bortz and C. T. Kelley. *The simplex gradient and noisy optimization problems*, volume 24 of *Computational methods for optimal design and control*. Birkhäuser Boston, Boston, MA, 1998. → pages 34

[BKS08] A. M. Bagirov, B. Karasözen, and M. Sezer. Discrete gradient method: derivative-free method for nonsmooth optimization. *J. Optim. Theory Appl.*, 137(2):317–334, 2008. → pages 5, 7, 8

[BLO02] J. V. Burke, A. S. Lewis, and M. L. Overton. Approximating subdifferentials by random sampling of gradients. *Math. Oper. Res.*, 27(3):567–584, 2002. → pages 5, 23

[BLO05] J. V. Burke, A. S. Lewis, and M. L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM J. Optim.*, 15(3):751–779, 2005. → pages 5, 23

[CC78] C. Charalambous and A. R. Conn. An efficient method to solve the minimax problem directly. *SIAM J. Numer. Anal.*, 15(1):162–187, 1978. → pages 23

[CJV08] A. L. Custódio, J. E. Dennis Jr., and L. N. Vicente. Using simplex gradients of nonsmooth functions in direct search methods. *IMA J. Numer. Anal.*, 28(4):770–784, 2008. → pages 34

[Cla90] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Classics Appl. Math. 5. SIAM, Philadelphia, second edition, 1990. → pages 2, 4, 8

[CSV09] A. Conn, K. Scheinberg, and L. Vicente. *Introduction to derivative-free optimization*, volume 8 of *MPS/SIAM Series on Optimization*. SIAM, Philadelphia, 2009. → pages 2, 3, 4, 5, 35, 53

[CTYZ00] X. Cai, K. Teo, X. Yang, and X. Zhou. Portfolio optimization under a minimax rule. *Manage. Sci.*, 46:957–972, July 2000. → pages 2

[CV07] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM J. Optim.*, 18(2):537–555, 2007. → pages 34

[DM02] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002. → pages 50

[DV04] R. Duvigneau and M. Visonneau. Hydrodynamic design using a derivative-free method. *Struct. and Multidiscip. Optim.*, 28:195–205, 2004. → pages 3

[Gol77] A. A. Goldstein. Optimization of Lipschitz continuous functions. *Math. Program.*, 13(1):14–22, 1977. → pages 31

[Gup77] A. M. Gupal. A method for the minimization of almost differentiable functions. *Kibernetika*, pages 114–116, 1977. (in Russian). → pages 5

[Har10] W. Hare. *Using derivative free optimization for constrained parameter selection in a home and community care forecasting model*. In *Int. Perspectives on Oper. Res. and Health Care, Proc. of the 34th Meeting of the EURO Working Group on Oper. Res. Applied to Health Sciences*, 2010. → pages 3

[HM11] W. Hare and M. Macklem. Derivative-free optimization methods for finite minimax problems. To appear in *Optim. Methods and Softw.*, 2011. 16 pages. → pages 3, 5, 7, 34

[HN12] W. Hare and J. Nutini. A derivative-free approximate gradient sampling algorithm for finite minimax problems. Submitted to *Comput. Optim. and Appl.*, February 2012. 33 pages. → pages vii

[IOKK04] J. Imae, N. Ohtsuki, Y. Kikuchi, and T. Kobayashi. Minimax control design for nonlinear systems based on genetic programming: Jung's collective unconscious approach. *Intern. J. Syst. Sci.*, 35:775–785, October 2004. → pages 2

[Kel99a] C. T. Kelley. Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. *SIAM J. Optim.*, 10(1):43–55, 1999. → pages 34

[Kel99b] C. T. Kelley. *Iterative Methods for Optimization*, volume 18 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1999. → pages 36, 37

[Kiw10] K. C. Kiwiel. A nonderivative version of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM J. Optim.*, 20(4):1983–1994, 2010. → pages 5, 7, 8, 38

[LLS06] G. Liuzzi, S. Lucidi, and M. Sciandrone. A derivative-free algorithm for linearly constrained finite minimax problems. *SIAM J. Optim.*, 16(4):1054–1075, 2006. → pages 3, 7

[LV00] L. Lukšan and J. Vlček. Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report V-798, ICS AS CR, February 2000. → pages 5, 23, 46, 47, 50

[Mad75] K. Madsen. Minimax solution of non-linear equations without calculating derivatives. *Math. Programming Stud.*, pages 110–126, 1975. → pages 3

[MFT08] A. L. Marsden, J. A. Feinstein, and C. A. Taylor. A computational framework for derivative-free optimization of cardiovascular geometries. *Comput. Methods Appl. Mech. Engrg.*, 197(21-24):1890–1905, 2008. → pages 3

[NW99] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999. → pages 13

[PGL93] G. Di Pillo, L. Grippo, and S. Lucidi. A smooth method for the finite minimax problem. *Math. Program.*, 60(2, Ser. A):187–214, 1993. → pages 3

[Pol87] E. Polak. On the mathematical foundations of nondifferentiable optimization in engineering design. *SIAM Rev.*, 29(1):21–89, 1987. → pages 2

[Pol88] R. A. Polyak. Smooth optimization methods for minimax problems. *SIAM J. Control Optim.*, 26(6):1274–1286, 1988. → pages 3

[PRW03] E. Polak, J. O. Royset, and R. S. Womersley. Algorithms with adaptive smoothing for finite minimax problems. *J. Optim. Theory Appl.*, 119(3):459–484, 2003. → pages 3

[RW98] R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1998. → pages 2, 4, 9, 19

[Sta05] R. Stafford. Random points in an n-dimensional hypersphere, 2005. → pages 12

[Wol75] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Math. Programming Stud.*, pages 145–173, 1975. → pages 5

[Wsc04] M. Wschebor. Smoothed analysis of $\kappa(A)$. *J. Complexity*, 20(1):97–107, 2004. → pages 36

[Xu01] S. Xu. Smoothing method for minimax problems. *Comput. Optim. Appl.*, 20(3):267–279, 2001. → pages 3

Appendix A

Tables

Table 6.1: Test Set Summary: problem name and number, problem dimension (N), and number of sub-functions (M); * denotes an absolute value operation (doubled number of sub-functions).

Prob. #	Name	N	M
2.1	CB2	2	3
2.2	WF	2	3
2.3	SPIRAL	2	2
2.4	EVD52	3	6
2.5	Rosen-Suzuki	4	4
2.6	Polak 6	4	4
2.7	PCB3	3	42*
2.8	Bard	3	30*
2.9	Kow.-Osborne	4	22*
2.10	Davidon 2	4	40*
2.11	OET 5	4	42*
2.12	OET 6	4	42*
2.13	GAMMA	4	122*
2.14	EXP	5	21
2.15	PBC1	5	60*
2.16	EVD61	6	102*
2.18	Filter	9	82*
2.19	Wong 1	7	5
2.20	Wong 2	10	9
2.21	Wong 3	20	18
2.22	Polak 2	10	2
2.23	Polak 3	11	10
2.24	Watson	20	62*
2.25	Osborne 2	11	130*

Table 6.2: Average accuracy for 25 trials obtained by the AGS and RAGS algorithms for the simplex gradient.

	AGS				RAGS			
	Regular Stop		Robust Stop		Regular Stop		Robust Stop	
Prob.	f-evals	Acc.	f-evals	Acc.	f-evals	Acc.	f-evals	Acc.
2.1	3018	2.082	2855	2.120	2580	9.470	202	6.759
2.2	3136	4.565	3112	4.987	4179	13.211	418	6.343
2.3	3085	0.002	3087	0.002	3090	0.002	3096	0.002
2.4	3254	2.189	3265	2.238	2986	11.559	367	7.570
2.5	3391	1.379	3138	1.351	3576	1.471	539	1.471
2.6	3260	1.236	3341	1.228	4258	1.338	859	1.338
2.7	2949	1.408	2757	1.367	4155	9.939	4190	7.230
2.8	4959	0.879	4492	0.913	3634	9.941	3435	7.655
2.9	2806	0.732	3303	0.581	16000	8.049	13681	3.975
2.10	2978	3.343	2993	3.342	3567	3.459	1924	3.459
2.11	3303	2.554	3453	2.559	35367	6.099	11725	5.063
2.12	2721	1.866	3117	1.871	15052	2.882	8818	2.660
2.13	2580	1.073	2706	0.874	43618	1.952	141	1.679
2.14	3254	1.585	3289	1.086	7713	2.696	4221	1.476
2.15	3917	0.262	5554	0.259	31030	0.286	12796	0.277
2.16	3711	2.182	4500	2.077	20331	3.242	11254	2.178
2.18	10468	0.000	10338	0.000	76355	17.717	30972	17.138
2.19	3397	0.376	3327	0.351	5403	7.105	1767	7.169
2.20	4535	1.624	4271	1.624	8757	8.435	7160	6.073
2.21	8624	2.031	8380	2.157	15225	1.334	11752	1.393
2.22	1563	0.958	1408	1.042	64116	3.049	1256	2.978
2.23	7054	2.557	10392	2.744	6092	6.117	970	6.178
2.24	4570	0.301	7857	0.298	93032	0.447	21204	0.328
2.25	3427	0.339	4197	0.340	98505	0.342	343	0.342

Table 6.3: Average accuracy for 25 trials obtained by the AGS and RAGS algorithms for the centered simplex gradient.

	AGS				RAGS			
	Regular Stop		Robust Stop		Regular Stop		Robust Stop	
Prob.	f-evals	Acc.	f-evals	Acc.	f-evals	Acc.	f-evals	Acc.
2.1	3769	2.054	3573	2.051	2351	9.469	221	7.125
2.2	3705	6.888	1284	5.154	4151	9.589	330	5.594
2.3	5410	0.003	5352	0.003	5332	0.003	5353	0.003
2.4	4059	2.520	4154	2.456	4347	11.578	296	6.834
2.5	3949	1.422	3813	1.437	4112	1.471	452	1.471
2.6	3756	1.302	3880	1.309	4815	1.338	879	1.338
2.7	4227	1.435	4187	1.373	5285	9.950	7164	6.372
2.8	6928	0.988	6933	1.003	4116	9.939	3754	7.775
2.9	3301	0.933	3743	0.949	17944	8.072	13014	2.436
2.10	3447	3.343	3424	3.342	4744	3.459	427	3.459
2.11	3593	2.768	4082	2.785	47362	6.344	11886	5.115
2.12	3321	1.892	3406	1.876	15550	2.843	10726	2.651
2.13	3067	1.355	3508	1.216	36969	1.873	519	1.643
2.14	3967	1.771	6110	1.152	9757	2.692	7284	1.510
2.15	4646	0.272	6014	0.273	23947	0.280	15692	0.277
2.16	4518	2.223	6911	2.074	22225	2.628	17001	2.215
2.18	30492	16.931	14671	16.634	125859	17.804	20815	17.293
2.19	4473	0.551	4484	0.591	8561	7.113	1697	5.851
2.20	5462	1.615	5503	1.599	8908	9.011	7846	6.042
2.21	11629	1.887	11724	1.661	18957	1.304	17067	1.339
2.22	1877	1.166	1604	1.160	1453	3.139	2066	3.644
2.23	3807	2.150	7850	3.586	15625	6.117	1020	6.230
2.24!	7198	0.302	12745	0.301	115787	0.436	61652	0.329
2.25	4749	0.339	4896	0.341	256508	0.342	568	0.342

Table 6.4: Average accuracy for 25 trials obtained by the AGS and RAGS algorithm for the Gupal estimate of the gradient of the Steklov averaged function.

	AGS				RAGS			
	Regular Stop		Robust Stop		Regular Stop		Robust Stop	
Prob.	f-evals	Acc.	f-evals	Acc.	f-evals	Acc.	f-evals	Acc.
2.1	2775	2.448	2542	2.124	13126	3.896	89	2.708
2.2	3729	3.267	2221	2.813	5029	15.904	1776	7.228
2.3	2243	0.000	2262	0.000	2276	0.000	2255	0.000
2.4	2985	2.771	2841	2.892	3475	3.449	2362	3.738
2.5	3493	1.213	3529	1.196	3447	1.211	338	1.200
2.6	3144	0.187	3245	0.188	3018	0.162	3059	0.162
2.7	2631	1.368	3129	1.248	2476	1.048	2208	1.047
2.8	2711	1.125	3898	0.893	2231	0.514	5846	0.515
2.9	3102	0.727	3011	0.600	2955	0.937	3248	0.863
2.10	3075	3.241	2927	3.272	3100	0.000	3050	0.000
2.11	2947	1.527	3307	1.528	3003	1.560	2905	1.560
2.12	3095	1.099	7179	0.000	2670	0.788	7803	0.000
2.13	2755	0.710	1485	0.715	2517	0.231	6871	0.227
2.14	2965	0.574	3070	0.427	2860	0.708	4571	0.668
2.15	2658	0.010	2386	0.017	3355	0.050	3210	0.031
2.16	3431	0.457	3256	0.459	2861	0.199	2620	0.119
2.18	3936	16.345	5814	0.000	3950	16.451	6598	4.542
2.19	3337	0.014	3270	0.011	3488	0.970	3376	0.957
2.20	4604	0.835	4434	0.808	9459	1.360	10560	1.359
2.21	5468	0.000	5418	0.000	6632	0.641	6159	0.635
2.22	21	0.000	21	0.000	21	0.000	21	0.000
2.23	5436	1.814	5176	1.877	1.00E+06	2.354	954	2.415
2.24	7426	0.280	171	0.017	7927	0.043	6389	0.283
2.25	4519	0.286	4814	0.300	3760	0.017	3209	0.023