ABSTRACT

———

I. H. Hawn, M.S.
Department of Mathematics
Northern Illinois University, 2022
Dr. Nathan Krislock, Director

In this thesis, we concern ourselves with solving the unconstrained optimization problem

$$\text{Minimize } f(x)$$

$$\text{subject to } x \in X$$

where $f\colon \mathbb{R}^N \to \mathbb{R}$ is a non-convex function, possibly with infinitely many local minima. Solving such a problem, especially in higher dimensions often proves to be an extraordinarily difficult task, either in time complexity or in the methodology itself. Indeed, mathematicians must often resort to algorithms which make use of problem structure and which may not generalize well. In this thesis, we present two algorithms which solve this problem, albeit with their own shortcomings.

First, we present a new, N-dimensional generalization of an already known deterministic algorithm in $\mathbb{R}$. This new generalization will provide insight into the often difficult space of non-convex optimization and makes use of elegant stopping conditions which provide a sufficient certificate of optimality.

Next, we address the time complexity issues of this algorithm with a known heuristic approach, an approach which is considered best practice among optimization practitioners. While no certificate of optimality may be obtained from this algorithm, its speed and accuracy give it some extreme advantages over deterministic approaches.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

MAY 2022

———

BY

I. H. HAWN

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER'S OF MATHEMATICS

DEPARTMENT OF MATHEMATICS

Thesis Director:
Dr. Nathan Krislock

ACKNOWLEDGEMENTS

I wish to dedicate this thesis to Kathryn Hawn, my loving wife, whose support enabled me to fully realize my educational aspirations.

Special thanks to my advisor, Dr. Nathan Krislock for his extraordinary help in completing this project. It would not have been possible without him.

TABLE OF CONTENTS

CHAPTER 1

DETERMINISTIC MINIMIZATION APPROACHES

Traditional methods of non-convex optimization are usually stochastic in nature and thus have no proof of convergence. In this section, we aim to address this by introducing a deterministic, non-convex optimization algorithm in $\mathbb{R}^N$. However, first we must introduce two preceding algorithms which will lay the groundwork for this new algorithm.

1.1 Uniform Grid Search

Described by Pinter [10], Uniform Grid Search provides the motivation for Piyavskii's algorithm in the next section. However, as will be shown, its practical feasibility as a standalone algorithm is somewhat questionable.

Consider a function f which is Lipschitz-continuous on some interval $[a, b]$ with Lipschitz constant L. Because it is Lipschitz, f has the following property:

$$|f(x) - f(y)| \leq L|x - y|, \ \forall \ x, y \in [a, b].$$

Now let the interval $[a, b]$ be partitioned into k sub-intervals of equal size with endpoints $x_0, \ldots, x_k$, where $a = x_0 \leq x_1 \leq \cdots \leq x_k = b$. Further, let $z_i = f(x_i)$ for $i = 0, \ldots, k$. By the Lipschitz inequality we have

$$\begin{cases} f(x) - z_{i-1} \geq -L(x - x_{i-1}), \\ f(x) - z_i \geq -L(x_i - x), \end{cases} \quad \forall \ x \in [x_{i-1}, x_i],$$

which implies that

$$\begin{cases} f(x) \geq z_{i-1} - L(x - x_{i-1}), \\ f(x) \geq z_i - L(x_i - x), \end{cases} \quad \forall\ x \in [x_{i-1}, x_i].$$

By adding these two inequalities, we obtain the following lower bound on f over the interval $[x_{i-1}, x_i]$, which we will refer to as ρ_i:

$$f(x) \geq \frac{z_{i-1} + z_i}{2} - L\frac{x_i - x_{i-1}}{2} = \rho_i, \quad \forall\ x \in [x_{i-1}, x_i].$$

From this, we obtain the following Theorem.

Theorem 1 ([10])**.** *Let*

$$\begin{cases} \alpha_k = \min\limits_{i=0,1,\ldots,k} z_i \geq \min_{x\in[a,b]} f(x) = f^*, \\ \beta_k = \min\limits_{i=1,2,\ldots,k} \rho_i \leq f^*. \end{cases}$$

As $k \to \infty$, it will be that

$$\begin{cases} \alpha_k \to f^*, \\ \beta_k \to f^*. \end{cases}$$

Proof. Assume that $-\infty < a \leq x \leq b < \infty$ and that f is Lipschitz on the interval $[a, b]$ and partition this interval into k equally spaced subintervals $[x_{i-1}, x_i]$, $i = 1, 2, \ldots, k$. Since each subinterval is equally spaced, $x_i = a + \frac{i}{k}(b - a)$, $i = 0, 1, \ldots, k$. Thus, we have

$$x_i - x_{i-1} = \frac{b - a}{k}.$$

Substitution can be applied to the inequality

$$f(x) \geq \frac{z_{i-1} + z_i}{2} - L\frac{x_i - x_{i-1}}{2}, \quad x \in [x_{i-1}, x_i],$$

to obtain

$$f(x) \geq \frac{z_{i-1} + z_i}{2} - L\frac{b-a}{2k}, \quad x \in [x_{i-1}, x_i].$$

Now let $\epsilon_k = L\frac{b-a}{2k}$. Then, $\beta_k + \epsilon_k \geq \frac{z_{k-1}+z_k}{2} \geq f^*$ since f^* is the minimum. Thus, $f^* - \epsilon_k \leq \beta_k \leq f^*$. Since $\epsilon_k \to 0$ as $k \to \infty$, we conclude that $\beta_k \to f^*$ as $k \to \infty$.

For α_k, the minimizer x^* of f is in some subinterval $[x_{i_k-1}, x_{i_k}] \subset [a, b]$. Note that $x_{i_k} - x_{i_k-1} = (b-a)/k \to 0$ as $k \to \infty$. Since $x^* \in [x_{i_k-1}, x_{i_k}]$, it will be that $0 \leq x^* - x_{i_k-1} \leq x_{i_k} - x_{i_k-1} \to 0$ as $k \to \infty$. Thus, $x_{i_k-1} \to x^*$ meaning $f(x_{i_k-1}) \to f^*$. Since $\alpha_k = \min_{i=0,1,\ldots,k} f(x_i)$, α_k also converges to f^*. □

As was stated, this algorithm provides little practical benefit. However, it proves useful as an outline for the Grid Search class of non-convex optimization algorithms.

1.2 Piyavskii's Algorithm

Described in detail by Pinter [10], Piyavskii's algorithm is a uniform grid search method first proposed by Piyavskii [11] in 1971 as a means of optimizing a one dimensional, non-convex objective over the interval $[a, b]$. Unlike the Uniform Grid Search method, this algorithm can be completed in reasonable time and for this reason, can be quite practical in the $\mathbb{R}$ setting.

1.2.1 Algorithm Overview

First we describe the general idea for this algorithm before presenting the precise procedure.

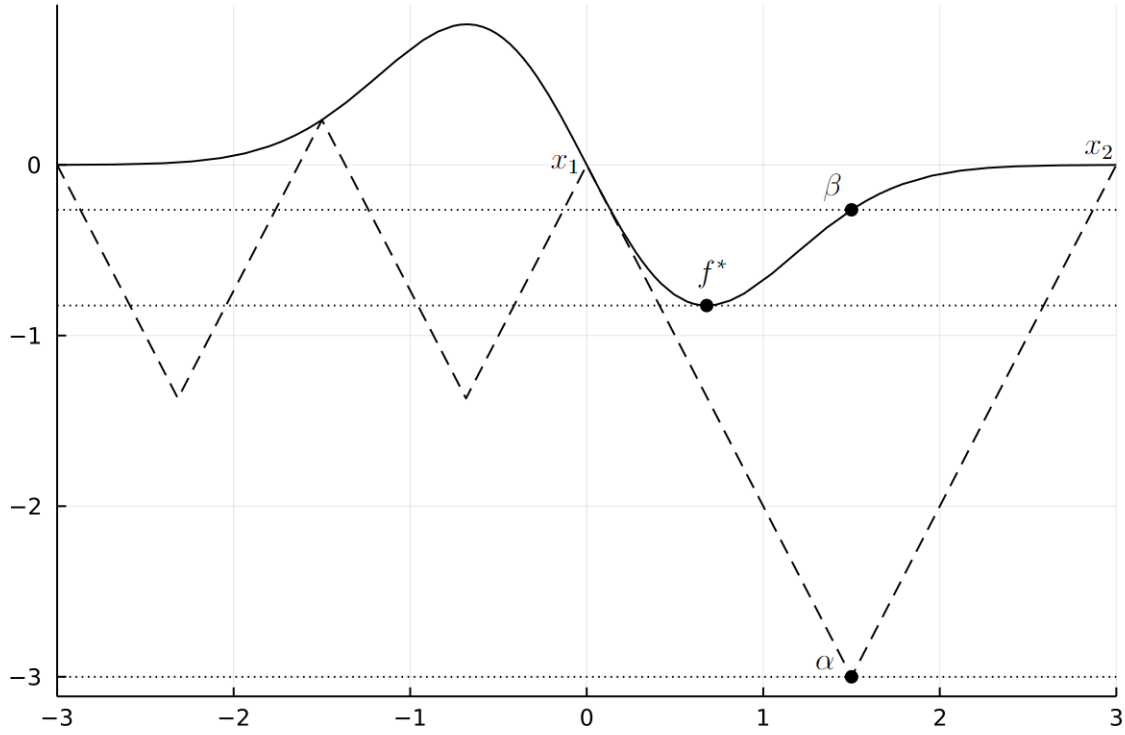

Figure 1.1: A Visualization of the Bounds Placed on f^* by Piyavskii's Algorithm

Consider objective $f{:}\mathbb{R} \to \mathbb{R}$ with Lipschitz constant L. The algorithm takes advantage of the Lipschitz continuity of f which guarantees that any two lines passing through the graph of f at points x_1 and x_2, respectively, (where $x_1 < x_2$) with slope equal to $-L$ and L, respectively, will intersect below f. In this way, a lower bound α and an upper bound β can be placed on f^*, as is shown in Figure 1.1. This notion is later proved as a lemma when we present the general version of this algorithm.

On the kth iteration, the algorithm generates a function G_k which is piecewise linear and lies below f; that is, $G_k(x) \leq f(x)$, for all $x \in [a, b]$. This property of G_k follows from the Lipschitz continuity of f, and implies that $\min_{x\in[a,b]} G_k(x) \leq f^*$. Furthermore, after each iteration we have that $\min_{x\in[a,b]} G_{k+1}(x) \geq \min_{x\in[a,b]} G_k(x)$. Thus, the sequence $\{\alpha_k\}_{k=1}^{\infty} = \{\min_{x\in[a,b]} G_k(x)\}_{k=1}^{\infty}$ is a monotonic sequence. In Theorem 4 we prove that $\alpha_k \to f^*$

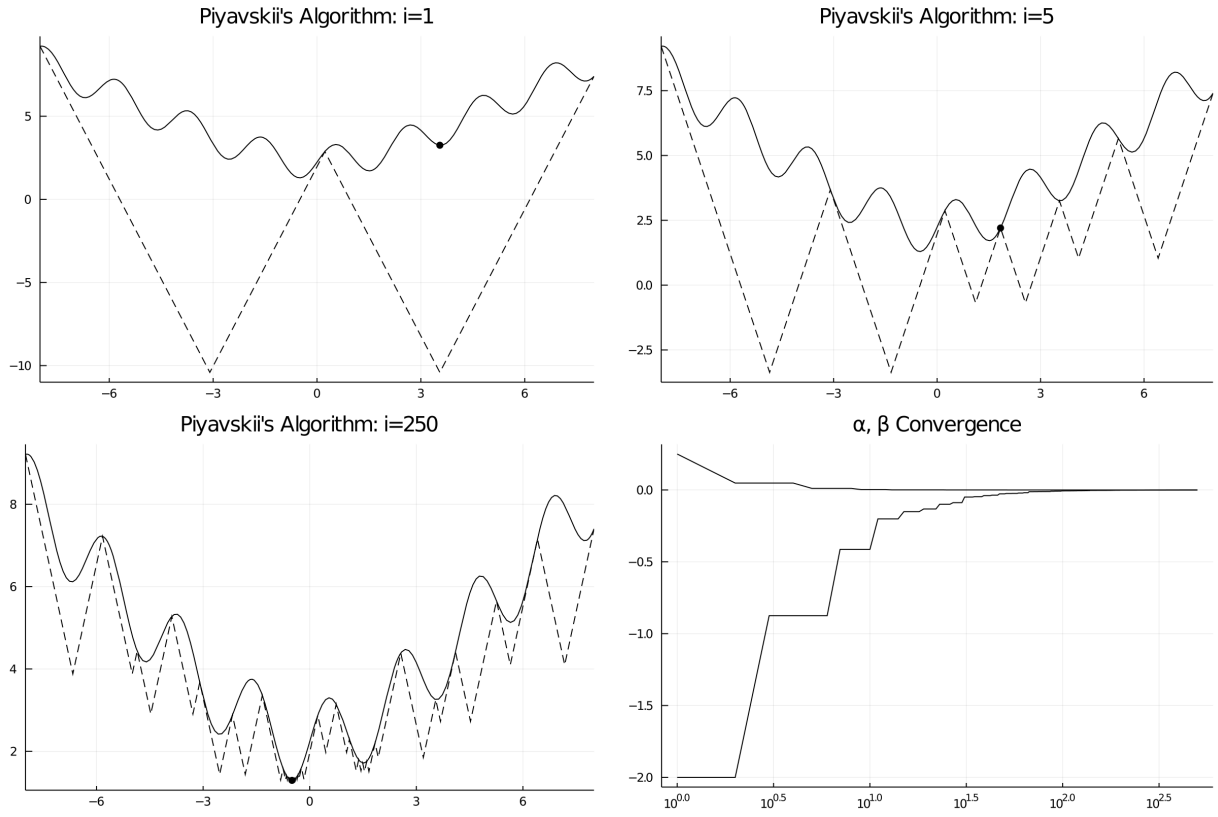

Figure 1.2: Piyavskii's Algorithm Iterative Improvement

as $k \to \infty$. By the construction of the algorithm, each local minimum of G_k will be known so minimizing it is as simple as finding the minimum of a list.

Figure 1.2 shows the iterative improvement that this algorithm provides. Notice that after each iteration k, the lower bound α_k becomes larger, or at least stays the same. In this way, the α_k values form a monotonic sequence converging to f^* from below.

The procedure is described in Algorithm 1.

1.3 Cone Search

Piyavskii's Algorithm lays the groundwork for a more complex approach in N-dimensions. This approach, while elegant in its theory, proves to be quite slow in practice. Indeed, such

Algorithm 1 Piyavskii's Algorithm

$X \coloneqq [a, b]$
$Y_0 \coloneqq \{a, b\}$
$g(x, y) \coloneqq \min\{f(y) + L(x - y), f(y) - L(x - y)\}$
$G_0(x) \coloneqq \max_{y \in Y_0} g(x, y)$
$\alpha_0 \coloneqq \min_{x \in X} G_0(x)$
$\beta_0 \coloneqq \min_{y \in Y_0} f(y)$
$k \coloneqq 0$
while $\beta_k - \alpha_k \geq \epsilon$ **do**
    $y_{k+1} \in \arg\min_{y \in Y_k} G_k(x)$
    $Y_{k+1} \coloneqq Y_k \cup \{y_{k+1}\}$
    $G_{k+1}(x) = \max_{y \in Y_{k+1}} g(x, y)$
    $\alpha_{k+1} \coloneqq \min_{x \in X} G_{k+1}(x)$
    $\beta_{k+1} \coloneqq \min_{y \in Y_{k+1}} f(y)$
    $k \coloneqq k + 1$
end while

an approach relies on certain subroutines that require an unreasonable time to run, especially for $N > 2$, at least in our implementation. However, its mathematical properties may still be of interest and hopefully a more efficient implementation can eventually be found. More details about the numerical results of this algorithm can be found in Section 3.2.1. In this section, we outline the algorithm itself as well as its proofs of convergence. Because Cone Search is a generalization of Piyavskii's algorithm, these proofs apply to that algorithm as well.

1.3.1 Algorithm Definition

In order to provide an element of practicality, we specify that the norm in use will be the infinity norm like so:

$$g(x, y) = f(y) - L\|x - y\|_\infty.$$

This transforms each g into a sort of pyramid. More specifically, each cone may be thought of as the minimum of $2N$ hyperplanes who intersect at the point $(y, f(y))$. A detailed description of the algorithm is given in Algorithm 2. The proofs of convergence can be found in Section 1.3.3.

Algorithm 2 Cone Search

$X := \{x \in \mathbb{R}^N : x_0 \in [a_0, b_0], \ldots, x_{N-1} \in [a_{N-1}, b_{N-1}]\}$
$y_0 \in X$
$Y_0 = \{y_0\}$
$g(x, y) = f(y) - L\|x - y\|_\infty$
$G_0(x) := \max_{y \in Y_0} g(x, y)$
$\alpha_0 := \min_{x \in X} G_0(x)$
$\beta_0 := \min_{y \in Y_0} f(y)$
$k := 0$
while $\beta_k - \alpha_k \geq \epsilon$ **do**
    $y_{k+1} \in \arg\min_{y \in Y_k} G_k(x)$
    $Y_{k+1} := Y_k \cup \{y_{k+1}\}$
    $G_{k+1}(x) = \max_{y \in Y_{k+1}} g(x, y)$
    $\alpha_{k+1} := \min_{x \in X} G_{k+1}(x)$
    $\beta_{k+1} := \min_{y \in Y_{k+1}} f(y)$
    $k := k + 1$
end while

1.3.2 Computing $\alpha_{k+1} = \min_{x \in X} G_{k+1}(x)$

It is worth noting that computing $\alpha_k = \min_{x \in X} G_k(x)$ can be a difficult endeavor, especially considering the large number of g functions which may compose G_k. In this section, we aim to give a more detailed instruction set for computing α_k. It should be noted that this section of the algorithm is currently the biggest bottleneck.

For mathematical purposes, we considered G_k and each g which composed it to be functions. However, when implementing this algorithm programmatically, it seemed more con-

venient to consider each g as a sort of cone and to look at the hyperplanes which compose it. We let $h_i(x, y)$, for $i = 1, \ldots, 2N$, be the functions of x whose graphs are the $2N$ hyperplanes intersecting at the point $(y, f(y))$ and that compose the cone centered at y. Then

$$g(x, y) = f(y) - L\|x - y\|_\infty = \min_i h_i(x, y).$$

We define an intersection of cones to be the unique intersection of $N+1$ of their component hyperplanes. Such an intersection $(\bar{x}, g(\bar{x}, y))$ will be valid if:

1. $g(\bar{x}, y) = f(y) - L\|\bar{x} - y\|_\infty$ holds for each cone involved in the intersection,
2. $g(\bar{x}, y) \geq f(y) - L\|\bar{x} - y\|_\infty$ holds for each cone not involved in the intersection.

In other words, an intersection will only be valid if it lies on the cones whose hyperplanes generated it and if it does not lie below any other cones which make up G_k. These conditions ensure that each intersection point lies on G_k. Finally, we may simply take the minimum of each of these valid intersection points to get α_k.

1.3.3 Proofs of Convergence

We now establish the proof of convergence for the Cone Search algorithm, following the general procedure used by Piyavskii in the original publication [11].

Lemma 2. *For continuous functions $f: \mathbb{R}^N \to \mathbb{R}$ and $g: \mathbb{R}^N \to \mathbb{R}$, the functions $h_1(x) = \min\{f_0(x), \ldots, f_k(x)\}$ and $h_2(x) = \max\{f_0(x), \ldots, f_k(x)\}$ are continuous.*

Proof. We will first prove this for $k = 1$. Let $f: \mathbb{R}^N \to \mathbb{R}$ and $g: \mathbb{R}^N \to \mathbb{R}$ be continuous functions. Define $h(x) = \min\{f(x), g(x)\}$. Consider some $\hat{x} \in \mathbb{R}^N$ where $f(\hat{x}) < g(\hat{x})$. Then

$h(\hat{x}) = f(\hat{x})$. Fix some $\epsilon > 0$ and define $\hat{\epsilon} = \frac{g(\hat{x})-f(\hat{x})}{2}$. Notice that for $|f(x) - f(\hat{x})| < \hat{\epsilon}$, we have $f(x) < g(\hat{x})$ immediately if $f(x) < f(\hat{x})$. If $f(x) \geq f(\hat{x})$, then:

$$f(x) - f(\hat{x}) < \frac{g(\hat{x}) - f(\hat{x})}{2} \implies f(x) < \frac{g(\hat{x}) + f(\hat{x})}{2} \leq \frac{g(\hat{x}) + f(x)}{2} \implies f(x) < g(\hat{x}).$$

Let $\epsilon^* = \min\{\epsilon, \hat{\epsilon}\}$. By the continuity of f, there is a $\delta > 0$ such that $\|x - \hat{x}\| < \delta$ implies $|f(x) - f(\hat{x})| < \epsilon^*$. Therefore, if $\|x - \hat{x}\| < \delta$, then $h(x) = f(x)$ and $|h(x) - h(\hat{x})| = |f(x) - f(\hat{x})| < \epsilon^* \leq \epsilon$. The proof for the case when $f(\hat{x}) > g(\hat{x})$ proceeds in a similar manner.

Consider when $f(\hat{x}) = g(\hat{x}) = h(\hat{x})$. Fix some $\epsilon > 0$. By the continuity of f and g, there exists $\delta_f > 0$ and $\delta_g > 0$ such that $\|x - y\| < \delta_f$ implies $|f(x) - f(y)| < \epsilon$ and $\|x - y\| < \delta_g$ implies $|g(x) - g(y)| < \epsilon$. Let $\delta = \min\{\delta_f, \delta_g\}$ and let $\|x - \hat{x}\| < \delta$. For $f(x) \leq g(x)$, we will have that $|h(x) - h(\hat{x})| = |f(x) - f(\hat{x})| < \epsilon$. In the same way, for $g(x) < f(x)$, we will have that $|h(x) - h(\hat{x})| = |g(x) - g(\hat{x})| < \epsilon$, proving continuity in both cases. Thus, h is continuous.

Let $\{f_i\}_{i=0}^k$ be a finite collection of continuous functions and let

$$h_1(x) = \min\{f_0(x), \ldots, f_k(x)\}.$$

Note that

$$\min\{f_0(x), \ldots, f_k(x)\} = \min\{f_0(x), \min\{f_1(x), \min\{f_2(x), \ldots, \min\{f_{k-1}(x), f_k(x)\}\cdots\}\}\}.$$

Thus, h_1 is continuous by the first part of this proof. By extension,

$$h_2(x) = \max\{f_0(x), \ldots, f_k(x)\}$$

is continuous as well since

$$\max\{f_0(x), \ldots, f_k(x)\} = -\min\{-f_0(x), \ldots, -f_k(x)\}.$$

Hence, the original statement is proved. □

Lemma 3. *Let $f: \mathbb{R}^N \to \mathbb{R}$ be a Lipschitz continuous function with Lipschitz constant L and let $g(x, y) = f(y) - L\|x - y\|$. Then $g(x, y) \leq f(x)$, for all $x \in \mathbb{R}^N$, with equality when $x = y$.*

Proof. The function f is Lipschitz so $|f(x) - f(y)| \leq L\|x - y\|$. Thus, we have:

$$\begin{aligned} f(x) - g(x, y) &= f(x) - f(y) + L\|x - y\| \\ &\geq f(x) - f(y) + |f(x) - f(y)| \geq 0. \end{aligned}$$

In addition, we have $g(y, y) = f(y) - L\|y - y\| = f(y)$. □

Theorem 4 ([11])**.** *For a Lipschitz function $f: X \to \mathbb{R}$, the sequence α_k generated by Algorithm 2 converges to f^*.*

Proof. Let $f: X \to \mathbb{R}$ be a Lipschitz function with Lipschitz constant L. Recall that the function $g: X \times X \to \mathbb{R}$ is defined as

$$g(x, y) = f(y) - L\|x - y\|.$$

Further, recall that

$$G_k(x) = \max_{y \in Y_k} g(x, y).$$

Since each $g(x, y)$ is continuous, $G_k(x)$ is also continuous by Lemma 2 . We see that for each g which defines G_k, we have that $g(y, y) = f(y)$ for all $y \in Y_k$ and $g(x, y) \leq f(x)$ by Lemma 3. This implies that $G_k(x) \leq f(x)$ for all $x \in X$.

Since $Y_k \subset Y_{k+1}$, we will have that $\max_{y \in Y_k} g(x,y) \leq \max_{y \in Y_{k+1}} g(x,y)$. Thus, $G_0(x) \leq G_1(x) \leq \cdots \leq G_k(x) \leq f(x)$. Recall that

$$\alpha_k = \min_{x \in X} G_k(x).$$

Therefore we have that

$$\alpha_k \leq \min_{x \in X} f(x) = f^*.$$

Furthermore, we see that α_k is a monotone sequence. Thus, it must converge to some value $d \leq f^*$.

Assume, for the sake of contradiction, that $d < f^*$. Let $\epsilon = f^* - d > 0$. Consider positive integers j and k such that $j < k$. Then, $y_j \in Y_{k-1}$. Since $G_{k-1}(y) = f(y)$ for any $y \in Y_{k-1}$, we must have that $G_{k-1}(y_j) = f(y_j) \geq f^*$ for any $j < k$. However, $\alpha_{k-1} = G_{k-1}(y_k)$ by definition. Further, we have that $G_{k-1}(y_k) \leq d < f^*$ implying $G_{k-1}(y_j) - G_{k-1}(y_k) \geq f^* - d = \epsilon$. Since G_{k-1} is continuous, there is a $\delta > 0$ such that for $x, y \in X$ we have that $\|x - y\| < \delta$ implies $|G_{k-1}(x) - G_{k-1}(y)| < \epsilon$. Equivalently, $|G_{k-1}(x) - G_{k-1}(y)| \geq \epsilon$ implies $\|x - y\| \geq \delta$. Therefore, $\|y_j - y_k\| \geq \delta$ for all distinct integers j and k. Thus, $\{y_k\}_{k=0}^{\infty}$ has no convergent subsequence. However, $\{y_k\}_{k=0}^{\infty} \subset X$ and X is a compact set. By the Bolzano-Wierstrass Theorem [9], $\{y_k\}_{k=0}^{\infty}$ must have a convergent subsequence so we have reached a contradiction. Therefore, $\lim_{k \to \infty} \alpha_k = d = f^*$ □

Theorem 5 ([11])**.** *For a Lipschitz function $f: X \to \mathbb{R}$, the sequence β_k generated by Algorithm 2 converges to f^*.*

Proof. Recall that

$$\beta_k = \min_{y \in Y_k} f(y).$$

Thus, $\beta_k \geq f^*$ and β_k is monotone implying that it converges to some value $\beta \geq f^*$. Suppose $\beta > f^*$ and fix an $\epsilon > 0$ such that $\epsilon < \beta - f^*$. Define the sublevel set $F_\epsilon := \{x \in X : f(x) \leq f^* + \epsilon\}$. Notice that since $\beta_k = \min_{y \in Y_k} f(y) \geq \beta > f^* + \epsilon$, we will have that $y_k \notin F_\epsilon$ for all k.

We define the function

$$\psi(x) = \begin{cases} f(x), & x \in F_\epsilon \\ f^* + \epsilon, & x \notin F_\epsilon \end{cases}$$

and construct a sequence a sequence $A_k = \min_{x \in X} H_k(x)$ where

$$h(x, y) = \psi(y) - L\|x - y\|.$$

$$H_k(x) = \max_{y \in Y_k} h(x, y).$$

We will have that A_k converges to $f^* + \epsilon$ by the previous theorem since $f^* + \epsilon$ is the minimum of ψ. Notice that since $y_k \notin F_\epsilon$ for any k, we have that $H_k = G_k$ for all k. Thus, $A_k = \alpha_k$ which is a contradiction since

$$\lim_{k \to \infty} \alpha_k = f^* \neq f^* + \epsilon = \lim_{k \to \infty} A_k.$$

Hence, $\beta = f^*$ and the statement is proved. □

CHAPTER 2

A HEURISTIC MINIMIZATION APPROACH

Deterministic optimization approaches have many desirable properties, not the least of which is their proof of convergence. Unfortunately, not many deterministic optimization techniques are known and those that are can be quite slow as we show. Hopefully, a more efficient implementation of Cone Search can eventually be found to mitigate this. In the meantime, other optimization methods must be considered. In light of this, practitioners will use heuristic approaches; approaches which do not have a proof of convergence but work well enough to accurately solve practical problems in a timely manner.

2.1 Basin Hopping

Basin Hopping is one of the most effective methods of heuristic optimization. The premise is fairly straightforward: obtain a starting point x_0 and compute the local solution x^* using local optimization techniques. Then take a random distribution of points centered at x^* with radius ℓ. Next, pick a new x^* based on some acceptance criterion and repeat all steps until $f(x^*)$ has remained the same for some amount of fixed iterations. This method is not guaranteed to converge to the global solution but it almost certainly will based on the implementation [7].

2.1.1 The Monte-Carlo Step

Basin Hopping makes use of a class of methods known as the Monte-Carlo Step. Broadly speaking, any sort of Monte-Carlo method involves randomly selecting a large number of trial points uniformly over a search domain [5]. In the case of Basin Hopping, the Monte-Carlo step involves selecting random points $x \in B_\ell(x^*) := \{x : \|x - x^*\| \leq \ell\}$ and performing a local search from each. Whether or not the resulting local solution is accepted as the new solution is determined by certain acceptance criteria.

2.1.2 Acceptance Criteria

The most widely used acceptance criterion in basin hopping is the Metropolis Acceptance Criterion [3]. This criterion takes the old accepted solution and the new solution candidate, denoted x^* and $\hat{x}^*$ respectively and determines if $\hat{x}^*$ should be accepted or rejected based on a probability curve. A random point $\xi \in [0, 1]$, is selected and a new x^* is then appointed based on the following:

$$x^* := \begin{cases} \hat{x}^*, & \text{if } f(\hat{x}^*) < f(x^*) \text{ or } \xi < \exp\left(\min\left(0, \frac{f(x^*)-f(\hat{x}^*)}{T}\right)\right), \\ x^*, & \text{otherwise.} \end{cases}$$

In other words, if $f(\hat{x}^*) < f(x^*)$, the new solution is accepted automatically. Otherwise, it is accepted according to the probability curve $\exp\left(\frac{x^*-\hat{x}^*}{T}\right)$. Here, T is known as the Metropolis parameter.

2.1.3 Adjusting Search Length

A fixed search radius ℓ is usually not used since it can be difficult to escape local minima if valley size varies widely. To combat this, the value of ℓ can be determined by the acceptance rate of iterative points. Let the target acceptance rate and the actual acceptance rate be denoted ρ^* and ρ_k, respectively. If $\rho_k > \rho^*$, a basin trap is likely so ℓ is scaled down by some constant parameter γ until $\rho_k = \rho^*$. If $\rho_k < \rho^*$, the algorithm is returning very few better solutions. Thus, ℓ is scaled up by γ in order to broaden the search [7].

2.2 Algorithm Description

Following is a detailed description of the Basin Hopping algorithm, including its various subroutines. Due to its stochastic nature, this algorithm does not have the luxury of a proof of convergence. However, its speed is quite unmatched when solving these difficult problems as is shown in the numerical results.

2.2.1 Basin Hopping Main

Here we describe the base algorithm (Algorithm 3).

2.2.2 Local Search Subroutines

One may use many different local search methods in order to find the next x^*. For our purposes, Gradient Descent proved to be most useful for local optimization during the

Algorithm 3 Basin Hopping

$M \in \mathbb{N}$
$k := 0$
$\rho_1 := 0$
$\rho_2 := 0$
$\tau := 0$
$r(t) :=$ generate a random number in $[0, t]$
$R(t) :=$ generate a random vector in $[-t, t]^N$
$x_0 \in X$
$x_0,\ f_0 :=$ GradientDescent(f, x_0)
while $k < M$ **do**
    $v := R(\ell)$
    $\hat{x}, \hat{f} :=$ GradientDescent$(f, x_{k-1} + v)$
    if $f(x_k) < f(x_{k-1})$ or $r(1) \leq \exp\left(\min\left(0, \frac{x_k - x_{k-1}}{T}\right)\right)$ **then**
        $x_k := \hat{x}$
        $f_k := \hat{f}$
        $\rho_2 := \rho_2 + 1$
        $\tau := 0$
    else
        $\rho_1 := \rho_1 + 1$
        $\tau := \tau + 1$
    end if
    $\rho_k = \frac{\rho_2}{\rho_1 + \rho_2}$
    if $\rho_k > \hat{\rho}$ **then**
        $\ell := \frac{\ell}{\gamma}$
    else
        $\ell := \gamma\ell$
    end if
    if $\tau > \hat{\tau}$ **then**
        $x^*, f^* =$ NewtonMethod$(f, \hat{x})$
        Break
    end if
    $k := k + 1$
end while

Monte-Carlo step. Although this method is primarily used to optimize convex objectives, it shows itself to be a good local optimizer for our non-convex functions with a balanced blend of accuracy and speed.

For the sake of speed, we use a relatively large tolerance when performing local optimizations with Gradient Descent. Thus, a more intensive local optimum search is needed when the Basin Hopping algorithm detects that it has reached the global solution. Newton's Method proved to be the perfect method for this purpose since it can pick up where Gradient Descent left off and compute a much more accurate final solution.

Describing these methods in detail would only provide distraction from the main purpose of this thesis. A much more in-depth look at Gradient Descent and Newton's Method can be found in Boyd [1].

CHAPTER 3

NUMERICAL RESULTS

The purpose of this chapter is to compare and contrast the results and speed of the two main optimization approaches presented in this paper: Cone Search and Basin Hopping. In these results, the speed difference between the two methods will be emphasized and the accuracy of Basin Hopping will be explored more in depth. In addition, we will shed light on the speed issues of cone search, further emphasizing our claim that the speed of Basin Hopping overcomes its other shortcomings.

3.1 Experiment Details

Following are some parameters and details which need to be outlined in order to understand the results.

3.1.1 Objective Details

For the purposes of visualization and speed, only objectives in $\mathbb{R}^2$ will be used for cone search. As the reader will find, using this method to optimize objectives of higher dimensions will require more than reasonable time with the current implementation. As for Basin Hopping, objectives in $\mathbb{R}^N$ with $N \leq 25$ will be used to showcase the speed and accuracy of this method.

Following are the objectives which will be used. These have been selected due to their difficulty and extreme emphasis on being non-convex [13].

1. Griewank Function

   · Definition:

$$f(x) = \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

   · Domain: $x_i \in [-10, 10]$

   · Global Minimum: $f^* = 0$ at $x^* = (0, \ldots, 0)$

   · Description: This objective has infinitely many local minima evenly distributed over a shallow, parabolic valley.

2. Ackley Function

   · Definition:

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right) + 20 + \exp(1)$$

   · Domain: $x_i \in [-10, 10]$

   · Global Minimum: $f^* = 0$ at $x^* = (0, \ldots, 0)$

   · Description: This objective is characterized by largely flat outer region with infinitely many, evenly distributed local minima. For $x_i \in [-10, 10]$, the function steeply drops to its global minimum.

3. Rosenbrock Function

   · Definition:

$$f(x) = \sum_{i=1}^{N-1} \left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right)$$

- Domain: $x_i \in [-10, 10]$
- Global Minimum: $f^* = 0$ at $x^* = (1, \ldots, 1)$
- Description: This objective exhibits a sharp descent curving to its global minimum which lies in a shallow, valley.

4. Drop-Wave Function

- Definition:

$$f(x) = -\frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{0.5\left(x_1^2 + x_2^2\right) + 2}$$

- Domain: $x_i \in [-10, 10]$
- Global Minimum: $f^* = -1$ at $x^* = (0, 0)$
- Description: This objective is characterized by a prominent radial wave shape emanating from the origin.

5. Easom Function

- Definition:

$$f(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$$

- Domain: $x_i \in [-10, 10]$
- Global Minimum: $f^* = -1$ at $x^* = (\pi, \pi)$
- Description: This objective is almost completely flat except in the region around (π, π) where it drops sharply to the global minimum.

6. Rastrigin Function

- Definition:

$$f(x) = 10N + \sum_{i=1}^{N}\left(x_i^2 - 10\cos(2\pi x_i)\right)$$

- Domain: $x_i \in [-10, 10]$
- Global Minimum: $f^* = 0$ at $x^* = (0, \ldots, 0)$
- Description: This objective has infinitely many local minima evenly distributed over a steep parabolic shape.

3.1.2 Starting Points

This will only apply to Basin Hopping since Cone Search does not require a starting point. For the purposes of visibility and consistency, each basin hopping run will begin at the smallest x value for each dimension. For example in the space $x_i \in [-10, 10]$, the starting point will be $x = (-10, -10)$.

3.1.3 Lipschitz Constant

This will only apply to Cone Search since Basin Hopping does not require a Lipschitz constant. For simplicity, the Lipschitz constant for each objective will not be computed symbolically. Instead, the gradient at 1 million points will be computed for each objective and a small overestimation factor will be added to the maximum of this gradient data to come up with a working Lipschitz constant. This L value will be static throughout the algorithm, though making L dynamic may speed up the algorithm considerably.

3.1.4 Basin Hopping Parameters

Being a heuristic method, Basin Hopping requires several different parameters in order to generate accurate solutions. In practice these parameters may need to be tuned to the problem being considered. However, the following tests will utilize the same set of parameters in order to maintain consistency. These parameters are as follows:

- $\eta = 1e-4$: The stopping tolerance of the gradient descent subroutine called whenever locally optimizing a new basin.
- $\epsilon = 1e-8$: The stopping tolerance of the Newton's method subroutine called upon completion of the algorithm in order to increase accuracy.
- $\ell = 1$: The search length. This parameter refers to the maximum distance in which the algorithm may search for a new basin. This parameter is adjusted dynamically throughout the algorithm.
- $T = 1$: The Metropolis parameter used in the Monte Carlo Step.
- $\gamma = 0.9$: The ℓ scaling constant.

3.1.5 Implementation Languages

A variety of technologies were used to generate the numerical data in this section. Basin Hopping proved to be a relatively simple algorithm to implement. Thus the Julia language was sufficient to implement and generate all Basin Hopping data. On the other hand, the Cone Search algorithm proved to be quite difficult to implement programmically and a variety of different languages and technologies were used to obtain the desired result.

When considered only in $\mathbb{R}$, Cone Search is simply Piyavskii's algorithm which was simple enough to implement in pure Julia. However, when building the multidimensional case, other technologies were needed. The original prototype was built using the C# language in conjunction with the Unity game engine. This was done so as to lend a more visual approach to debugging which cut down on development time considerably. After the prototype was fully constructed, the final version was implemented in the Rust language in order to provide a high degree of speed and control to the final result. Rust is a strongly typed, low-level, systems language and was perfectly suited for this role.

Links to the code:

· Cone Search Rust code

· Cone Search C# code

· Piyavskii's Algorithm Julia code

We wrote many additional scripts and programs which exist in the above repositories; these ended up being unused, either due to performance or implementation issues.

3.1.6 Stopping Criteria

· Cone Search stopping criteria:

As was shown already, Cone Search makes use of upper and lower bounds parameters in order to determine when to stop. As was proven, values β and α converge to f^* from above and below respectively. Thus, the value $\beta - \alpha$ provides a good stopping condition. These values provide additional use in validating the selected Lipschitz constant L. If ever $\beta < \alpha$, then we can be sure that L is too small.

· Basin Hopping stopping criteria:

Due to the its heuristic nature, we cannot truly know if Basin Hopping has reached the global minimum of objective f. However, certain conditions can be checked to give a fair degree of certainty. The local optimization subroutine which computes the minimum of each searched basin relies on the norm of the gradient $\|\nabla f\|$ in order to determine if a local minimum has been reached. If $\|\nabla f\| < \epsilon$, the local optimization algorithm will cease and the algorithm will continue. As was already mentioned, we will be using Gradient Descent for this local optimization subroutine due to its consistent speed, even when far from a local solution.

The Basin Hopping algorithm itself will stop when it detects that x^* has remained the same for τ iterations. If multiple global minima exist, and x^* cannot stay the same despite f^* remaining constant, our implementation of this algorithm will detect this and switch to detecting if f^* has remained the same for τ iterations. If this is the case, the basing Hopping algorithm stops and switches to Newton's method for a high degree of accuracy.

3.2 Numerical Comparison of Algorithms

In this section, we compare both the accuracy and speed of Cone Search to that of Basin Hopping. Due to the speed limitations of Cone Search, our numerical study of this algorithm will be limited to $\mathbb{R}^2$. The algorithm was proved to converge in the more general $\mathbb{R}^N$ space but due to the time requirements of the current implementation even in $\mathbb{R}^2$, it seemed not worthwhile to test for $N > 2$.

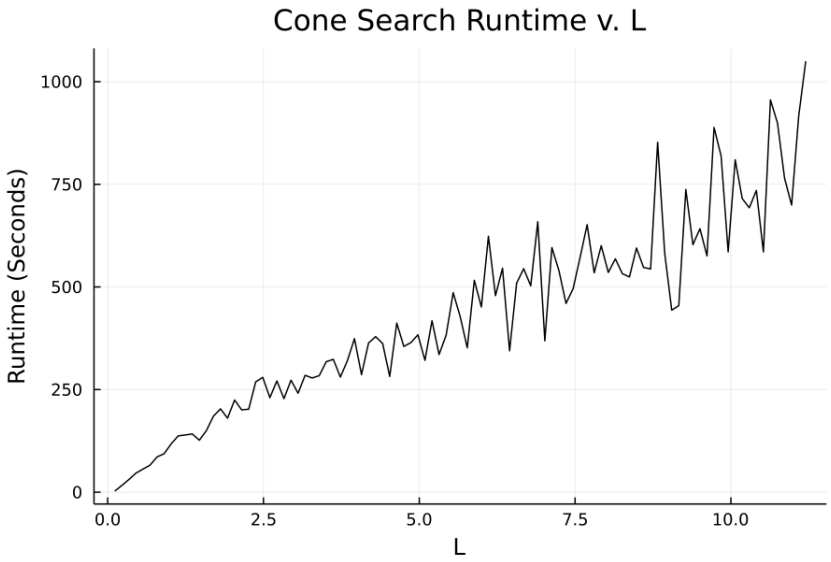

Figure 3.1: Cone Search Direct Relationship Between L and Runtime

3.2.1 Cone Search Results

Unsurprisingly, Cone Search proved to be a reliable solver, though the speed of convergence heavily relied on the gradient steepness. For example, the Griewank and Ackley functions have fairly shallow slopes throughout the test domain and thus were much faster to converge than the Easom function, which has a much steeper gradient. The solution time increase when compared to the Lipschitz constant of the objective can be seen in Figure 3.1.

From these tests, the limitations of Cone Search become apparent. Overall, the algorithm is quite slow, and while it provides interest in a mathematical sense, its time requirements render it impractical for any real-world use case. We believe that there is a much better possible implementation but were unable to find it. Figures 3.2-3.5 show the convergence of Cone Search on our objectives.

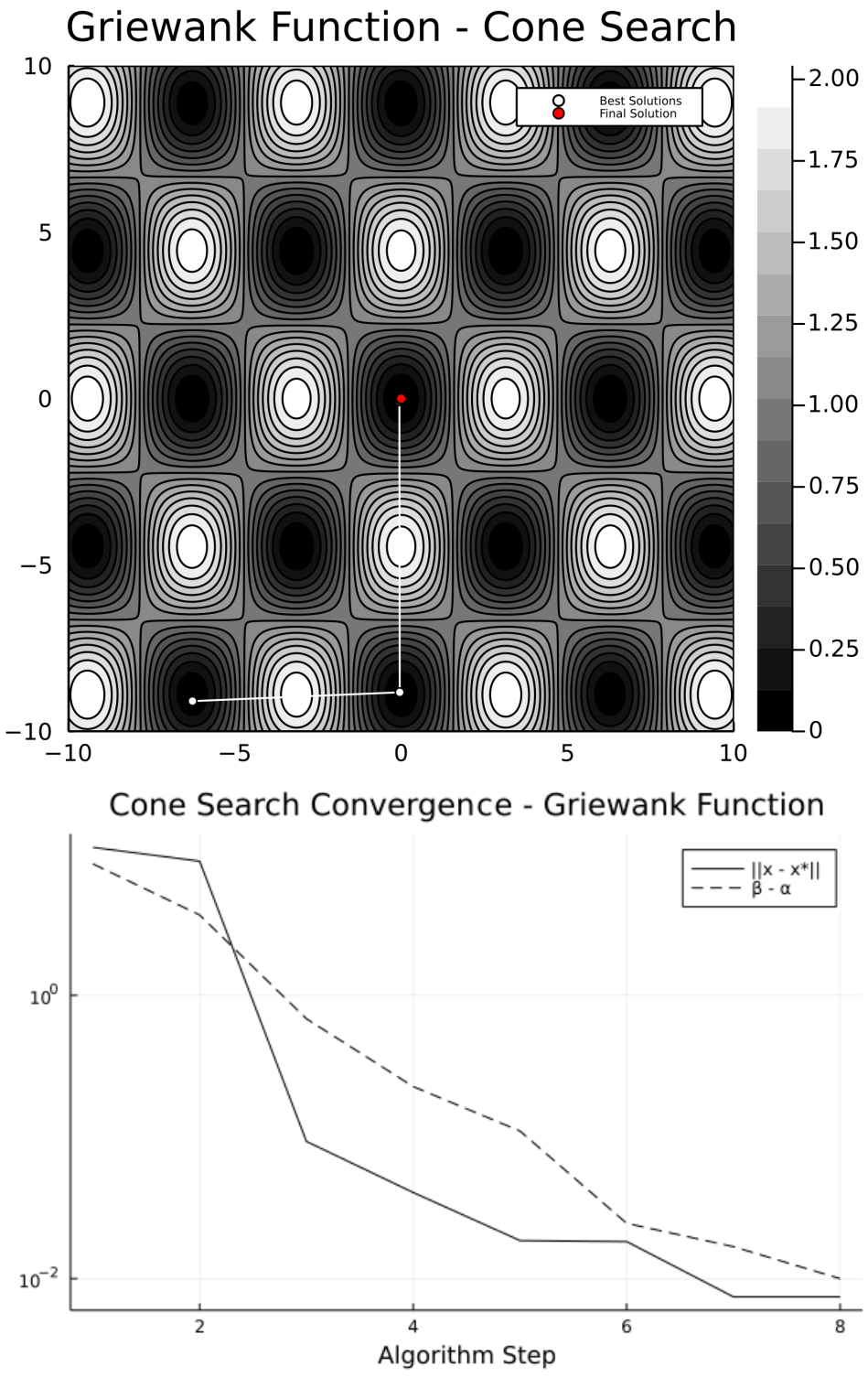

Figure 3.2: Griewank Function Solved With Cone Search

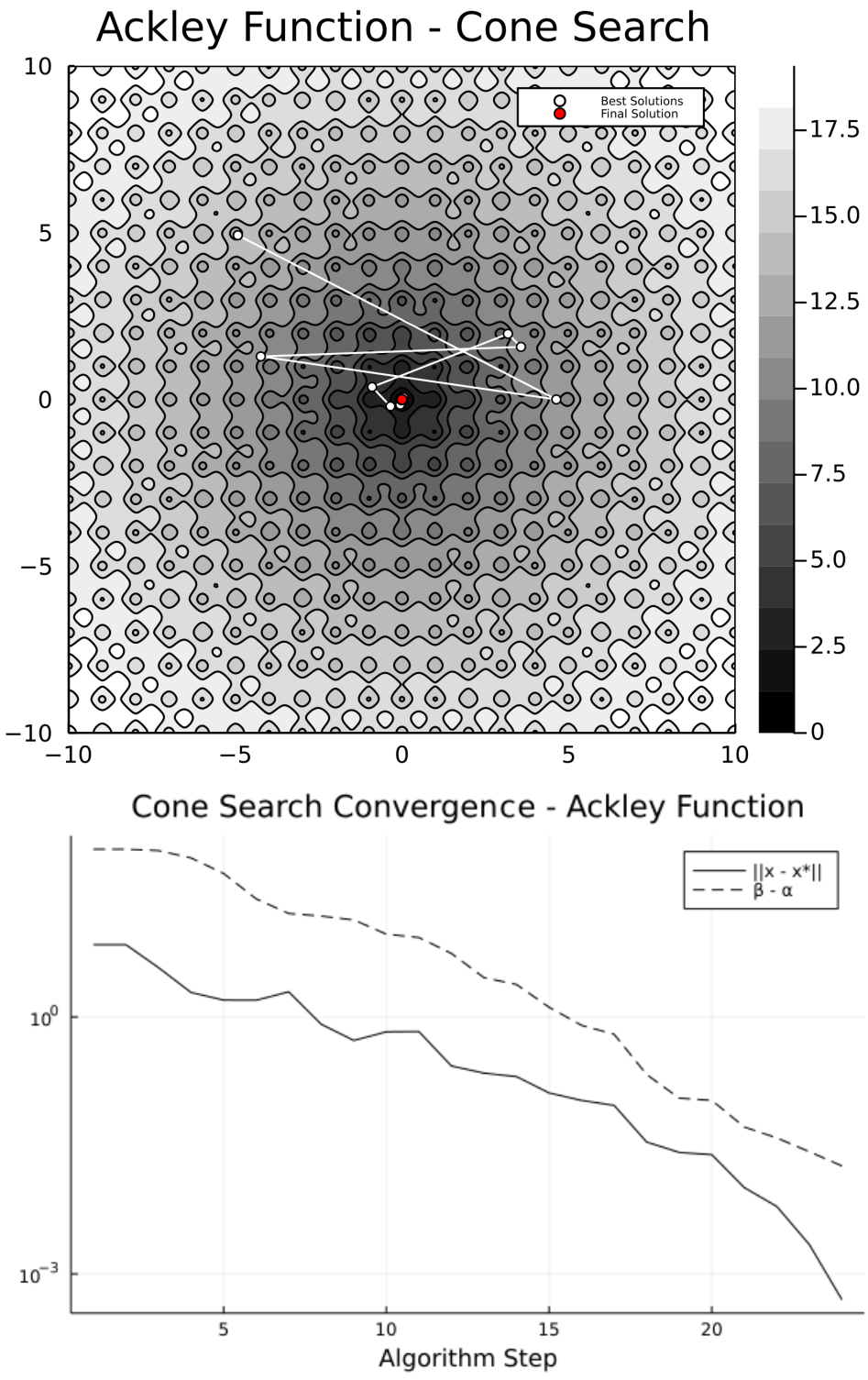

Figure 3.3: Ackley Function Solved With Cone Search

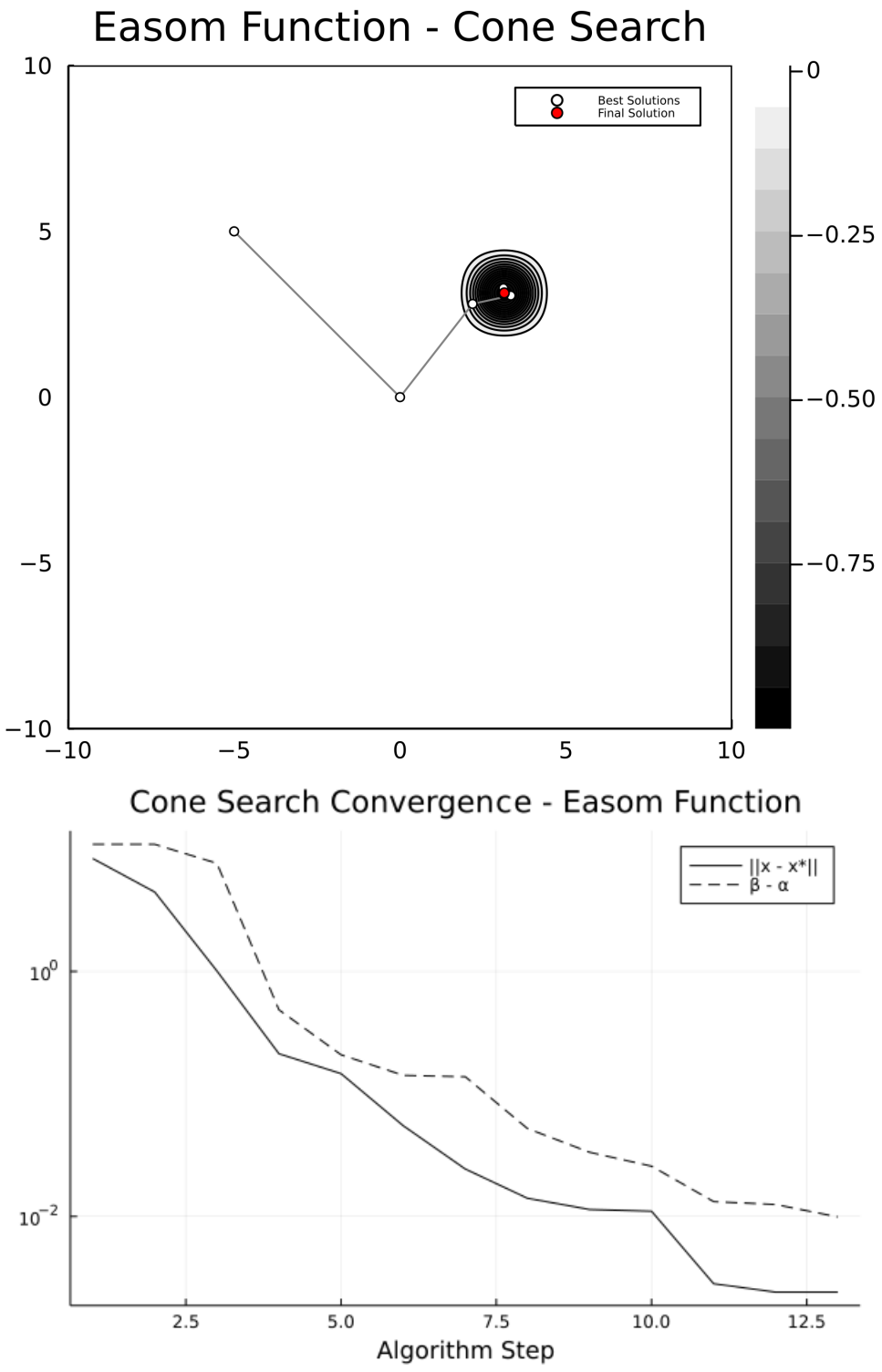

Figure 3.4: Easom Function Solved With Cone Search

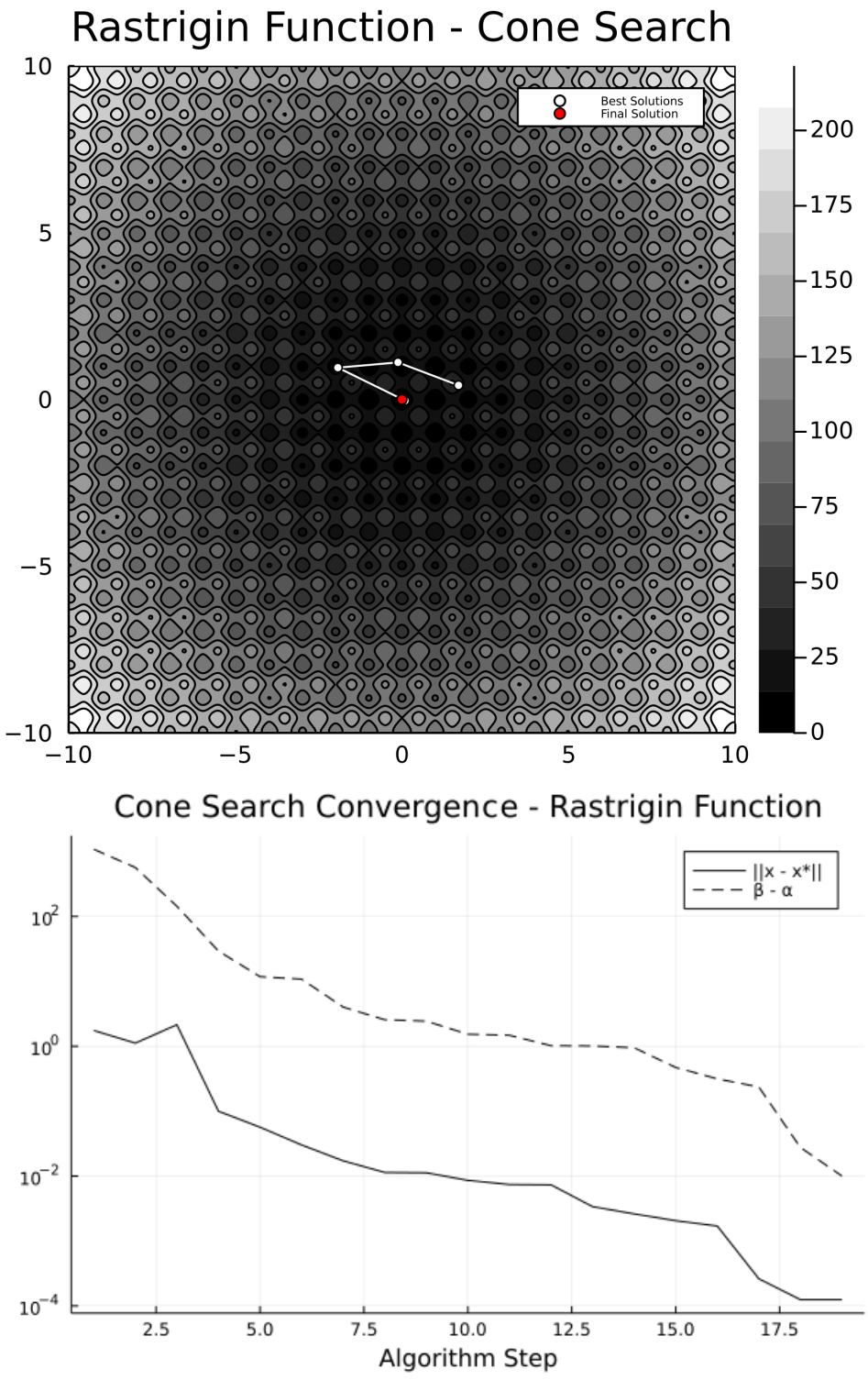

Figure 3.5: Rastrigin Function Solved With Cone Search

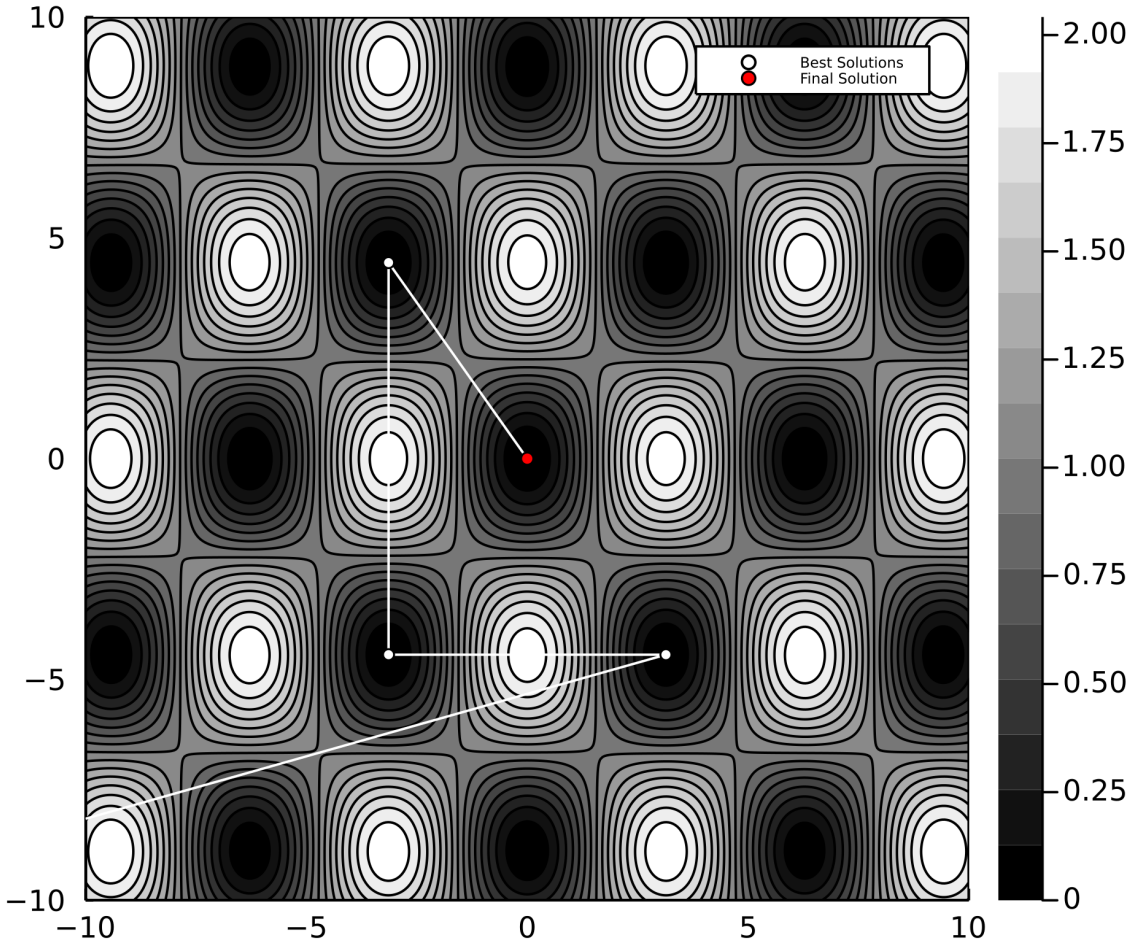

Figure 3.6: Griewank Function - Solved with Basin Hopping

3.2.2 Basin Hopping Results

In Figures 3.6-3.10 we show the convergence results of Basin Hopping across our test objectives. Unsurprisingly, these solutions were found quite quickly as can be seen in the Tables section.

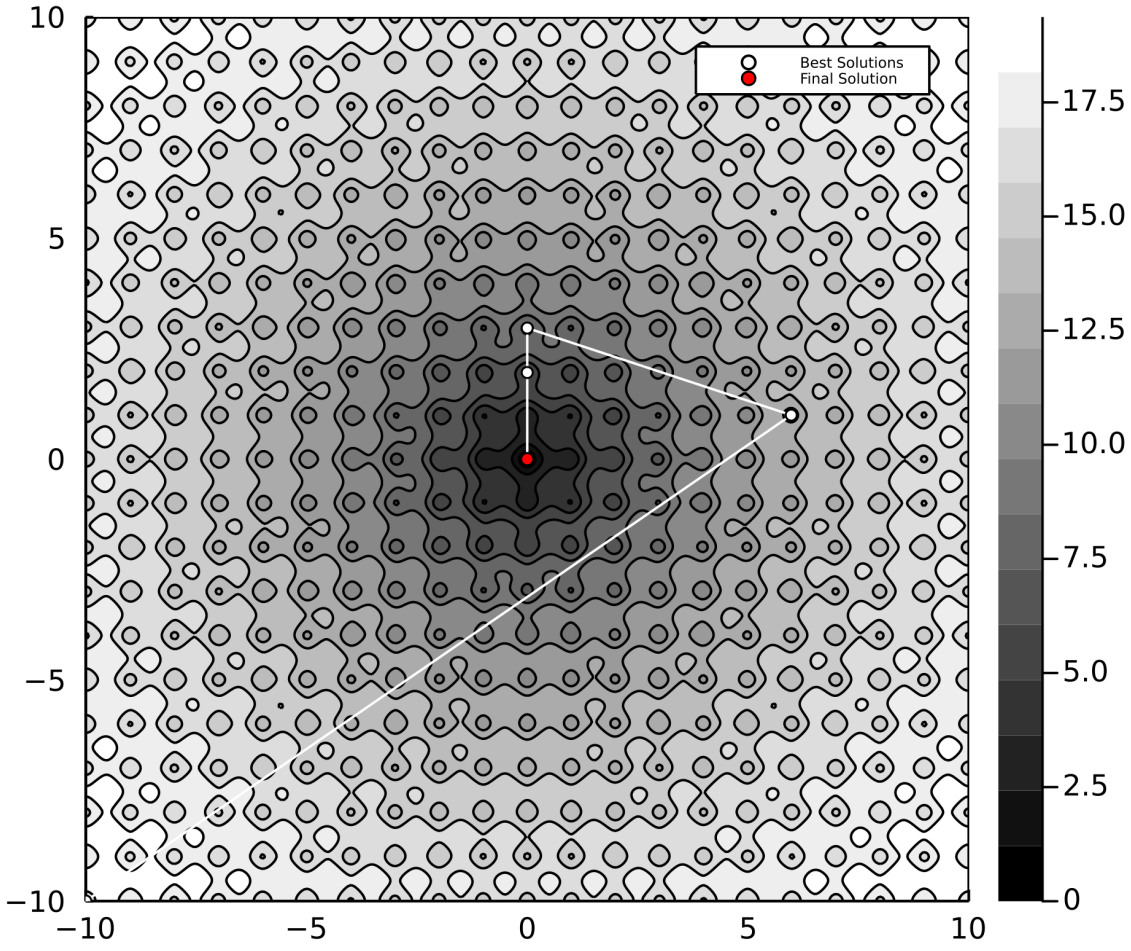

Figure 3.7: Ackley Function - Solved with Basin Hopping

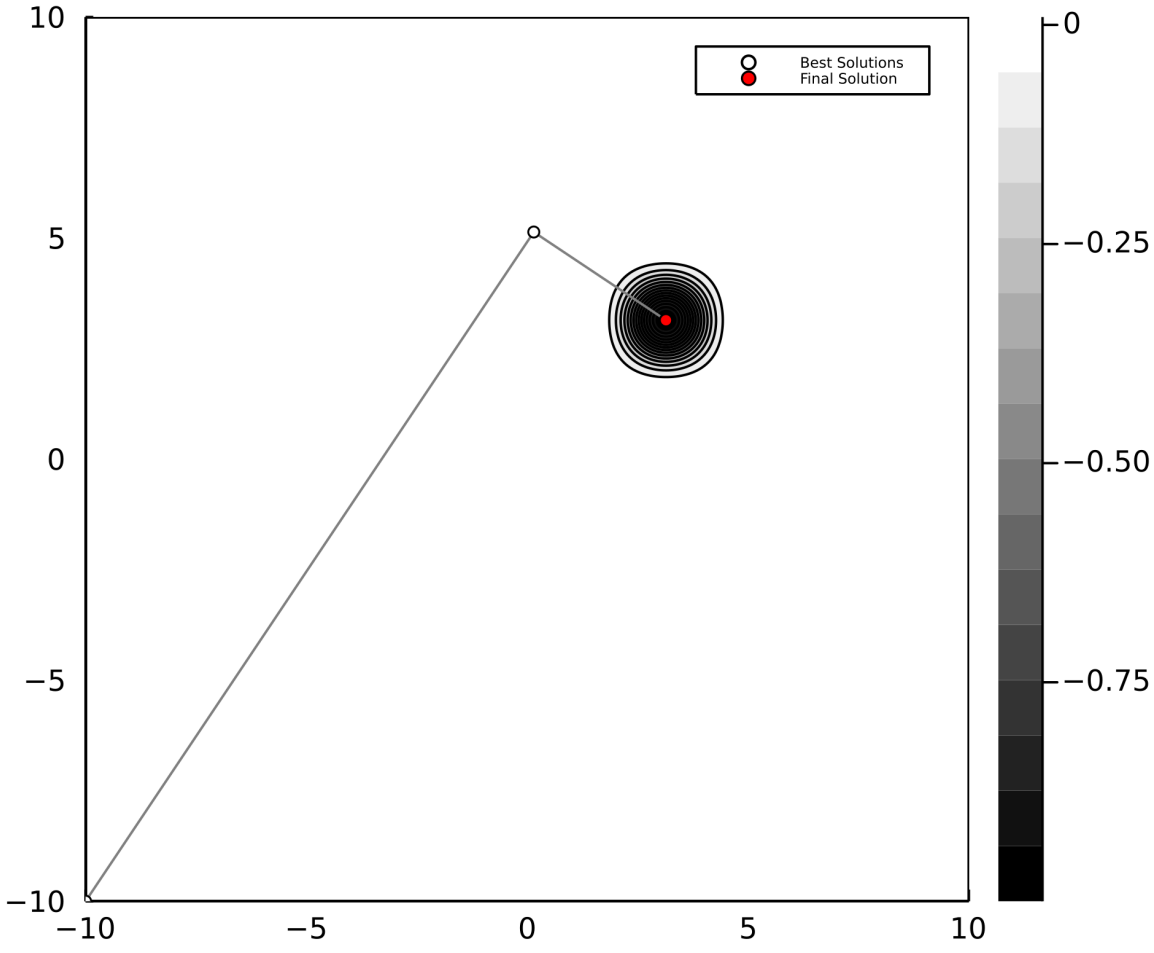

Figure 3.8: Easom Function - Solved with Basin Hopping

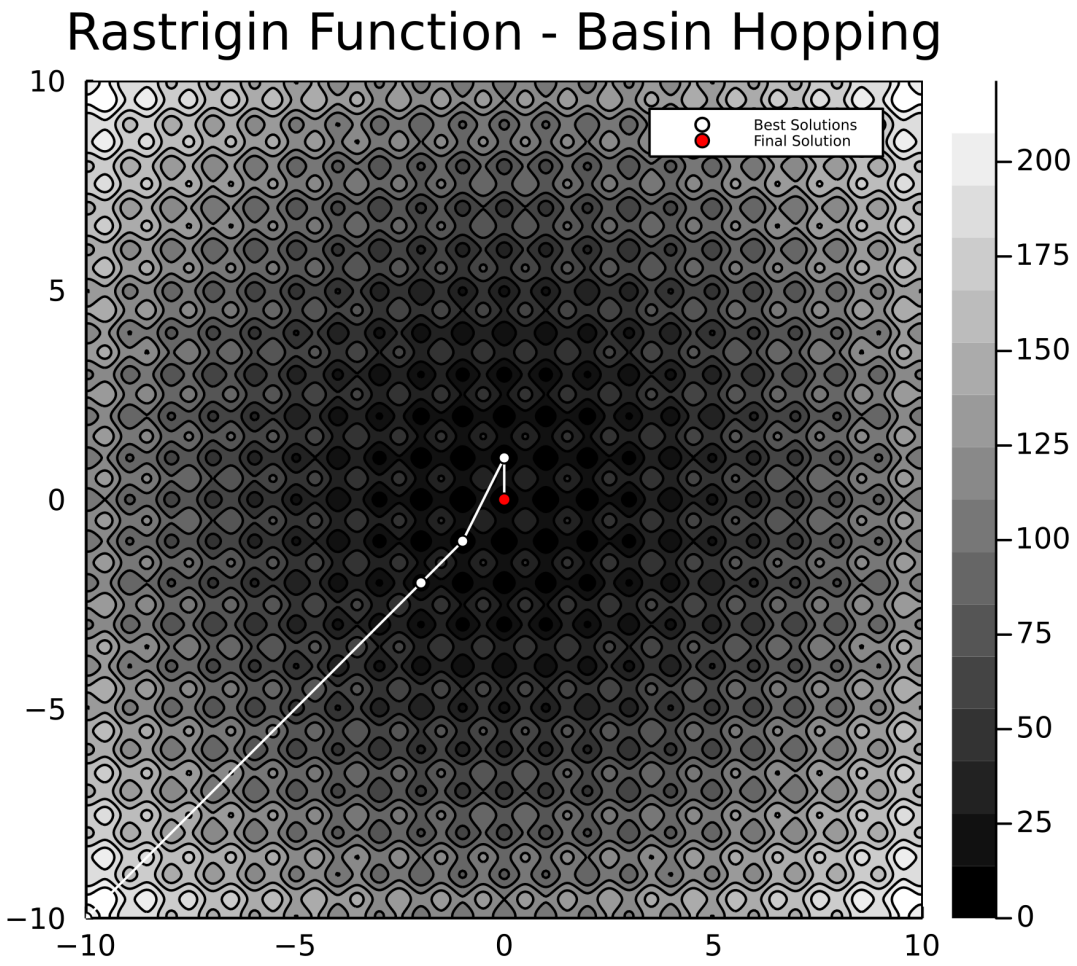

Figure 3.9: Rastrigin Function - Solved with Basin Hopping

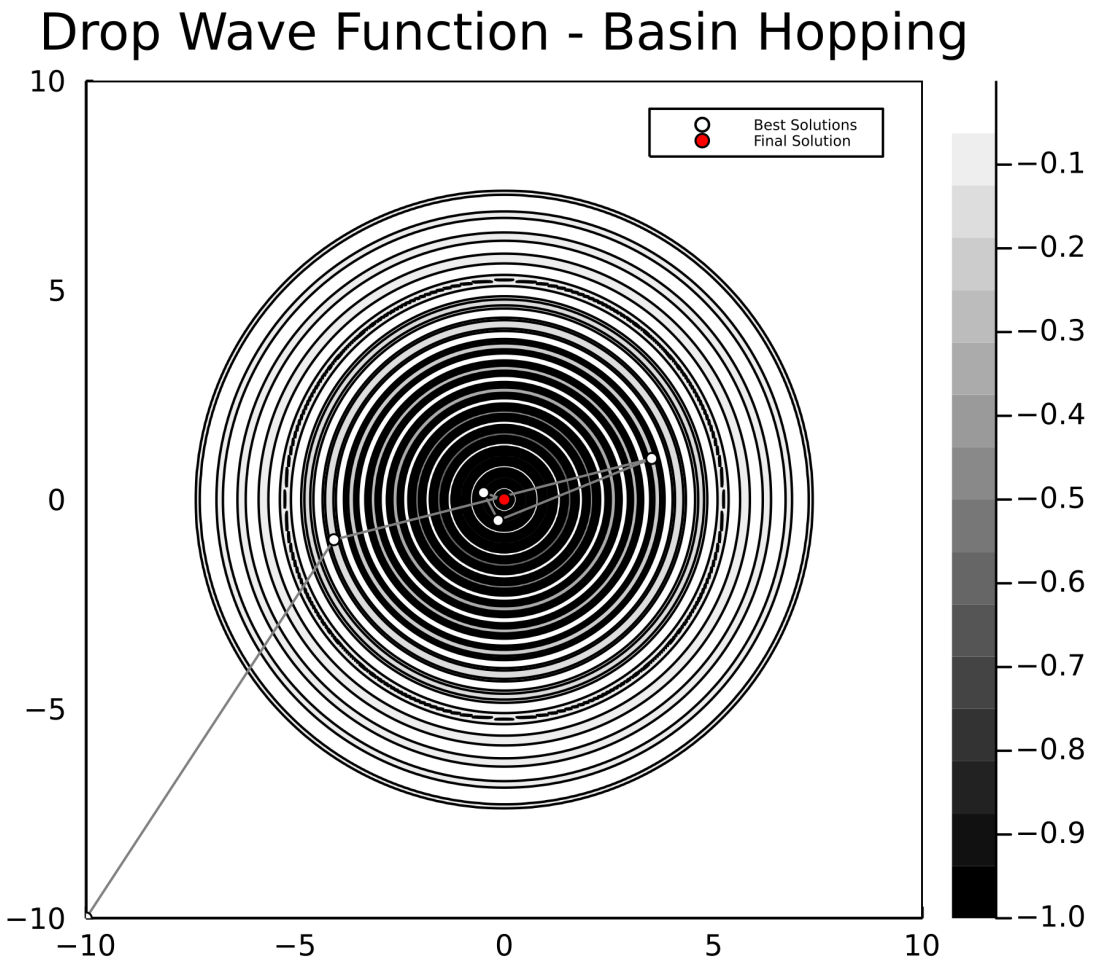

Figure 3.10: Drop Wave Function - Solved with Basin Hopping

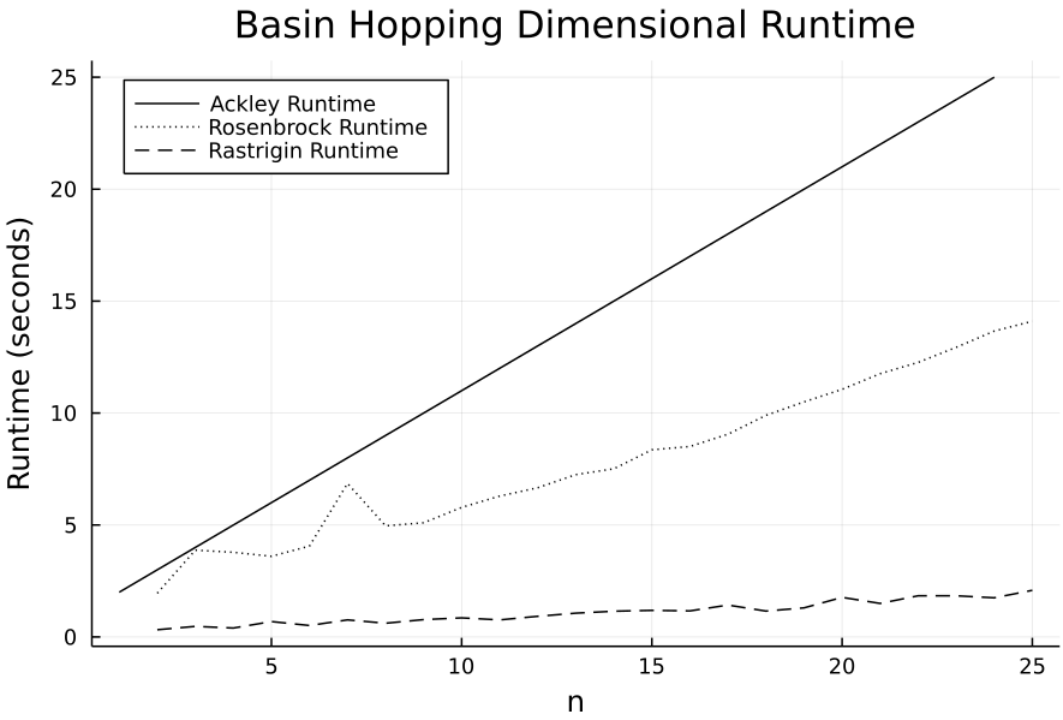

Figure 3.11: Dimensional Runtime of Basin Hopping

3.2.3 Basin Hopping v. Cone Search Direct Comparison in $\mathbb{R}^2$

Overall, Basin Hopping proves to be a much faster method of solving these non-convex problems. Understand too that these test objectives represent a worst-case scenario in terms of their solution difficulty. In practice, the problem may be much less difficult to solve quickly.

On the other hand, Cone Search proved to be much slower. Although, being a deterministic algorithm, it may be preferable in certain edge cases. Certainly, it does well in finding an approximate solution if we take ϵ to be large. This was shown to be true especially in the case when an objective has infinitely many local minima such as in the Ackley or Rastrigin functions. Such objectives require Basin Hopping to be meticulously tuned in order to achieve a solution in a high dimensional setting; indeed such a solution may not even be the

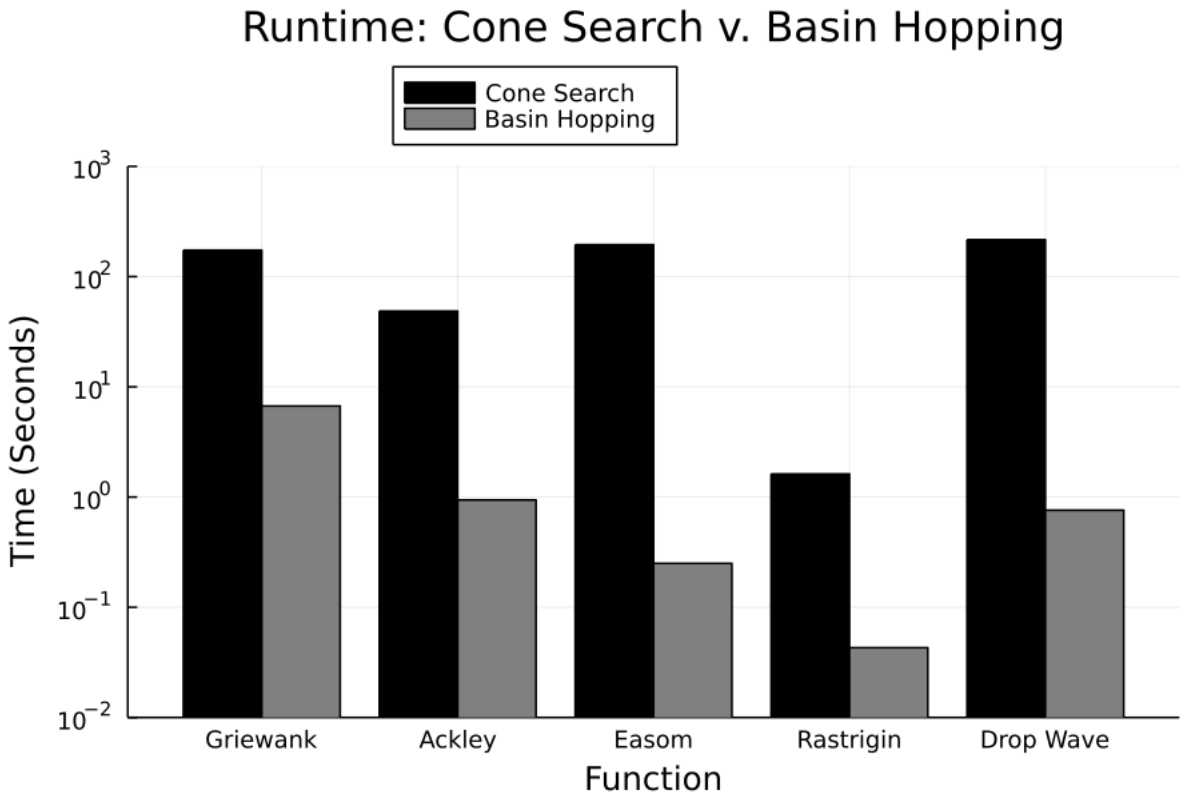

Figure 3.12: Cone Search v. Basin Hopping in $\mathbb{R}^2$

global solution. However, Cone Search provides us with a definite guarantee in this regard with its $\beta - \alpha$ value.

3.3 Tables

Following are the respective tables which correspond to the solution plots in the previous section.

x	y	$\beta - \alpha$	Step Time	Hyperplanes
-6.2712	-9.0894	8.4491	0.0019	48
-0.0427	-8.8252	3.6833	0.6387	292
-0.0526	-0.0766	0.6822	22.6698	1508
-0.0284	-0.0290	0.2268	25.7981	2796
-0.0162	-0.0091	0.1102	22.7753	3940
0.0142	0.0114	0.0246	63.5208	7128
-0.0010	-0.0074	0.0168	13.8273	7816
-0.0010	-0.0074	0.0100	24.4966	9024

Table 3.1: Griewank Function - Solved With Cone Search

x	y	$\beta - \alpha$	Step Time	Hyperplanes
-4.9238	4.9238	95.5254	0.0001	20
-4.9238	4.9238	95.5254	0.0001	24
4.6288	0.0000	92.3399	0.0115	76
-4.2346	1.2891	92.2765	0.0039	84
3.5689	1.5767	72.6603	0.0260	116
3.1781	1.9674	67.8186	0.0506	152
-0.8968	0.3738	35.4419	0.8179	328
-0.3454	-0.2043	22.6505	4.2805	596
-0.0531	-0.1579	4.8044	18.2617	1500
0.0476	0.0644	0.7667	12.1829	2108
-0.0148	-0.0242	0.3631	0.5610	2136
-0.0111	0.0057	0.2362	0.3982	2156
-0.0083	0.0056	0.2273	0.0801	2160
-0.0022	-0.0003	0.1030	1.2833	2224
0.0014	-0.0013	0.0321	3.8056	2412
0.0001	0.0016	0.0279	0.5758	2440
0.0007	0.0009	0.0200	2.8310	2576
0.0002	0.0005	0.0140	2.0383	2672
-0.0002	0.0000	0.0112	0.4293	2692
-0.0002	0.0000	0.0095	1.0123	2736

Table 3.2: Ackley Function - Solved With Cone Search

x	y	$\beta - \alpha$	Step Time	Hyperplanes
-5.0000	5.0000	10.9000	0.0001	20
0.0000	0.0000	10.9000	0.0539	128
2.1875	2.8125	7.6367	0.0971	180
3.3392	3.0631	0.4854	89.1373	2680
3.1165	3.2857	0.2086	32.4531	3480
3.1305	3.0877	0.1418	5.0344	3604
3.1305	3.1199	0.1381	0.1638	3608
3.1277	3.1394	0.0522	16.0755	4004
3.1395	3.1304	0.0334	5.7941	4148
3.1523	3.1442	0.0257	8.1499	4348
3.1401	3.1392	0.0132	23.8997	4924
3.1406	3.1394	0.0125	2.0187	4972
3.1406	3.1394	0.0099	11.8328	5248

Table 3.3: Easom Function - Solved With Cone Search

x	y	$\beta - \alpha$	Step Time	Hyperplanes
1.6960	0.4240	1083.2658	0.0000	16
-0.1208	1.1113	567.2144	0.0184	88
-1.9231	0.9536	143.5810	18.3197	1052
0.0931	-0.0356	29.2583	59.7823	3140
-0.0508	-0.0249	11.8017	32.3060	4268
-0.0073	-0.0293	10.8048	1.4928	4320
0.0059	-0.0161	4.0118	33.5679	5484
0.0037	0.0108	2.5598	13.9642	5964
0.0036	0.0107	2.4355	3.8850	6096
-0.0067	-0.0054	1.5322	17.5973	6680
-0.0063	-0.0039	1.4750	0.9680	6712
0.0012	0.0072	1.0201	20.0368	7340
-0.0011	-0.0032	1.0117	0.1410	7344
0.0017	0.0020	0.9475	1.5612	7392
-0.0010	-0.0018	0.4714	34.6527	8452
0.0015	0.0007	0.3147	36.8970	9580
0.0002	0.0001	0.2358	25.2774	10360
-0.0001	0.0001	0.0276	416.8436	22592
-0.0001	0.0001	0.0100	311.6352	31388

Table 3.4: Rastrigin Function - Solved with Cone Search

x	y	$f(x,y) - f^*$	Step Time
-12.56009	-8.876983	0.059178	0.008693
-12.56009	8.876957	0.059178	≈ 0
9.420067	4.438364	0.027125	0.007999
-3.140022	4.43837	0.007396	≈ 0
3.140022	4.438384	0.007396	0.002000
-1.59E-39	9.21E-05	2.12E-09	0.000999
-5.65E-40	-7.73E-05	1.49E-09	0.002000
-3.69E-41	5.55E-05	7.72E-10	0.004999
1.15E-41	5.21E-05	6.79E-10	0.017000
-1.25E-42	-5.20E-05	6.78E-10	0.019999
3.38E-40	5.08E-05	6.47E-10	0.030999
1.78E-42	5.03E-05	6.33E-10	0.562999
-4.73E-42	-5.01E-05	6.27E-10	0.944000
-3.73E-41	-4.99E-05	6.24E-10	0.338000
-1.36E-42	4.99E-05	6.24E-10	1.612999
-5.66E-42	-4.99E-05	6.23E-10	2.309999
-5.35E-42	-4.99E-05	6.22E-10	0.832000

Table 3.5: Griewank Function - Solved with Basin Hopping

x	y	$f(x,y) - f^*$	Step Time
-9.994947	-9.994947	17.2919	0.0052652
-0.998729	9.987280	15.16748	0.0023212
-2.979223	2.979223	9.001093	0.006999
-0.968477	-0.968477	3.574451	0.002000
1.70E-08	1.02E-08	5.61E-08	0.004999
-1.41E-08	-1.15E-08	5.16E-08	0.021000
-1.16E-09	5.07E-09	1.47E-08	0.004999
-2.20E-09	3.52E-09	1.17E-08	0.305999
2.02E-09	-3.22E-09	1.08E-08	0.138000
-3.01E-09	2.58E-10	8.56E-09	0.280999
1.74E-09	-1.90E-09	7.28E-09	0.084000
1.11E-09	-2.22E-09	7.02E-09	0.065999
1.60E-09	1.25E-09	5.74E-09	0.017999

Table 3.6: Ackley Function - Solved with Basin Hopping

x	y	$f(x,y) - f^*$	Step Time
-10	-10	1	0.007
-7.152729	-7.488938	1	≈ 0
8.906438	9.576182	1	0.007000
5.878359	-1.282337	1	≈ 0
-0.332919	5.619518	0.99999	0.217000
5.192867	1.013851	0.99996	0.001999
3.141592	3.141592	2.86E-14	≈ 0
3.141592	3.141592	7.77E-16	≈ 0
3.141592	3.141592	3.33E-16	0.012000
3.141592	3.141592	1.11E-16	0.004999

Table 3.7: Easom Function - Solved with Basin Hopping

x	y	$f(x,y) - f^*$	Step Time
-10	-10	200	0.012
-1.989912	-1.989912	7.959662	0.014000
-0.994958	-0.994958	1.989918	0.011529
2.06E-09	0.994958	0.994959	0.014000
6.52E-10	0.994958	0.994959	0.003999
2.20E-09	-9.60E-10	0	0.010999

Table 3.8: Rastrigin Function - Solved with Basin Hopping

x	y	$f(x,y) - f^*$	Step Time
-9.995128	-9.995128	0.980376	0.006000
-4.071946	-0.959154	0.814155	0.005421
3.525656	0.980094	0.770276	0.006000
-0.146229	-0.499240	0.063754	0.003652
-0.493771	0.163751	0.063754	0.009999
-6.81E-09	1.23E-07	5.48E-13	0.029000
-1.06E-07	4.49E-08	4.83E-13	0.466000
8.00E-08	-6.17E-08	3.70E-13	0.062999
3.15E-08	2.59E-08	6.03E-14	0.179000

Table 3.9: Drop Wave Function - Solved with Basin Hopping

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

The purpose of this Thesis was to present a known, heuristic method for solving non-convex optimization problems and propose a new, deterministic method for comparison. The foundations of Cone Search were known: Piyavskii had already proposed it for $\mathbb{R}$. However, the $\mathbb{R}^N$ generalization of this algorithm proposed in this paper has hopefully shed new light into the difficult and often murky area of Non-convex Optimization.

In addition to the sources cited in the text, we also found several other sources to be useful when studying this topic. These are included in the references as well.

4.2 Future Work

Many possible optimizations for Cone Search presented themselves throughout its development process. However, the purpose of this Thesis was merely to develop the foundations of this Algorithm and hopefully inspire future optimizers to increase its practical viability. In light of this, we present the following improvements which could be made to Cone Search.

1. Currently the Lipschitz constant used is static throughout the algorithm process. However, this is quite disadvantageous when minimizing objectives with a lot of slope variation. Adding a subroutine to change the L value dynamically to reflect the local

gradient limits of an objective would greatly reduce run-time, especially for objectives with highly variable slope.

2. Currently there is no way to know if a cone is adjacent to another cone other than checking all cones currently being considered. To rectify this, one may provide a method to sort the collection of cones in order to only consider intersections of adjacent cones. This is trivially done in the 1 dimensional case but doing this in $\mathbb{R}^N$ may be considerably more difficult. If done correctly, this would increase overall speed by several orders of magnitude.

3. Due to the speed limitations of Cone Search, it is often not practical to obtain high accuracy solutions. In light of this, it would be beneficial to the usability of Cone Search if a local optimization subroutine like Newton's Method were run after Cone Search finishes. In this way, we may obtain high accuracy results after Cone Search gets relatively close to a solution.

4. Finally, it would greatly speed up performance if dynamic box constraints were implemented in order to eliminate areas of the search domain.

REFERENCES

[1] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[2] L. Breiman and A. Cutler, *A deterministic algorithm for global optimization*, Mathematical Programming, 58 (1993), pp. 179–199.

[3] V. Gerbaud and X. Joulia, *Molecular modeling for physical property prediction*, (2006).

[4] V. Gergel, *A Global Optimization Algorithm for Multivariate Functions with Lipschitzian First Derivatives*, vol. 10, Springer Science & Business Media, 1997.

[5] V. Gergel and E. Kozinov, *Accelerating parallel multicriterial optimization methods based on intensive using of search information*, Procedia Computer Science, 108 (2017), pp. 1463–1472. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[6] N. Ghafari and H. Mohebi, *Optimality conditions for nonconvex problems over nearly convex feasible sets*, Arabian Journal of Mathematics, 10 (2021), pp. 395–408.

[7] M. Iwamatsu and Y. Okabe, *Basin hopping with occasional jumping*, Chemical Physics Letters, 399 (2004), pp. 396–400.

[8] Z. Li and H. A. Scheraga, *Monte Carlo-minimization approach to the multiple-minima problem in protein folding*, Proceedings of the National Academy of Sciences, 84 (1987), pp. 6611–6615.

[9] G. Oman, *A short proof of the Bolzano-Weierstrass Theorem*, The College Mathematics Journal, (2017).

[10] J. D. Pintér, *Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications*, vol. 6, Springer Science & Business Media, 2013.

[11] S. Piyavskii, *An algorithm for finding the absolute extremum of a function*, USSR Computational Mathematics and Mathematical Physics, 12 (1972), pp. 57–67.

[12] Y. Sergeyev and D. Kvasov, *A deterministic global optimization using smooth diagonal auxiliary functions*, Communications in Nonlinear Science and Numerical Simulation, 21 (2014).

[13] S. Surjanovic and D. Bingham, *Virtual library of simulation experiments: Test functions and datasets.* Retrieved October 15, 2021, from `http://www.sfu.ca/~ssurjano`.

[14] D. J. Wales and J. P. K. Doye, *Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms*, The Journal of Physical Chemistry A, 101 (1997), pp. 5111–5116.