UNIVERSITY OF CALIFORNIA, SAN DIEGO

Delaunay-based Global Optimization Algorithms and their Applications

A Dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Engineering Sciences with a Specialization in Computational Science

by

Pooriya Beyhaghi

Committee in charge:

Professor Thomas Bewley, Chair
Professor Juan Carlos Del Alamo
Professor Mauricio de Oliveira
Professor Philip Gill
Professor David Meyer

2016

The Dissertation of Pooriya Beyhaghi is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2016

DEDICATION

To my family, in the broadest sense possible.

EPIGRAPH

Those who are possessed of knowledge, they are the ones who have power.

—Abolghasem Ferdowsi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Prof. Thomas Bewley for the continuous support during my PhD studies and research. His enthusiasm, motivation, and immense knowledge have helped in innumerable ways. His guidance has contributed to my personal growth, as a person and as a professional. I could not have imagined a better advisor and mentor during the development of my research. For him, I have nothing but the utmost respect and gratitude.

I would also like to thank the rest of my thesis committee: Prof. Juan Carlos Del Alamo, Prof. Mauricio de Oliveira, Prof. Prof. Philip Gill, and Prof. David Meyer, for their time and feedback. My sincere thanks also goes to Dai Feng, who gave me the opportunity to join his team as an intern. While I was working at his company, I had a pleasant time, and that experience has helped to improve my technical skills.

I thank my friends Gianluca Meneghello, Daniele Cavaglieri, Shahrouz Alimohammadi, Hossein Sadeghi, and Huan Yu for the academic discussions, BBQs, parties, dinners, hikes, and coffee times. With you guys, I've always felt at home.

I would like to thank my family: my parents and my sister, for supporting me not only during these years, but also in every moment of my life. Without their endless love and constant support, none of this would have been possible. I would also like to express my gratitude to my extended family, who has aided me and encouraged me throughout this endeavor.

Special thanks go to Tony (Kazem) and Patricia, my uncle, and aunt, who were in fact like my parents during the last five years. Their endless support and encouragement have accompanied me in every moment during this long, beautiful journey. They helped me in more ways that I can imagine, and I am grateful for every second I spent with them.

For this, and much more, they deserve my endless gratitude.

Finally, I would like to thank Prof. Alison Mardsen and Prof. Paolo Luchini which make a great contribution for this work.

This thesis contains parts of the following publications with Prof. Thomas Bewley, Dr. Gianluca meneghello, Dr. Daniele Cavaglieri, and Shahrouz Alimohammadi as coauthors:

Chapter 2 is published in Journal of Global Optimization in the following article: P. Beyhaghi, D. Cavaglieri, T.R. Bewley, “Delaunay-based derivative-free optimization via global surrogates, part I: linear constraints", *Journal of Global Optimization*, (2015): 1-52. The dissertation author was the primary investigator and author of this paper.

Chapter 3 is also published in Journal of Global Optimization as a different article as: P. Beyhaghi, T.R. Bewley, “Delaunay-based derivative-free optimization via global surrogates, part II: convex constraints", *Journal of Global Optimization*, (2016): 1-33. The dissertation author was the primary investigator and author of this paper.

Chapter 4 is submitted to Journal of Global optimization as the following article: P. Beyhaghi, T.R. Bewley, “Implementation of Cartesian grids to accelerate Delaunay-based derivative-free optimization." The dissertation author was the primary investigator and author of this paper.

Chapter 5 is submitted to SIAM Journal on Optimization as the following article: P. Beyhaghi, T.R. Bewley, P. Beyhaghi, T.R. Bewley, “A derivative-free optimization for efficiently minimizing the infinite time-averaged statistics." The dissertation author was the primary investigator and author of this paper.

Chapter 6 is in the preparation for submission to SIAM Journal of Uncertainty Quantification as the following article: P. Beyhaghi, S. Alimohammadi, T.R. Bewley, "A

multiscale, asymptotically unbiased approach to uncertainty quantification in the numerical approximation of infinite time-averaged statistics". The dissertation author was the primary investigator and author of this paper.

Chapter 7 is submitted to Journal of Ocean Engineering as the following article: G. Meneghello, P. Beyhaghi, T.R. Bewley, “Simulation-based optimization of the hydrofoil of a flying catamaran", *Submitted to Journal of Ocean Engineering*, 2016. The dissertation author was the secondary investigator and author of this paper.

VITA

2011	B. S. in Mechanical Engineering, Sharif University of Technology, Tehran, Iran
2012	M. S. in Mechanical Engineering, University of California, San Diego
2016	Ph. D. in Engineering Sciences with a Specialization in Computational Science, University of California, San Diego

PUBLICATIONS

P. Beyhaghi, D. Cavaglieri, T.R. Bewley, “Delaunay-based derivative-free optimization via global surrogates, part I: linear constraints", *Journal of Global Optimization*, (2015): 1-52.

P. Beyhaghi, T.R. Bewley, “Delaunay-based derivative-free optimization via global surrogates, part II: convex constraints", *Journal of Global Optimization*, (2016): 1-33.

P. Beyhaghi, T.R. Bewley, “Implementation of Cartesian Lattice to improve Delaunay-based Optimization Algorithm", *Submitted to Journal of Global Optimization*, 2016

P. Beyhaghi, T.R. Bewley,“A derivative-free optimization algorithm for the efficient minimization of time-averaged statistics", *Submitted to SIAM Journal on Optimization*, 2016

P. Beyhaghi, D. Cavaglieri, and T.R. Bewley. "Delaunay-based Derivative-free Optimization via Global Surrogates (Δ-DOGS)." (2013).

G. Meneghello, P. Beyhaghi, T.R. Bewley, “Simulation-based optimization of the hydrofoil of a flying catamaran", *Submitted to Journal of Ocean Engineering*, 2016

D. Cavaglieri, P. Beyhaghi, and T. R Bewley. "Low-storage IMEX Runge-Kutta schemes for the simulation of Navier-Stokes systems." 21st AIAA computational fluid dynamics conference, San Diego, CA. 2013.

ABSTRACT OF THE DISSERTATION

Delaunay-based Global Optimization Algorithms and their Applications

by

Pooriya Beyhaghi

Doctor of Philosophy in Engineering Sciences with a Specialization in Computational Science

University of California, San Diego, 2016

Professor Thomas Bewley, Chair

A new class of derivative-free optimization algorithms is developed to solve, with remarkable efficiency, a range of practical nonconvex optimization problems whose function evaluations are computationally (or experimentally) expensive. These new algorithms, which are provably convergent under the appropriate assumptions, are response surface methods which iteratively minimize metrics based on an interpolation of existing datapoints and a synthetic model of the uncertainty of this interpolant, which itself is built on the framework of a Delaunay triangulation over the existing datapoints. Unlike other response surface methods, our algorithms can employ any well-behaved interpolation strat-

egy.

An important subproblem in α-DOGS is the estimation of the averaging process in stationary ergodic processes. A new approach for determining this quantity is also presented which is mathematically rigorous and numerically accurate. There are six main algorithms developed in this class thus far (only four of them are presented in this thesis) which address a wide range of practical optimization problems. The first algorithm, dubbed Δ-DOGS, efficiently minimizes expensive objective functions inside linearly-constrained feasible domains. The second algorithm, Δ-DOGS(C), extends Δ-DOGS to handle efficiently more general convex search domains. The third algorithm incorporates a grid into Δ-DOGS to achieve better convergence by performing fewer function evaluations at the boundary of feasibility. The fourth algorithm, dubbed α-DOGS, efficiently minimizes objective functions which are noisy, generally derived by taking the statistics of stationary and ergodic random processes. An important subproblem in α-DOGS is the estimation of the averaging process in stationary ergodic processes. A new approach for determining this quantity is also presented which is mathematically rigorous and numerically accurate.

Rigorous convergence analyses of all of the algorithms proposed are presented, and necessary conditions to guarantee convergence to the global minimum are provided. For validation, the algorithms proposed have been tested on both well-known benchmark optimization problems as well as some new application-oriented optimization problems in ship design; some illustrative results will be shown.

Chapter 1

Introduction

An important class of optimization problems is the simulated (or experiment)-based optimization problems whose objective or constrained functions are obtained by a computationally (or experimentally) expensive process. There are two main classes of numerical optimization algorithms. The first class is the Derivative-based methods which find a local solution to the optimization problem efficiently. Moreover, they can solve high dimensional problems; however, they have two main limitations: First, they can only find a local minimum of the objective function. Furthermore, the information of the derivative or a valid estimate for it is needed for their implementation. The second class is the Derivative-free methods that are suitable for those optimization problems that neither the derivative of $f(x)$ nor its accurate numerical approximation is readily available, as is the case when $f(x)$ is nonsmooth. This is common situations in which the function $f(x)$ is derived either from an experiment or many types of numerical simulations.

The goal of this thesis is to present a new class of derivative-free optimization algorithms which are mathematically rigorous and computationally efficient for low dimensional problems (n<10). These methods develop a framework to construct a new class of

methods which minimizes nonconvex and stochastic objective functions.

Chapter 2 introduces a new derivative-free optimization algorithm Δ-DOGS for nonconvex functions within a feasible domain bounded by linear constraints. Convergence to the global minimum is guaranteed for twice differentiable functions with bounded Hessian and is found to be remarkably efficient even for many functions which are not differentiable. Like other Response Surface Methods, at each optimization step, the algorithm minimizes a metric combining an interpolation of existing function evaluations and a model of the uncertainty of this interpolation. By adjusting the respective weighting of these two terms, the algorithm incorporates a tunable balance between global exploration and local refinement; a rule to adjust this balance automatically is also presented. Unlike other methods, any well-behaved interpolation strategy may be used. The uncertainty model is built upon the framework of a Delaunay triangulation of existing datapoints in parameter space. A quadratic function which goes to zero at each datapoint is formed within each simplex of this triangulation; the union of each of these quadratics forms the desired uncertainty model. Care is taken to ensure that function evaluations are performed at points that are *well situated* in parameter space; that is, such that the simplices of the resulting triangulation have circumradii with a known bound. This facilitates well-behaved local refinement as additional function evaluations are performed.

Chapter 3 present a modification for Δ-DOGS (Δ-DOGS(C)) that solve that optimization that the feasible domain is nonlinearly constrained (but convex) domain. Unlike Δ-DOGS, the initialization step of Δ-DOGS(C) only choose $n + 1$ feasible datapoints instead of all vertices of the feasible domain. Therefore, the convex hull of the initial points does not cover the whole feasible domain. In this way, as the algorithm proceeds, additional feasible datapoints are added in such a way that the convex hull of the available datapoints

efficiently increases towards the boundaries of the feasible domain. Similar to Δ-DOGS, at each step of the algorithm, a search function is defined based on an interpolating function which passes through all available datapoints and a synthetic uncertainty function which characterizes the distance to the nearest datapoints. This uncertainty function, in turn, is built on the framework of a Delaunay triangulation, which is itself based on all available datapoints together with the (infeasible) vertices of an exterior simplex which completely contains the feasible domain. The search function is minimized within those simplices of this Delaunay triangulation that do not include the vertices of the exterior simplex. If the outcome of this minimization is contained within the circumsphere of a simplex which includes a vertex of the exterior simplex, this new point is projected out to the boundary of the feasible domain. For problems in which the feasible domain includes edges (due to the intersection of multiple twice-differentiable constraints), a modified search function is considered in the vicinity of these edges to assure convergence.

Chapter 4 introduces another modification of the algorithm of chapter 2 which reduces the number of function evaluation on the boundary of feasibility as compared with the original Δ-DOGS. One of the problems associated with the basic Δ-DOGS is the poor behavior of the generated uncertainty function near the boundary of feasibility which leads to significantly more function evaluations along the boundary of feasibility that might not be necessary. To address this issue, another search function is introduced which has an improved behavior near the boundary of search domain. Additionally, the data points are quantized on the Cartesian grid over the search domain, both of which lead to a reduced number of datapoint accumulating on the boundary of feasibility, and faster overall convergence.

Chapter 5, extends Δ-DOGS to solve efficiently those problems for which the pro-

cess of performing a function evaluation is inaccurate, but the accuracy which can be improved by increasing the cost (e.g, the averaging time) associated with function evaluation. The particular case focused on in our work is when the true objective function is the infinite time-averaged value of a statistic of a stationary ergodic process. Most existing derivative-free optimization algorithms that are implemented for minimizing problems with noisy function evaluations use estimates with the same level of accuracy for all data points that are considered in the optimization process. In this chapter, we developed a unique framework that employs the function evaluations with various amount of accuracy at different points in the feasible domain. These variations in the accuracy of the estimation processes remarkably reduce the total cost of the optimization process. Furthermore, the proof of convergence to the global minimum under specific conditions is established.

An important subproblem of α-DOGS is the uncertainty quantification (UQ) of the averaging process of a stationary ergodic random processes. In Chapter 6, we develop an unbiased framework to quantify the uncertainty of such averaging process. The mathematical properties of this estimator is analyzed, and its behavior is compared with the existing methods on both synthetic dataset as well as data obtained from turbulence simulations.

Finally, in Chapter 7, one of the optimization algorithm developed here, is applied to the hydrofoil of a racing catamaran, with the objective of maximizing its lift/drag ratio at a fixed working condition, illustrating the effective implementation of the present work on an application of significant engineering interest.

1.1 Organization of the thesis

The content of this thesis may be summarized as follows: Chapter 2 presents Δ-DOGS as a global optimization algorithm which can minimize any nonconvex objective

function inside a linearly constrained domain.

Chapter 3 presents Δ-DOGS(C), which extent Δ-DOGS to convex nonlinear constrained problems.

Chapter 4 addresses one of the performance issues associated with the original Δ-DOGS which leads to a significant amount of improvement in the speed of convergence.

Chapter 5 develops an optimization algorithm, dubbed as α-DOGS, which minimizes those objective functions that are obtained by taking the infinite time-averaged statistics of a stationary ergodic process.

Chapter 6 addresses an important subproblem in α-DOGS, quantifying the uncertainty associated with time of stationary ergodic random processes.

Finally, Chapter 7 implements Δ-DOGS on a problem of hydrofoil design.

Chapter 2

Delaunay-based Optimization for Linearly constrained domain: Δ-DOGS

2.1 Introduction

In this chapter, a new derivative-free optimization algorithm is presented to minimize a (possibly nonconvex) function subject to linear constraints on a bounded feasible region in parameter space[1]:

$$\text{minimize } f(x) \text{ with } \ x \in L = \{x | Ax \leq b\}, \tag{2.1a}$$

[1]Taking a and b as vectors, $a \leq b$ implies that $a_i \leq b_i \ \forall i$.

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and L is assumed to be bounded. A special case of this problem, with simpler "box" constraints, is also considered:

$$\text{minimize } f(x) \text{ with } \; x \in L_{\text{box}} = \{x | a \leq x \leq b\}, \tag{2.1b}$$

where $a, b \in \mathbb{R}^n$. Derivative-free algorithms are well suited for such problems even if neither the derivative of $f(x)$ nor its accurate numerical approximation is readily available, as is the case when $f(x)$ is nonsmooth. This is common in situations in which the function $f(x)$ is derived either from an experiment or from many types of numerical simulations.

An important class of derivative-free algorithms, dating back to the 1960s, is Direct Search Methods, as reviewed in [1]. An early and famous algorithm in this class is the Nelder-Mead simplex algorithm, variations of which are implemented in several numerical optimization packages. This method is examined in, e.g., [2]. Another category of Direct Search Methods, called Adaptive Direction Search Algorithms, includes the Rosenbrock [3] and Powell [4] methods. More modern methods in this class, dubbed Pattern Search Methods, are characterized by a series of exploratory moves on a regular pattern of points in parameter space called a grid (often, the Cartesian grid is used); the Generalized Pattern Search (GPS) is a typical example. The efficiency and convergence of such algorithms is examined in [5] and [6].

In general, Direct Search Methods identify a local minimum of a function from some initial guess in parameter space. The harder problem of attempting to identify accurately the global minimum of a nonconvex function $f(x)$, with as few function evaluations as possible, is an issue of significant interest.

Response Surface Methods employ an underlying (inexpensive-to-compute, differentiable) model of the actual (expensive-to-compute, possibly nondifferentiable) function

of interest in order to summarize the trends evident in the available datapoints, at which the function has already been evaluated, over the entire feasible region in parameter space as the iteration proceeds. The Trust Region method is one of the first optimization algorithms appearing in the literature which uses such a model; however, the model used by this method does not use all of the available datapoints at each step. Other Response Surface Methods, such as the Expected Improvement algorithm [7], typically use all available datapoints to build an inexpensive and useful model (often called a "surrogate") of the actual function of interest. An insightful review of global optimization methods based on such surrogate functions is given by [8].

The most popular surrogate function used in such global optimization schemes is the Kriging method [9], [10], [11], which inherently builds both an estimate of the function itself, $p(x)$, as well as a model of the uncertainty of this estimate, $e(x)$, over the entire feasible domain of parameter space. With this interpolation strategy, the function is modeled as a Gaussian random variable at every point within the feasible domain of parameter space. This stochastic model is constructed carefully, such that the variance of the random variable is zero, and the expected value of the random variable is equal to the (known) function value at each datapoint available in parameter space. Away from the datapoints, the expected value of the random variable in the Kriging model effectively interpolates the known function values, and the variance of the random variable is greater than zero, effectively quantifying the distance in parameter space to the nearest available datapoints. As eloquently described in [8], the estimate $p(x)$ and the uncertainty of the estimate, $e(x)$, provided by this model may be used together to identify a point in the feasible domain with a high probability of a reduced function value. A particularly efficient algorithm for global optimization is the Surrogate Management Framework (SMF; see [12]), which combines

the Expected Improvement algorithm with a Generalized Pattern Search. This algorithm was significantly extended in [13], in which the search is coordinated by a grid derived from a dense sphere packing, with significantly improved uniformity of grid points over parameter space as compared with the Cartesian grid, in order to accelerate convergence.

The Kriging interpolation strategy has various shortcomings, the most significant of which is the numerical stiffness of the computational problem of fitting the Kriging model to the datapoints, and the subsequent inaccuracy of this fit. This problem is exacerbated when there are many datapoints available, some of which are clustered in a small region of parameter space, as illustrated in Appendix 2.4. Furthermore, both the computation of the Kriging interpolant itself, as well as the minimization over the feasible region of parameter space of the search function based on this interpolant, are nonconvex optimization problems; both of these problems must be solved with another global optimization algorithm, which represents a sometimes significant computational expense.

As discussed above, modern Response Surface Methods need both an estimate of the function itself as well as a model of the uncertainty of this estimate over the entire feasible domain of parameter space. Most interpolation methods, other than Kriging, don't provide this. For the specific case of interpolation with radial basis functions, an uncertainty function has been proposed and used by [14].

The Response Surface Method proposed in this work is innovative in the way it facilitates the use of *any* well behaved interpolation strategy that the user might favor for the particular problem under consideration. [In the present work, our numerical examples use polyharmonic spline interpolation, which is reasonably well behaved even when the available datapoints are clustered in various regions of parameter space; this interpolation strategy is fairly standard, though other interpolation schemes could easily be used in its place.]

To accomplish this, the present work proposes an artificially-generated function modeling the "uncertainty" of the interpolant based on the distance to the nearest datapoints. This uncertainty model is built directly on the framework of a Delaunay triangulation of the available datapoints in parameter space.

The structure of the chapter is as follows. Section 2.2 discusses how the present algorithm may be initialized. Section 5.2 then proposes a simple strawman form of the algorithm based on the present ideas, laying out the essential elements of the final algorithm and analyzing its various properties, including a proof of convergence under the conditions that (a) the underlying function of interest $f(x)$ has bounded Lipshitz norm, and (b) the maximum circumradii of the simplicies in the triangulations are bounded as the algorithm proceeds. This simple strawman form of the optimization algorithm, however, fails to ensure condition (b). Section 2.5 modifies the strawman form of the optimization algorithm proposed previously by, when necessary, adjusting the points in parameter space at which new function evaluations are performed, thereby ensuring condition (b). Section 2.6 presents a rule to adjust the parameter which tunes the balance between global exploration and local refinement as the iteration proceeds. Section 2.7 addresses how the algorithm may be modified to run efficiently using parallel computations. In Section 7.4, the algorithm proposed is applied to a select number of test functions in order to illustrate its behavior. Some conclusions are presented in Section 7.5.

2.2 Initialization

The optimization algorithm developed in this chapter is initialized as shown in Algorithm 2.1.

as described in detail in the remainder of §2.2. The optimization algorithm developed in

Algorithm 2.1 Initialization of Δ-DOGS

1: perform function evaluations at all of the vertices of the feasible domain L,
2: remove all redundant constraints from the rows of $Ax \leq b$, and
3: project out any equality constraints implied by multiple rows of $Ax \leq b$; in other words, we project the feasible domain onto the lower dimensional space that satisfies the equality constraints.

later sections then builds a Delaunay triangulation within the convex hull of the available function evaluations, which coincides with the feasible domain itself, and incrementally updates this Delaunay triangulation at each new datapoint (that is, at each new feasible point $x \in L$ at which $f(x)$ is computed as the iteration proceeds). This approach is justified by the following result, which is proved in [15]:

Theorem 1. *The convex hull of the vertices of a bounded domain L constrained such that $Ax \leq b$ is equivalent to the domain L itself.*

Due to the simplicity of step (A) of Algorithm 2.1, this step is recommended for most low-dimensional problems. In high-dimensional problems bounded by many linear constraints, however, the feasible domain might have a lot of vertices, and it might be unnecessarily expensive to follow such an approach; in such cases, chapter 3 of this work demonstrates how this initialization step may be cleverly sidestepped.

In the case of box constraints, (2.1b), step (A) of Algorithm 2.1 corresponds to 2^n function evaluations which are trivial to enumerate.

In the more general case of linear constraints, (2.1a), identifying the vertices of the feasible domain is slightly more involved. We proceed as follows:

Definition 1. *The* **active set** *of the constraints $Ax \leq b$ at a given point $\hat{x} \in R^n$ in parameter space, denoted $A_{a(\hat{x})}\, \hat{x} = b_{a(\hat{x})}$, is given by those constraints (that is, by those rows of $Ax \leq b$) that hold with equality at $\hat{x}$. A* **feasible point** *$\hat{x}$ (satisfying $A\hat{x} \leq b$) is called a* **vertex** *of the*

feasible domain *(that is, the set of all $x \in \mathbb{R}^n$ such that $Ax \leq b$) if rank$(A_{a(\hat{x})}) = n$.*

A simple brute-force method to find all of the vertices of the feasible domain then follows:

1) Check the rank of all $\binom{m}{n}$ $n \times n$ linear systems that may be chosen from the $m > n$ rows of $Ax \leq b$.
2) For those linear systems in step 1 that have rank n, solve $A_{a(\hat{x})}\, \hat{x} = b_{a(\hat{x})}$.
3) For each solution found in step 2, check to see if $A\hat{x} \leq b$; if this condition holds, it is a vertex.

The set of points thus generated is then scrutinized to eliminate duplicates. This brute-force method is tractable only in relatively low-dimensional problems (note that most problems that are viable candidates for derivative-free optimization are, in fact, fairly low-dimensional). The number of vertices is typically much less than the number of linear systems considered by this method; for example, $m = 20$ constraints in $n = 10$ dimensions requires us to examine $184,756$ $n \times n$ matrices in step 1, and would typically result in roughly $M \sim O(10^3)$ vertices.

Finding the M vertices of an n-dimensional polyhedron is a well-known problem in convex analysis; see, e.g., [16], [17] and [18]. These papers suggest a somewhat more involved yet significantly more computationally efficient iterative procedure, based on the simplex method, to find the vertices of the feasible domain in problems that are high-dimensional and/or have many linear constraints. With this approach, a pivot operation is used to move from one vertex of the feasible domain to its neighbors (the vertex v_1 and v_2 are called neighbors if their active sets differ in exactly one row). The number of linear solves required by this approach is $O(nM)$.

Step (B) of Algorithm 2.1 then removes all redundant constraints given by the redundant rows of $Ax \leq b$. Each row of $Ax \leq b$ is checked at each vertex of the feasible

domain. Those rows that are not satisfied as equalities at at least n distinct vertices are eliminated, as they do not play a role in defining an $(n-1)$-dimensional face of the feasible domain. Of the rows that remain, the rows of the augmented matrix $\left[\begin{array}{cc} A & b \end{array}\right]$ that are multiples of other rows are also eliminated, as they define identical faces.

Finally, step (C) of Algorithm 2.1 projects out all equality constraints in the problem formulation, as algorithms for the construction of an n-dimensional Delaunay triangulation will encounter various problems if the feasible domain actually has dimension less than n. In the case of (2.1a), equality constraints may easily be found and projected out, resulting in a lower-dimensional optimization problem. To illustrate, consider $\{x_1, x_2, \ldots, x_M\}$ as the set of vertices of the feasible domain of x, computed as described above. Define the $n \times (M-1)$ matrix C as follows:

$$C = \left[\begin{array}{cccc} (x_1 - x_2) & (x_1 - x_3) & \cdots & (x_1 - x_M) \end{array}\right]. \tag{2.2}$$

The rank r of the matrix C is the rank of the optimization problem at hand. If $r < n$, there are one or more equality constraints to contend with. In this case, taking the reduced $\underline{QR}$ decomposition $C = \underline{QR}$, the r linearly-independent columns of $\underline{Q}$ provide a new basis in which the optimization problem may be written. Defining $x = x_1 + \underline{Q}\,\underline{x}$, a new r-dimensional optimization problem is posed in the space of $\underline{x} \in R^r$, and the feasible domain of $\underline{x}$ is defined by $(A\underline{Q})\underline{x} \leq b - Ax_1$.

2.3 Strawman form of algorithm

In this part, the essential basis of the optimization algorithm is presented. The general framework of Δ-DOGS is presented in 2.2. The main novelty of this algorithm

Algorithm 2.2 Strawman of Δ-DOGS

1: Prepare the problem for optimization by executing Algorithm 2.1, as described in §2.2. Assume that the resulting optimization problem is n dimensional, and that the feasible domain L has M vertices. Then, proceed as follows:
2: Take the set of initialization points S^0 as all M of the vertices of the feasible domain L together with one or more user-specified points of interest on the interior of L.
3: Evaluate the function $f(x)$ at each of these initialization points. Set $k = 0$.
4: Calculate (or, for $k > 0$, update) an appropriate interpolating function $p^k(x)$ through all points in S^k.
5: Calculate (or, for $k > 0$, update) a Delaunay triangulation Δ^k over all of the points in S^k.
6: For each simplex Δ_i^k of the triangulation Δ^k:
   a. Calculate the circumcenter z_i^k and the circumradius r_i^k of the simplex Δ_i^k.
   b. Define the **local uncertainty function**

$$e_i^k(x) = (r_i^k)^2 - \|x - z_i^k\|^2. \tag{2.3}$$

   c. Define the **local search function**

$$s_i^k(x) = p^k(x) - K\, e_i^k(x). \tag{2.4}$$

   d. Minimize the local search function $s_i^k(x)$ within Δ_i^k.
7: Find the smallest of all of the local minima identified in step 6d. Evaluate $f(x)$ at this new datapoint x_k, and set $S^{k+1} = S^k \cup \{x_k\}$. Increment k and repeat from step 4 until convergence.

is the new construction of the uncertainty function. The local uncertainty functions $e_i^k(x)$ model the uncertainty in the unexplored regions within each simplex of the triangulation Δ^k at step k. As discussed in §2.3.1, the union of these simplices coincides precisely with feasible domain of parameter space. The **global uncertainty function** $e^k(x)$ and the **global search function** $s^k(x)$ are defined over the feasible domain as $e_i^k(x)$ and $s_i^k(x)$, respectively, within each simplex Δ_i^k. Note that $e^k(x)$ reaches zero by construction at each datapoint, and $e^k(x)$ reaches a maximum within each simplex as far from all of the available datapoints as possible; it is shown in §2.3.2 that $e^k(x)$ is Lipschitz. In §2.3.3, a method of simplifying the searches performed in step 6d of Algorithm 2.2 is discussed.

The (single, constant) tuning parameter K specifies the trade-off in Algorithm 2.2 between global exploration (which is emphasized for large K) and local refinement (which is emphasized for small K). In §2.3.4, global convergence of Algorithm 2.2 is proved for functions $f(x)$ with bounded Lipshitz norm, assuming sufficiently large K and boundedness of the circumradii of the triangulation generated by Algorithm 2.2. In §2.5, a small but technically important modification of the Algorithm 2.2 is introduced which guarantees boundedness of the circumradii of the triangulation generated as the iteration proceeds.

2.3.1 Characterizing the triangulation

The uncertainty function in Algorithm 2.2 is built on the framework of a Delaunay triangulation of the feasible domain with, in a certain sense, maximally regular simplices, which we now characterize.

Definition 2. *Consider the* $(n+1)$ *vertices* $\{V_0, V_1, \ldots, V_n\} \in \mathbb{R}^n$ *such that the vectors* $(V_0 - V_1), (V_0 - V_2), \ldots, (V_0 - V_n)$ *are linearly independent. The convex hull of these vertices is called a* **simplex** *(see, e.g., [19, p. 32]). Associated with this simplex, the* **circumcenter**

z is the point that is equidistant from all $n+1$ vertices, the **circumradius** *r is the distance between z and any of the vertices V_i, and the* **circumsphere** *is the set of all points within a distance r from z.*

Lemma 1. *For any simplex, the circumcenter is unique.*

Proof. Assume z is equidistant from $V_0, \ldots, V_n$, i.e.

$$\|V_0 - z\| = \|V_1 - z\| = \cdots = \|V_n - z\|$$

For $i = 1, \ldots, n$, simplification leads to:

$$\begin{aligned} V_0^2 - 2V_0^T z &= V_i^2 - 2V_i^T z \\ \Rightarrow \quad 2\,(V_0 - V_i)^T z &= V_0^2 - V_i^2. \end{aligned}$$

Thus, z is equidistant from all vertices if

$$2\begin{bmatrix} (V_0 - V_1)^T \\ \vdots \\ (V_0 - V_n)^T \end{bmatrix} z = \begin{bmatrix} V_0^2 - V_1^2 \\ \vdots \\ V_0^2 - V_n^2 \end{bmatrix}. \tag{2.5}$$

This system has a unique solution if the matrix on the LHS is nonsingular; which follows from the linear independence of $(V_0 - V_1), (V_0 - V_2), \ldots, (V_0 - V_n)$ in Definition 2. □

The two following definitions are taken from [20].

Definition 3. *If S is a set of points in $\mathbb{R}^n$, a* **triangulation** *of S is a set of simplices whose vertices are elements of S such that the following conditions hold:*

- *Every point in S is a vertex of at least one simplex in the triangulation. The union of all of these simplices fully covers the convex hull of S.*

- *The intersection of two different simplices in the triangulation is either empty or a k-simplex such that $k = 0, 1, \ldots, n-1$. For example, in the case of $n = 3$ dimensions, the intersection of two simplices (in this case, tetrahedra) must be an empty set, a vertex, an edge, or a triangle.*

Definition 4. *A* **Delaunay triangulation** *is a triangulation (see Definition 3) such that the intersection of the open circumsphere around each simplex with S is empty. This special class of triangulation, as compared with other triangulations, has the following properties:*

- *The maximum circumradius among the simplices is minimized.*

- *The sum of the squares of the edge lengths weighted by the sum of the volumes of the elements sharing these edges is minimized.*

Delaunay triangulations exhibit an additional property which makes them essential in Algorithm 2.2. By the definitions of $e_i^k(x)$ and $e^k(x)$ above, it follows that $e_i^k(x) = e^k(x)$ within the simplex Δ_i^k. The following may be established *if* the triangulation Δ^k is Delaunay:

Lemma 2. *Assume the triangulation Δ^k is Delaunay. For any i and any feasible point $x \in L$, $e^k(x) \geq e_i^k(x)$.*

Proof. By Theorem 1 and Definition 3, since $x \in L$, a simplex Δ_j^k exists which contains x (that is, $x \in \Delta_j^k$). We must show that, for all $i \neq j$, $e_i^k(x) \leq e_j^k(x)$. By construction, $e_j^k(x) = 0$ at the vertices of simplex Δ_j^k; since the triangulation is Delaunay (see Definition 4), these vertices are not inside the circumsphere of the simplex Δ_i^k. Thus, $e_i^k(x) \leq 0$ at the vertices

of simplex Δ_j^k. It follows simply from the definition of $e_i^k(x)$ that, for all $x \in L$,

$$e_i^k(x) - e_j^k(x) = (r_i^k)^2 - (r_j^k)^2 - |z_i^k|^2 + |z_j^k|^2 + 2(z_i^k - z_j^k)^T x;$$

that is, $e_i^k(x) - e_j^k(x)$ is a linear function of x. Since $e_i^k(x) - e_j^k(x) \leq 0$ at the vertices of simplex Δ_j^k, it follows that $e_i^k(x) - e_j^k(x) \leq 0$ everywhere within simplex Δ_j^k. □

Remark 1. *Lemma 2 holds only for Delaunay triangulations, not arbitrary triangulations. Lemma 2 is used in §2.3.3 to simplify the searches performed in step 6d of Algorithm 2.2.*

Remark 2. *In step 3a of Algorithm 2.2, linear systems of the form given in* (2.5) *must be solved in order to find the circumcenter of each simplex. The use of Delauney triangulations improves the accuracy of these numerical solutions. If the ratio between the circumradius and the maximum distance between two edges of a simplex is large, this system is ill conditioned. Delaunay triangulations (see Definition 4) minimize the maximum circumradius of the simplices in the triangulation, thereby minimizing the worst-case ill conditioning of the linear systems of the form given in* (2.5) *that need to be solved.*

The determination of Delaunay triangulations is a benchmark problem in computational geometry, and a large number of algorithms have been proposed; extensive reviews are given in [21] and [20]. Qhull [2], Hull [3], and CGAL-DT [4] are among the most commonly-used approaches today for computing Delaunay triangulations in moderate dimensions. In the present work, a Delaunay triangulation must be performed over a set of initial evaluation points, then updated at each iteration when a new datapoint is added. Hence, the incremental method originally proposed in [22] is particularly appealing. The New-DT

[2] http://www.qhull.org. Accessed 29 June 2016
[3] http://netlib.org/voronoi/hull.html. Accessed 3 June 2016
[4] http://www.cgal.org. Accessed 3 June 2016

and Del-graph algorithms (see [23] and [24]) are the leading, memory-efficient implementations of this incremental approach; the present work implements the Del-graph algorithm.

The most expensive step of Algorithm 2.2, apart from the function evaluations, is the minimization of $s_j^k(x)$ (in step 6d) in each simplex Δ_j^k. The cost of this step is proportional to the total number of simplices S in the Delaunay triangulation. As derived in [25], a worst-case upper bound for the number of simplices in a Delaunay triangulation is $S \sim O(N^{\frac{n}{2}})$, where N is the number of vertices and n is the dimension of the problem. As shown in [26] and [27], for vertices with a uniform random distribution, the number of simplices is $S \sim O(N)$.

2.3.2 Smoothness of the uncertainty

We now characterize precisely the smoothness of the uncertainty function proposed in Algorithm 2.2.

Lemma 3. *The function $e^k(x)$ is C_0 continuous.*

Proof. Consider a point x on the boundary between two different simplices Δ_i^k and Δ_j^k with circumcenters z_i^k and z_j^k and local uncertainty functions $e_i^k(x)$ and $e_j^k(x)$. By Definition 3, the intersection of Δ_i^k and Δ_j^k, when it is nonempty, is another simplex of lower dimension, denoted here simply as Δ. The projection of z_i^k and z_j^k on the lower-dimensional hyperplane that contains Δ is by construction its circumcenter, denoted here as z. Thus, the lines from z_i^k to z and from z_j^k to z are perpendicular to the simplex Δ. Now consider x_Δ as one of the vertices of the simplex Δ. Some trivial analysis of the triangles $z_i^k - x - z$ and $z_i^k - x_\Delta - z$

give:

$$e_i^k(x) = \|z_i^k - x_\Delta\|^2 - \|z_i^k - x\|^2,$$
$$\|z_i^k - x_\Delta\|^2 = \|z_i^k - z\|^2 + \|z - x_\Delta\|^2,$$
$$\|z_i^k - x\|^2 = \|z_i^k - z\|^2 + \|z - x\|^2.$$

Combining these three equations gives

$$e_i^k(x) = \|z - x_\Delta\|^2 - \|z - x\|^2.$$

By similar reasoning, we obtain

$$e_j^k(x) = \|z - x_\Delta\|^2 - \|z - x\|^2.$$

Hence, $e_i^k(x) = e_j^k(x)$ for all $x \in \Delta$ (that is, at the interface of simplices Δ_i^k and Δ_j^k). □

The continuity of $e^k(x)$ is illustrated in 2D in Figure 2.1, where two neighboring simplices (triangles) with vertices {(0.2, 0), (0, 1), (1, 1)} and {(0, 1), (1, 1), (0.5, 2)} are represented.

We now establish a stronger property, that the uncertainty function is Lipschitz.

Lemma 4. *The function $e^k(x)$ generated by Algorithm 2.2 at the k^{th} iteration is Lipschitz within the convex polyhedron L, with a Lipschitz constant of r_{max}^k, where r_{max}^k is the maximum circumradius of the triangulation Δ^k.*

Proof. We first show that $e^k(x)$ is Lipschitz inside each simplex; we then show that $e^k(x)$ is Lipschitz everywhere.

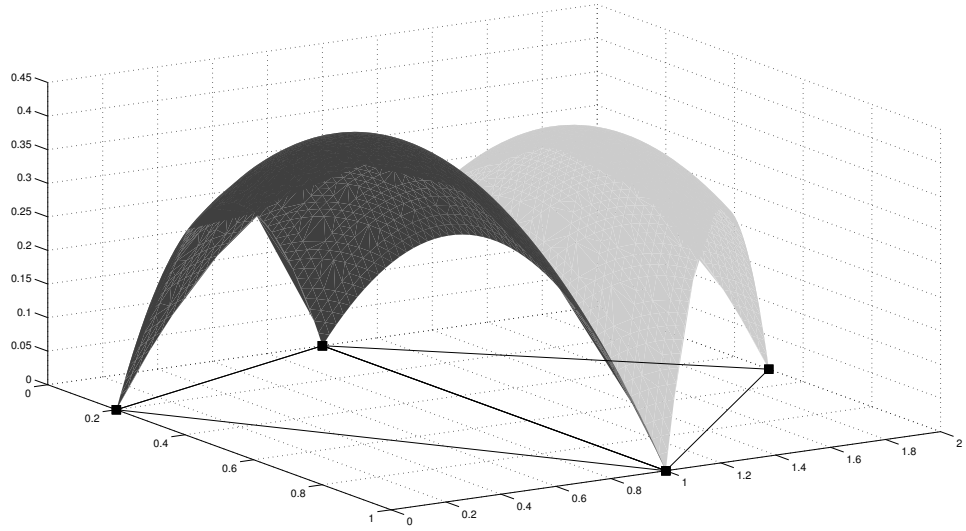


Figure 2.1: The uncertainty function $e^k(x)$ over two neighboring simplices in two dimensions.

Assume first that x_1 and x_2 are inside the simplex Δ_i^k, with circumcenter z_i^k and circumradius r_i^k. By (3.14), we have

$$e^k(x_1) - e^k(x_2) = \|x_2 - z_i^k\|^2 - \|x_1 - z_i^k\|^2.$$

Now, assume that z^* is a projection of z_i^k along the line from x_1 to x_2, and that x_M is the midpoint between x_1 and x_2. It follows that

$$\begin{aligned} |\,\|x_2 - z_i^k\|^2 - \|x_1 - z_i^k\|^2\,| &= |\,\|x_2 - z^*\|^2 - \|x_1 - z^*\|^2\,| \\ &= 2\,\|z^* - x_M\|\|x_1 - x_2\|. \end{aligned}$$

Since

$$\|z^* - x_M\| \leq \max\left(\|z^* - x_1\|,\, \|z^* - x_2\|\right) \leq r_i^k,$$

we have

$$|e^k(x_1) - e^k(x_2)| \leq 2\, r_i^k\, ||x_1 - x_2||. \tag{2.6}$$

Thus, the uncertainty function $e^k(x)$ is Lipschitz inside each simplex.

In order to prove that $e^k(x)$ is Lipschitz over the entire feasible domain L, consider now x_1 and x_2 as two arbitrary points inside L, and define a series of points $t_1,\ t_2,\ \ldots,\ t_m$ on the line segment between x_1 to x_2 such that $t_1 = x_1$ and $t_m = x_2$, and such that each couple (t_i, t_{i+1}) lies within the same simplex, with circumradius r_i. In other words, each point t_i for $1 < i < m$ must be at the interface between two neighboring simplices along the line from $t_1 = x_1$ to $t_m = x_2$. In this framework, we have

$$|e^k(x_1) - e^k(x_2)| \leq \sum_{i=1}^{m-1} |e^k(t_i) - e^k(t_{i+1})|.$$

Since t_i and t_{i+1} are in the same simplex, (2.6) gives

$$|e^k(t_i) - e^k(t_{i+1})| \leq 2\, r_i^k\, ||t_i - t_{i+1}||.$$

Since $t_1,\ t_2,\ \ldots,\ t_m$ lie along the same line, we have

$$||t_1 - t_m|| = \sum_{i=1}^{m-1} ||t_i - t_{i+1}||.$$

Combining these three equations, we have

$$\begin{aligned} |e^k(x_1) - e^k(x_2)| &\leq 2 \max_{1 \leq i \leq m-1} (r_i^k)\, ||x_1 - x_2|| \\ &\leq 2\, r_{\max}^k\, ||x_1 - x_2||. \end{aligned}$$

□

2.3.3 Minimizing the search function

At iteration k of Algorithm 2.2, the search function $s^k(x) = p^k(x) - Ke^k(x)$ must be minimized over $x \in L$. Recall that, within each simplex Δ_i^k in the triangulation, the uncertainty function $e^k(x)$ is defined by $s_i^k(x) = p^k(x) - Ke_i^k(x)$ for $x \in \Delta_i^k$. In order to minimize $s^k(x)$ over the entire feasible domain L, the minima $x_{\min,i}^k$ must first be found within each simplex Δ_i^k as follows:

$$x_{\min,i}^k = \operatorname{argmin}_{x \in \Delta_i^k} s_i^k(x); \tag{2.7a}$$

the global minimum $x_{\min}^k = \operatorname{argmin}_{x \in L} s^k(x)$ is then given by $x_{\min}^k = x_{\min,i_{\min}}^k$ where

$$i_{\min} = \operatorname{argmin}_{i \in \{1,\ldots,S^k\}} \left[s_i^k(x_{\min,i}^k) \right], \tag{2.7b}$$

where S^k is the number of simplices in the triangulation Δ^k. In other words, in order to find $x_{\min}^k$, we must first solve S^k nonconvex optimization problems with linear constraints $x \in \Delta_i^k$. This computational task is significantly simplified by following result.

Lemma 5. *If the linear constraints $x \in \Delta_i^k$ in the optimization problems defined in* (2.7a) *are relaxed to the entire feasible domain, $x \in L$, the resulting value of x_{min}^k remains unchanged.*

Proof. By Lemma 2, for any feasible point $x \in L$ and for any i such that $1 \leq i \leq S^k$, $e^k(x) \geq e_i^k(x)$. More precisely,

$$e^k(x) = \max_{i \in \{1,\ldots,S^k\}} \left[e_i^k(x) \right].$$

Since K is a positive real number,

$$s^k(x) = p^k(x) - K\, e^k(x) = \min_{i \in \{1,\ldots,S^k\}} \left[p^k(x) - K\, e_i^k(x) \right].$$

By the definition of $x^k_{\min}$

$$s^k(x^k_{\min}) = \min_{x \in L} \left[s^k(x) \right].$$

Combining the above equations and swapping the order of the minimization gives

$$\begin{aligned} s^k(x^k_{\min}) &= \min_{x \in L} \min_{i \in \{1,\ldots,S^k\}} \left[p(x) - K\, e_i^k(x) \right] \\ &= \min_{i \in \{1,\ldots,S^k\}} \min_{x \in L} \left[p(x) - K\, e_i^k(x) \right]. \end{aligned}$$

It can be observed that $\min_{x \in L}[p(x) - K\, e_i^k(x)]$ is just the optimization problem (2.7a) with the linear constraint $x \in \Delta_i^k$ relaxed to $x \in L$; thus, when this constraint is relaxed in this manner in (2.7), the resulting value of $s^k(x^k_{\min})$ remains unchanged. □

At each iteration k, we thus seek to minimize the local search function $s_i^k(x)$ for $x \in L$ for each $i \in \{1, \ldots, S^k\}$; if for a given i the minimizer of $s_i^k(x)$ lies outside of Δ_i^k, that minimizer is not the global minimum of $s^k(x)$. Hence, we may terminate and reject any such search as soon as it is seen that the minimizer of $s_i^k(x)$ lies outside of Δ_i^k.

Using a local optimization method with a good initial guess within each simplex, the local minimum of $s_i^k(x)$ within the simplex Δ_i^k, if it exists, can be found relatively quickly. For the smooth local search functions $s_i^k(x)$ we use in the present work, we have analytic expressions for both the gradient and the Hessian; these expressions play a valuable role in the local minimization of these functions.

Recall that the local search functions $s_i^k(x)$ considered in Algorithm 2.2 are linear

combinations of the local uncertainty functions $e_i^k(x)$ and the user's interpolation function of choice, $p^k(x)$. The local uncertainty function $e_i^k(x)$ is a quadratic function whose gradient and Hessian are

$$\nabla e_i^k(x) = -2\,(x - x_{s_i}), \qquad \nabla^2 e_i^k(x) = -2\,I.$$

For the polyharmonic spline interpolation method used in the present numerical implementation, analytical expressions for the gradient and Hessian of the interpolation function are derived in the appendix. If a different interpolation strategy is used, for the purpose of the following discussion, we assume that analytical expressions for the gradient and the Hessian of the interpolation function are similarly available.

In order to locally minimize the function $s_i^k(x)$ within each simplex, a good initial guess of the solution is valuable. To generate such an initial guess analytically, consider the result of a simplified optimization problem obtained by implementing piecewise linear interpolation of the datapoints at the vertices of the simplex Δ_i^k, together with the local uncertainty function $e_i^k(x)$. Following this approach, we rewrite the coordinates of a point inside the simplex Δ_i^k as a linear combination of its vertices:

$$x = X_i\,w,$$

where X_i is an $n \times (n+1)$ matrix whose columns are the coordinates of the $n+1$ vertices of the simplex Δ_i^k, and w is an $(n+1)$-vector with components w_j that form a partition of unity; that is,

$$\sum_{j=1}^{n+1} w_j = 1, \qquad w_j \geq 0 \quad j = 1, 2, \ldots, n+1,$$

which may be written compactly in matrix form as

$$\begin{bmatrix} 1 & \dots & 1 \end{bmatrix} w = 1, \qquad -Iw \leq 0. \tag{2.8}$$

In each simplex Δ_i^k, we thus minimize a new search function $\overline{s}_i^k(w)$ defined as

$$\begin{aligned} \overline{s}_i^k(w) &= Y_i\, w - K\left[R_i^2 - (X_i\, w - z_i^k)^T (X_i\, w - z_i^k)\right] \\ &= K\, w^T\, X_i^T X_i\, w + (Y_i - 2K(z_i^k)^T X_i)\, w \\ &\quad + K\,[(z_i^k)^T z_i^k - R_i^2]. \end{aligned} \tag{2.9}$$

where Y_i is an $1 \times (n+1)$ row vector whose elements are the function values at the $n+1$ vertices of the simplex Δ_i^k. Minimization of (2.9), subject to the constraints (2.8), can be performed exceptionally quickly using convex quadratic programming. This optimization gives a vector of weights w_0, which defines the initial guess for the local minimization of the function $s_i^k(x)$ within the simplex Δ_i^k. Since we have analytic expressions for the gradient and Hessian of $s_i^k(x)$, and a good initial guess of its minimum, we can apply either a Trust Region method, or Newton's method with Hessian modification, in order to find quickly the minimum of $s_i^k(x)$. Newton's method is a line search algorithm with a descent direction derived based on both the gradient and the Hessian of the function; because of the nonconvexity of $s_i^k(x)$, Hessian modification is required to ensure convergence. The Hessian modification that has been used in our numerical code is modified Cholesky factorization [28], and the line search algorithm used is a backtracking line search algorithm (Algorithm 3.1 in [29]). Convergence of this local optimization algorithm is proved in [29].

2.3.4 Convergence of Algorithm 2.2

Before analyzing the convergence properties of Algorithm 2.2, we establish a useful lemma.

Lemma 6. *Assume that the function of interest $f(x)$ and the interpolating function $p^k(x)$ at step $k > 0$ of Algorithm 2.2 are continuously twice differentiable functions with bounded Hessians. Denote $\lambda_{\max}(\cdot)$ as the maximum eigenvalue of its argument, and K is chosen as follow*

$$K > \lambda_{\max}(\nabla^2 f(x) - \nabla^2 p^k(x))/2 \tag{2.10}$$

for all x located in the feasible domain L. Then, there is a point $\tilde{x} \in L$ for which

$$s^k(\tilde{x}) \leq f(x^*), \tag{2.11}$$

where x^* is a global minimizer of $f(x)$.

Proof. Consider Δ_i^k as a simplex in Δ^k which includes x^*. Since the uncertainty function e_i^k is defined for all x inside the simplex Δ_i^k as

$$e_i^k(x) = (r_i^k)^2 - (x - z_i^k)^T (x - z_i^k),$$

the Hessian of the uncertainty function e_i^k is simply

$$\nabla^2 e_i^k(x) = -2\, I.$$

Now define a function $G(x)$ for all $x \in L$ such that

$$G(x) = p^k(x) - K\, e^k(x) - f(x), \tag{2.12}$$

and thus

$$\nabla^2 G(x) = (\nabla^2 p^k(x) - \nabla^2 f(x)) + 2\, K\, I. \tag{2.13}$$

By choosing K according to (2.10), the function $G(x)$ is strictly convex inside the closed simplex that includes x^*; thus, the maximum value of $G(x)$ is located at one of its vertices (see, e.g. Theorem 1 of [30]). Moreover, by construction, the value of $G(x)$ at the vertices of this simplex is zero; thus, $G(x^*) \leq 0$, and therefore $s^k(x^*) \leq f(x^*)$. □

Lemma 6 allows us to establish the convergence of Algorithm 2.2.

Theorem 2. *At step $k \geq 0$ of Algorithm 2.2, assume that S^k is the set of available data-points, that the function of interest $f(x)$ and the interpolating function $p^k(x)$ are continuously twice differentiable functions, and that L_p is a Lipschitz constant of $p^k(x)$. Assume also that K satisfies* (2.10)*. Define x^* and x_k as the global minimizers of $f(x)$ and the search function $s^k(x)$ at step k, respectively, and r^k_{max} as the maximum circumradius of the triangulation Δ^k; then*

$$0 \leq \min_{z \in S^k} f(z) - f(x^*) \leq \epsilon_k \quad \textit{where} \tag{2.14a}$$

$$\epsilon_k = (L_f + L_p + 2Kr^k_{max}) \cdot \min_{i<k} \|x_i - x_k\|. \tag{2.14b}$$

Proof. Select that i, with $i < k$, such that $\delta = \|x_i - x_k\|$ is minimized. By the Lipschitz norms

of $p^k(x)$ and (2.6), we have

$$\|p^k(x_i) - p^k(x_k)\| \leq L_p\,\delta,$$
$$\text{and} \quad \|e^k(x_i) - e^k(x_k)\| \leq 2\,r^k_{\max}\,\delta.$$

Noting that $s^k(x) = p^k(x) - Ke^k(x)$, we have

$$\|s^k(x_i) - s^k(x_k)\| \leq (L_p + 2\,Kr^k_{\max})\,\delta.$$

Since x_i is one of the evaluation points at the k-th step, at this point the value of the uncertainty function $e^k(x_i)$ is zero, and the values of the interpolant $p^k(x_i)$ and the function $f(x_i)$ are equal. That is,

$$s^k(x_i) = p^k(x_i) - K\,e^k(x_i) = f(x_i). \tag{2.15}$$

Since x_k is taken to be the global minimum of $s^k(x)$ and K satisfies (2.10), based on Lemma 6, $s^k(x_k) \leq f(x^*)$; Thus,

$$\begin{aligned} f(x^*) \geq s^k(x_k) &\geq s^k(x_i) - (L_p + 2\,K\,r^k_{\max})\,\delta \\ &= f(x_i) - (L_p + 2\,Kr^k_{\max})\,\delta \\ &\geq [f(x_k) - L_f\,\delta] - (L_p + 2\,K\,r^k_{\max})\,\delta, \\ \Rightarrow \quad f(x^*) &\geq f(x_k) - (L_f + L_p + 2\,K\,r^k_{\max})\,\delta. \end{aligned}$$

□

Remark 3. *Algorithm 2.2 begins from a set of initial datapoints, and computes one new datapoint at each iteration. In this setting,* (2.14) *provides a (sometimes conservative)* **termination certificate** *that guarantees that a desired degree of convergence ϵ_{des} has been*

attained. Algorithm 2.2 is simply marched in k until $\epsilon_k \leq \epsilon_{des}$*, where* ϵ_k *is defined in* (2.14b).

Note that, if r^k_{max} *is bounded,* $\epsilon_k \to 0$ *in the limit in which* $k \to \infty$*. Thus, there is a finite k for which* $\epsilon_k \leq \epsilon_{des}$*.*

Remark 4. *The existence of a bound* L_p *on the Lipschitz norm of* $p^k(x)$ *is required. The Lipschitz norm* L_p *of the interpolation* $p^k(x)$ *is, in general, cumbersome to compute. Subject to the above stated assumptions, a simpler way to prove convergence of (and, to certify a termination criterion for) Algorithm 2.2 that ensures that* (2.14a) *is attained for some* $\epsilon_k \leq \epsilon_{des}$ *is to calculate* ε_k *using linear interpolation, for which it can be shown that* $L_p \leq L_f$*, thereby replacing* (2.14b) *with*

$$\epsilon_k = (2\,L_f + 2Kr^k_{max}) \cdot \min_{y \in S^k} \|x_k - y\|.$$

Remark 5. *In addition to the required assumptions of a bound* L_p *for the Lipschitz norm of* $p^k(x)$*, and existence of K which* (2.10) *holds, a bound for the maximum circumradius* r^k_{max} *of the triangulation* Δ^k *is also required. In general, algorithm 2.2 cannot guaranty this property. A slight but technically important change to Algorithm 2.2 is presented in the next section to address this issue.*

2.4 Polyharmonic spline interpolation

The algorithms described above require the gradient and Hessian of the interpolant being used to facilitate Newton-based minimizations of the search function. Since our numerical tests all implement the polyharmonic spline interpolation formula, we now derive analytical expressions of the gradient and Hessian in this case.

The polyharmonic spline interpolation $p(x)$ of a function $f(x)$ in $\mathbb{R}^n$ is defined as

a weighted sum of a set of radial basis functions $\varphi(r)$ built around the location of each evaluation point, plus a linear function of x:

$$p(x) = \sum_{i=1}^{N} w_i \, \varphi(r) + v^T \begin{bmatrix} 1 \\ x \end{bmatrix}, \tag{2.16}$$

$$\text{where} \quad \varphi(r) = r^3 \quad \text{and} \quad r = \|x - x_i\|.$$

The weights w_i and v_i represent N and $n + 1$ unknowns, respectively, to be determined through appropriate conditions. First, we match the interpolant $p(x)$ to the known values of $f(x)$ at each evaluation point x_i, i.e. $p(x_i) = f(x_i)$; this gives N conditions. Then, we impose the orthogonality conditions $\sum_i w_i = 0$ and $\sum_i w_i x_{ij} = 0, \quad j = 1, 2, \ldots, n$. This gives $n + 1$ additional conditions. Thus,

$$\begin{bmatrix} F & V^T \\ V & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} f(x_i) \\ 0 \end{bmatrix} \quad \text{where}$$

$$F_{ij} = \varphi(\|x_i - x_j\|) \quad \text{and} \tag{2.17}$$

$$V = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_N \end{bmatrix}.$$

The gradient and Hessian of $p(x)$ may now be written as follows:

$$\nabla p(x) = \nabla \left(\sum_{i=1}^{N} w_i \|x - x_i\|^3 + v^T \begin{bmatrix} 1 \\ x \end{bmatrix} \right)$$

$$= 3 \sum_{i=1}^{N} w_i \|x - x_i\| (x - x_i) + \bar{v},$$

where $\bar{v} = [v_2, v_3, ..., v_{n+1}]^T$, and

$$\nabla^2 p(x) = \nabla^2 \left(\sum_{i=1}^{N} w_i ||x - x_i||^3 + v^T \begin{bmatrix} 1 \\ x \end{bmatrix} \right)$$
$$= 3 \sum_{i=0}^{N} w_i \left(\frac{(x - x_i)(x - x_i)^T}{||x - x_i||} + ||x - x_i|| I_{n \times n} \right).$$

Note that the calculation of the weights of a polyharmonic spline interpolant requires the solution of a $(N+n+1)\times(N+n+1)$ linear system. This system is not diagonally dominant, and does not show an easily-exploitable sparsity pattern facilitating fast factorization techniques. Nevertheless, since our algorithm adds only one point to the set of N evaluation points at each iteration, we can avoid the solution of the new linear system from scratch, and instead implement a rank-one update at each iteration as follows. First, for the set of initial points, we calculate the inverse $A = \left[\begin{smallmatrix} F & V^T \\ V & 0 \end{smallmatrix} \right]$. This step is somewhat time consuming, but reduces the computations required in subsequent steps. Using Matrix Inversion Lemma, we then update the inverse of A with the new information given at each step as follows:

$$A_{N+1}^{-1} = \begin{bmatrix} A_N & b^T \\ b & 0 \end{bmatrix}^{-1} = \begin{bmatrix} A_N^{-1} + A_N^{-1} b^T b A_N^{-1}/c & -A_N^{-1} b^T/c \\ -b A_N^{-1}/c & 1/c \end{bmatrix}, \tag{2.18}$$

where b is a vector of length $n+1$ defined as $b = \left[\begin{smallmatrix} 1 & x_{N+1} \end{smallmatrix} \right]^T$, and $c = -b A_N^{-1} b^T$ is a scalar. Multiplication of A_{N+1}^{-1} in (2.18) with the vector $\left[\begin{smallmatrix} f(x_i) & 0 & f(x_{N+1}) \end{smallmatrix} \right]^T$ gives the vector of weights in an unordered fashion, i.e. $\left[\begin{smallmatrix} w_i & v_i & w_{N+1}) \end{smallmatrix} \right]^T$. Therefore, before adding the new function evaluation in the following iteration and performing the next rank-one update, it is necessary to

permute the matrix A_{N+1}^{-1}, given by

$$A_{N+1}^{-1} = \begin{bmatrix} F & V^T & \varphi(\|x_{N+1} - x_i\|)^T \\ V & 0 & \begin{bmatrix} 1 & x_{N+1} \end{bmatrix} \\ \varphi(\|x_{N+1} - x_i\|) & \begin{bmatrix} 1 \\ x_{N+1} \end{bmatrix} & 0 \end{bmatrix}^{-1},$$

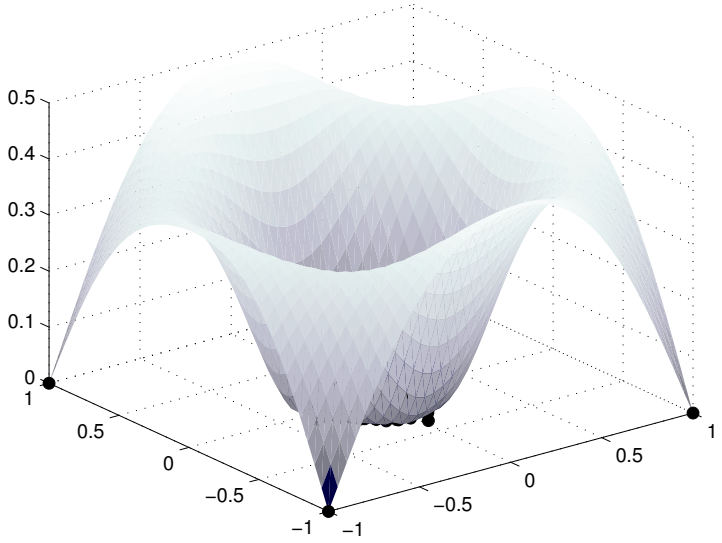
such that the desired 2×2 block form at the next iteration is recovered:

$$\begin{aligned} PA_{N+1}^{-1}P^T &= \begin{bmatrix} F_+ & V_+^T \\ V_+ & 0 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} F & \varphi(\|x_{N+1} - x_i\|)^T & V^T \\ \varphi(\|x_{N+1} - x_i\|) & 0 & \begin{bmatrix} 1 & x_{N+1} \end{bmatrix} \\ V & \begin{bmatrix} 1 \\ x_{N+1} \end{bmatrix} & 0 \end{bmatrix}^{-1}. \end{aligned}$$

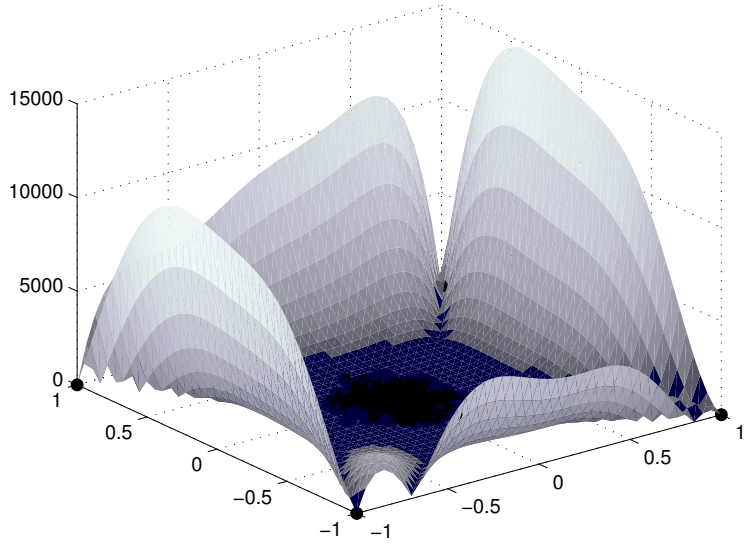
After this permutation, it is possible to apply the Matrix Inversion Lemma (2.18) at the following step.

Remark 6. *Another fast method to find the coefficients of radial basis functions is described in [31]. Since the present algorithms build the dataset incrementally, the method described above is less expensive in the present case.*

As mentioned earlier, variations of Kriging interpolation are often used in Response Surface Methods, such as the Surrogate Management Framework, for derivative-free optimization. DACE (see [32]) is one of the standard packages used for numerically computing the Kriging interpolant. Figures 2.2a and 2.2b compare of the polyharmonic spline interpolation method described above and the Kriging interpolation method computed using DACE, as applied to the test function $f(r) = r * \sin 1/r$, where $r^2 = x^2 + y^2$ with $N = 1004$ data points. The data points used in this example are the 4 corners of a square domain, and 1000 random-chosen points clustered within a small neighborhood of the center of the



a The error of the polyharmonic spline interpolation interpolant (5.7).



b The error of the Kriging interpolant with a Gaussian model for the corrrelation, computed using DACE.

Figure 2.2: Comparison of polyharmonic spline interpolation and Kriging interpolant. The difference between the actual function, $f(r) = r * \sin 1/r$, where $r^2 = x^2 + y^2$, and its interpolant for two different interpolation strategies when 1000 function evaluations are clustered near the center of a square domain.

square, which highlights the numerical challenge of performing interpolation when grid points begin to cluster in a particular region, which is common when a response surface

method for derivative-free optimization approaches convergence. Figures 2.2a and 2.2b plot the difference between the real value of f and the value of the corresponding interpolants.

An observation which motivated the present study is that, in such problems, the Kriging interpolant is often spurious in comparison with other interpolation methods, such as polyharmonic splines. Note that various methods have been proposed to regularize such spurious interpolations in the presence of clustered datapoints, such as combining interpolants which summarize global trends with interpolants which account for local fluctuations. Our desire in the present effort was to develop a robust response surface method that can implement any good interpolation strategy, the selection of which is expected to be somewhat problem dependent.

2.5 Bounding the circumradii

In the previous section, we established that Algorithm 2.2 converges to the global minimum of the cost function under two assumptions: (a) the underlying function of interest $f(x)$ is Lipschitz and twice differentiable with bounded Hessian, and (b) the maximum circumradii of the simplicies in the triangulations are bounded as the algorithm proceeds. Without assumption (a), or something like it, not much can be done to assure global convergence, beyond requiring that the grid becomes everywhere dense as the number of function evaluations approaches infinity. Assumption (b), however, is problematical, as Algorithm 2.2 doesn't itself ensure this condition. In this section, we make a small but important technical adjustment to Algorithm 2.2 to ensure that condition (b) is satisfied.

Distributing points in $\mathbb{R}^n$ such that the resulting Delaunay triangulation of these points has a bounded maximum circumradius is a common problem in 2D and 3D mesh

generation (see [33], [34], [35]). Applying known strategies for this problem to the present application is challenging, however, due to the incremental nature of the point generation, the interest in $n > 3$, the large number and nonuniform distribution of points generated, and the possibly sharp corners of the feasible domain itself. In this section, we thus develop a new method for solving this problem that meets these several challenges.

The feasible domain L considered is defined in (2.1a); we assume for the remainder of this section that the problem is n-dimensional, is bounded by m constraints which result in M vertices of the feasible domain, that all redundant constraints have been eliminated [see step (B) of Algorithm 2.1], and that all equality constraints implied by the condition $Ax \leq b$ have been projected out of the problem [see step (C) of Algorithm 2.1].

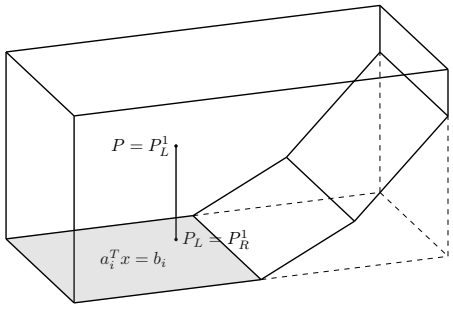
Definition 5. *Consider P as a point in L, $a_i^T x \leq b_i$ as a constraint which is not* **active** *(that is, equality) at P, and H_i as the $(n-1)$-dimensional hyperplane given by equality in this constraint. The* ***feasible constraint projection*** *of P onto H_i is the point P_L defined as the outcome of the following procedure:*

0. *Set $k = 1$ and $P_L^1 = P$.*

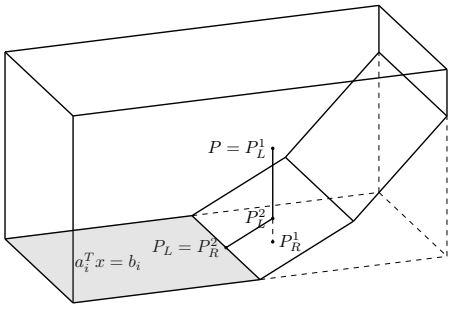
1. *Define m_a^k as the number of active constraints at P_L^k, and H_L^k as the hyperplane (with dimension less than or equal to n) implied by this set of constraints. If $m_a^k = n - 1$, set $P_L = P_L^k$ and exit.*

2. *If $m_a^k > 0$ and there is no vertex of L that is contained within both H_L^k and H_i, set $P_L = P_L^k$ and exit.*

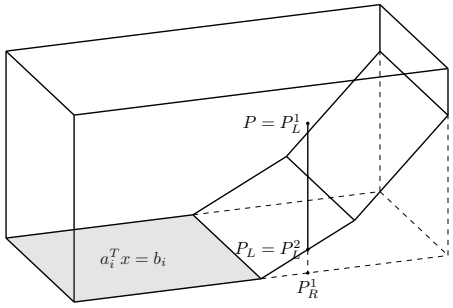
3. *Obtain P_R^k as the point which is contained within both H_L^k and H_i, and has minimum distance from P_L. If $P_R^k \in L$, set $P_L = P_R^k$ and exit.*



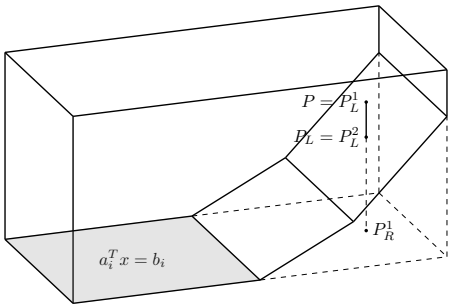
a Complete, termination at step 3 of iteration $k = 1$.



b Complete, termination at step 3 of iteration $k = 2$.



c Incomplete, termination at step 1 of iteration $k = 2$.



d Incomplete, termination at step 2 of iteration $k = 2$.

Figure 2.3: Feasible constraint projections of various points P onto the constraint $a_i^T x \leq b$.

.

4. *Otherwise, set* P_L^{k+1} *as the intersection of the line segment from* P_L^k *to* P_R^k *with the boundary of L, increment k, and repeat from step* 1.

If the above-described procedure exits at step 3, and thus $P_L \in H_i$*, then* P_L *is said to be a* ***complete*** *feasible constraint projection; otherwise (that is, if the procedure exits at step 1 or 2),* P_L *is said to be an* ***incomplete*** *feasible constraint projection. The procedure described above will terminate after some* $\overline{k}$ *steps, where* $\overline{k} < n$*; the* P_L^k *for* $k < \overline{k}$ *are referred to as* ***intermediate*** *feasible constraint projections.*

Some examples of complete and incomplete feasible constraint projections are given in Figure 2.3.

Lemma 7. *Consider the P_L^k as the intermediate feasible constraint projections at each step $k \leq \bar{k}$ of a feasible constraint projection (see Definition 5), which exits at iteration $\bar{k}$, of some point $P \in L$ onto the hyperplane H_i defined by the constraint $a_i^T x \leq b_i$. For any point V which lies within the intersection of H_L^k and H_i,*

$$\frac{\|P - V\|}{\|P - P_T\|} \leq \frac{\|P_L^k - V\|}{\|P_L^k - P_T^k\|} \tag{2.19}$$

for $k \leq \bar{k}$, where P_T and P_T^k are the projections of P and P_L^k on H_i.

Proof. In the procedure described in Definition 5, at each step k, if P_R^k is in L; then $P_L^{k+1} = P_R^k$. By construction, V is in the intersection of H_L^k and H_i, and P_R^k is the point in the intersection of H_L^k and H_i that has minimum distance from P_L^k; it thus follows that $\overrightarrow{P_L^k P_R^k}$ is perpendicular to $\overrightarrow{P_R^k V}$. Since P_L^{k+1} is a point on the line between P_L^k and P_R^k, it follows that $\overrightarrow{P_L^{k+1} P_R^k}$ is also perpendicular to $\overrightarrow{P_R^k V}$. Thus,

$$\begin{aligned}
\|P_L^k - V\|^2 &= \|P_L^k - P_R^k\|^2 + \|V - P_R^k\|^2, \\
\|P_L^{k+1} - V\|^2 &= \|P_L^{k+1} - P_R^k\|^2 + \|V - P_R^k\|^2, \\
\|P_L^{k+1} - P_R^k\| &\leq \|P_L^k - P_R^k\|, \\
\frac{\|P_L^k - V\|}{\|P_L^k - P_R^k\|} &\leq \frac{\|P_L^{k+1} - V\|}{\|P_L^{k+1} - P_R^k\|}.
\end{aligned} \tag{2.20}$$

Since P_L^k, P_L^{k+1} and P_R^k are collinear and P_R^k is on the hyperplane $a_i^T x = b_i$, it follows that

$$\frac{\|P_L^{k+1} - P_R^k\|}{\|P_L^k - P_R^k\|} = \frac{\|P_L^{k+1} - P_T^{k+1}\|}{\|P_L^k - P_T^k\|}. \tag{2.21}$$

Combining (2.20) and (2.21) over several steps k and noting that $P_L^1 = P$, (2.19) follows.

□

Definition 6. *Consider P as a point in a set of points S in the convex polyhedra L, $a_i^T x = b_i$ as a constraint which is not active at P, and H_i as the hyperplane defined by this constraint. The points P_L^k are taken as the intermediate feasible constraint projections of P onto H_i (see Definition 5, which we take as exiting at iteration $\bar{k}$). With respect to some parameter $r > 1$, the point P is said to be* ***poorly situated*** *with respect to the constraint $a_i^T x \leq b_i$ if the feasible constraint projection is complete and, for any point $V \in S$ which is in both H_L^k and H_i,*

$$\frac{\|P_L^k - V\|}{\|P_L^k - P_R^k\|} > r, \quad \forall\ 1 \leq k \leq \bar{k}; \tag{2.22}$$

otherwise, the point P is said to be ***well situated*** *with respect to the constraint $a_i^T x \leq b_i$. The set of data points S is a* ***well-situated set*** *if, for all pairs of points $P \in S$ and constraints $a_i^T x \leq b_i$ defining L, P is well-situated with respect to the constraint $a_i^T x \leq b_i$.*

Remark 7. *It is easy to verify that the set of vertices of the feasible domain L is itself a well-situated set for any $r > 1$.*

The important property of a well-situated set S is that the maximum circumradius of the Delaunay triangulation of S is bounded by r [the factor used in (2.22), which will be considered further at the end of §2.5] and some parameters related to L. These geometric parameters are identified in Definitions 7 and 8, and existence of a bound for the maximum circumradius is proved in Theorem 3, based on Lemmas 9 and 10.

Definition 7. *Consider $A_a(V)$ as the set of active constraints at a vertex V of a feasible domain L with redundant constraints eliminated [see step (B) of Algorithm 2.1], and all equality constraints removed [see step (C) of Algorithm 2.1]. Since there are no redundant constraints, $A_a(V)$ includes exactly n constraints which are linearly independent. Consider $a_i^T x \leq b_i$ as a constraint in $A_a(V)$, and $A_a^i(V)$ as the set of all active constraints at V except*

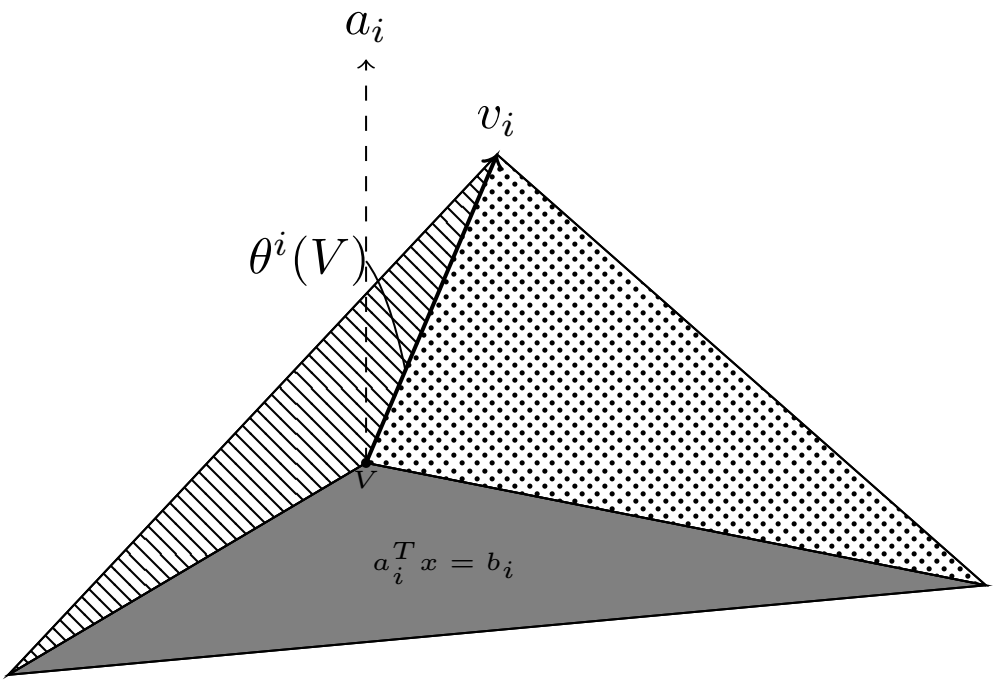


Figure 2.4: The skewness of a vertex V with respect to constraint $a_i^T x \leq b_i$ in $n = 3$ dimensions is defined as $S^i(V) = 1/\cos(\theta^i(V))$, where $\theta^i(V)$ is the angle indicated.

$\{a_i^T x \leq b_i\}$. *It can be observed that the space defined by* $A_a^i(V)$ *is a ray from* V. *In other words, each point in this one-dimensional space can be written as* $V + \alpha\, v_i$, *where* α *is a positive real number. Defining* $\theta^i(V)$ *as the angle between* a_i *and* v_i, *the skewness of the vertex* V *with respect to the constraint* $a_i^T x \leq b_i$ *is defined as* $S^i(V) = 1/\cos(\theta^i(V))$. *The skewness of the feasible domain* L, *denoted* $S(L)$, *is the maximum of* $S^i(V)$ *over all vertices* V *and active constraints* $a_i^T x \leq b_i$ *at* V.

Remark 8. *If the feasible domain* L *is defined by simple box constraints* ($a \leq x \leq b$), *then the skewness of all vertices with respect to all active constraints (and, thus, the skewness of the domain* L *itself) is equal to* 1. *In* $n = 2$ *dimensions, the skewness of each vertex is simply* $S^i(V) = 1/|\sin(\theta)|$, *where* θ *is the angle of vertex* V. *In* $n = 3$ *dimensions, the value of* $\theta^i(V)$ *is illustrated in Figure 2.4. Note that, in general,* $S^i(V) \geq 1$.

Lemma 8. *Consider V as a vertex of L, and $A_a(V)$ as the set of constraints active at V. Consider some point $x \in L$ at which the set of active constraints, denoted $A_{a_1}(V)$, is a proper subset of $A_a(V)$. Denote $a_i^T x \leq b_i$ as a constraint in $A_a(V)$ which is not in $A_{a_1}(V)$. Define x_1 as the projection of x onto the hyperplane defined by $a_i^T x = b_i$, and define x_2 as the projection of x onto the hyperplane defined by the union of $a_i^T x = b_i$ and $A_{a_1}(V)$. Then,*

$$1 \leq \frac{\|x - x_2\|}{\|x - x_1\|} \leq S^i(V) \leq S(L). \tag{2.23}$$

Proof. Consider x_3 as the point in the hyperplane defined by $A_a^i(V)$ with minimum distance from x, where $A_a^i(V)$ is identified in Definition 7. By construction, $\|x - x_3\| \geq \|x - x_2\|$; note that x_3 is also in the hyperplane defined by $A_{a_1}(V)$. Moreover, according to Definition 7, $\|x - x_3\|/\|x - x_1\| = S^i(V)$. □

Definition 8. *For each pair of vertices V of L and constraints $a_i^T x \leq b_i$ not active at V, V_L is defined as the projection of V onto the hyperplane defined by $a_i^T x = b$. Define $L_1 = \max_{x,y \in L} \|x - y\|$ as is the* **diameter** *of L. The* ***aspect ratio*** *$\kappa(L)$ is taken as the maximum value of $L_1/\|V - V_L\|$ for all pairs of vertices V and constraints $a_i^T x \leq b_i$ not active at V.*

Lemma 9. *Consider S as a set of feasible points in L (including the vertices of L) which is well-situated (see Definition 6) with factor r. Then, for each pair of points $P \in S$ and constraints $a_i^T x \leq b_i$ not active at P, there is a point $V \in S$ such that $a_i^T V = b_i$ and*

$$1 \leq \frac{\|P - V\|}{\|P - P_T\|} \leq \max\{rS(L), \kappa(L)\}, \tag{2.24}$$

where P_T is taken as the projection of P onto the hyperplane H_i, and $S(L)$ and $\kappa(L)$ are the skewness and aspect ratio of the feasible domain L.

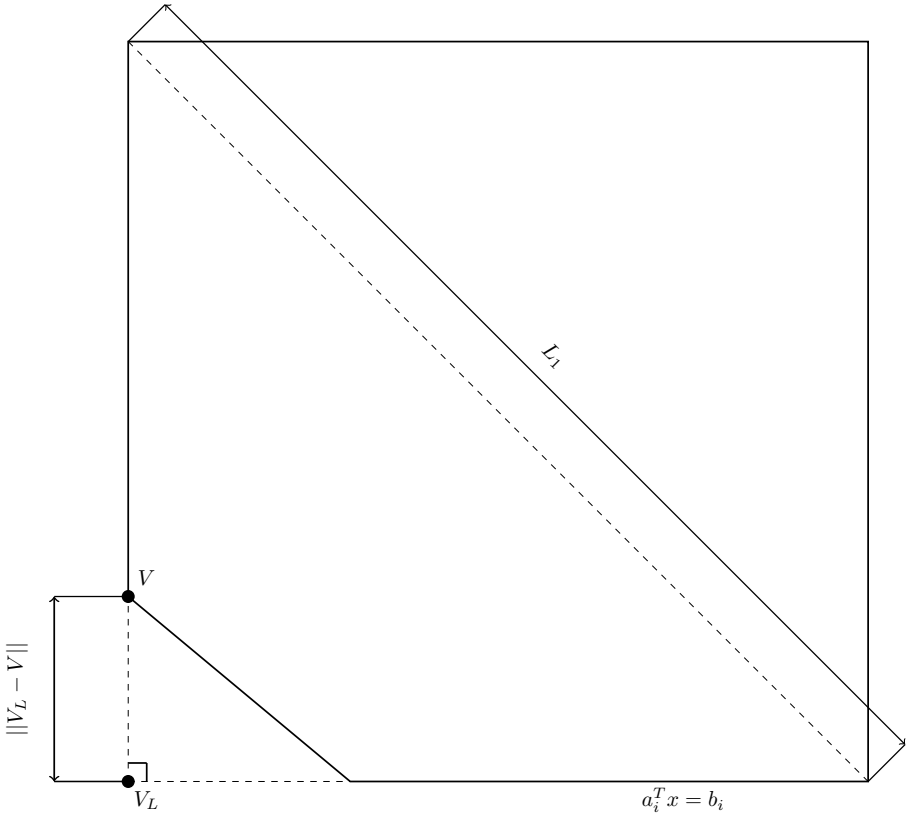


Figure 2.5: Aspect ratio of the convex polyhedra. This figure represent the aspect ratio of the convex polyhedra L. Note that the vertex V and constraint $a_i^T x = b_i$ has minimum distance from all pair of vertex and constraints, and L_1 is the diameter of the polyhedra L. The aspect ratio is equal to $\kappa(L) = \frac{L_1}{V - V_T}$ where $\|V - V_T\|$ and L_1 are shown.

Proof. Consider P_L as the feasible constraint projections of P onto H_i (see Definition 5, which we take as exiting at iteration $\bar{k}$). According to this procedure, there are three possible termination conditions, each of which is considered below.

First, consider the case in which the feasible constraint projection is terminated at step 2. In this case, $P_L = P_L^{\bar{k}}$, and there is no vertex of L that is contained within both $H_L^{\bar{k}}$ and H_i. Therefore, the point in intersection of the feasible domain L and the hyperplane $H_L^{\bar{k}}$ which has minimum distance from the hyperplane H_i is a vertex of L, denoted V. Thus,

$$\|V - V_T\| \leq \|P_L - P_{L^T}\|, \tag{2.25}$$

where P_{L^T}, P_T^k and V_T are the projections of P_L, P_L^k and V onto H_i; thus, $P_{L^T} = P_T^{\bar{k}}$. Further, at each step of the procedure of feasible constraint projection (Definition 5), the distance of the point considered to H_i is reduced. Thus,

$$\|P_L - P_{L^T}\| = \|P_L^{\bar{k}} - P_T^{\bar{k}}\| \leq \|P - P_T\|. \tag{2.26}$$

Using (2.25) and (2.26),

$$\|V - V_T\| \leq \|P - P_T\| \tag{2.27}$$

On the other hand, $\|P - V\| \leq L_1$, where L_1 is the diameter of the feasible domain L. Thus,

$$\frac{\|P - V\|}{\|P - P_T\|} \leq \frac{L_1}{\|V - V_T\|} \leq \kappa(L). \tag{2.28}$$

Next, consider the case in which the feasible constraint projection is terminated at step 1. In this case, the number of active constraints at P_L is $m_a = n - 1$, and $P_L = P_L^{\bar{k}}$. If there is no vertex of L that is contained within both $H_L^{\bar{k}}$ and H_i, the situation is similar to

the previous case, and (2.28) again follows. On the other hand, if there is a vertex V of L which is in both $H_L^{\bar{k}}$ and H_i, then it follows from Lemma 7 that

$$\frac{||P - V||}{||P - P_T||} \leq \frac{||P_L - V||}{||P_L - P_{L^T}||}. \tag{2.29}$$

Note that $A_a(P_L)$ is a proper subset of $A_a(V)$ which does not include the constraint $a_i^T x \leq b_i$, and V is the only point which is in both $H_L^{\bar{k}}$ and H_i, as the intersection of n linear independent hyperplanes is a unique point. Therefore, via Lemma 8 (taking $x = P_L$ and thus, by construction, $x_1 = P_{L^T}$ and $x_2 = V$),

$$\frac{||P_L - V||}{||P_L - P_{L^T}||} \leq S^i(V) \leq S(L) \tag{2.30}$$

Using (2.29) and (2.30)

$$\frac{||P - V||}{||P - P_T||} \leq S(L) \leq rS(L). \tag{2.31}$$

Finally, consider the case in which the feasible constraint projection is terminated at step 3, and thus the process is complete, and $P_L = P_R^{\bar{k}}$. Since P is well situated with respect to the constraint $a_i^T x \leq b_i$ (see Definition 6), there is a $k \in \{1, 2, \ldots, \bar{k}\}$ and a point $V \in S$ which is in both H_L^k and H_i such that

$$\frac{||P_L^k - V||}{||P_L^k - P_R^k||} \leq r. \tag{2.32}$$

Since V is in both H_L^k and H_i, there is a vertex of L, denoted W, which is in both H_L^k and H_i. Moreover, P_L^k is not in H_i, and $A_a(P_L^k)$ is a proper subset of $A_a(W)$. Denote again P_T^k as the projection of P_L^k on the hyperplane H_i. Via Lemma 8 (taking $x = P_L^k$ and thus, by

construction, $x_1 = P_T^k$ and $x_2 = P_R^k$), it follows that

$$\frac{||P_L^k - P_R^k||}{||P_L^k - P_T^k||} \leq S^i(W) \leq S(L). \tag{2.33}$$

Combining (2.32) and (2.33),

$$\frac{||P_L^k - V||}{||P_L^k - P_T^k||} \leq rS(L)$$

Thus, via Lemma 7, it follows that

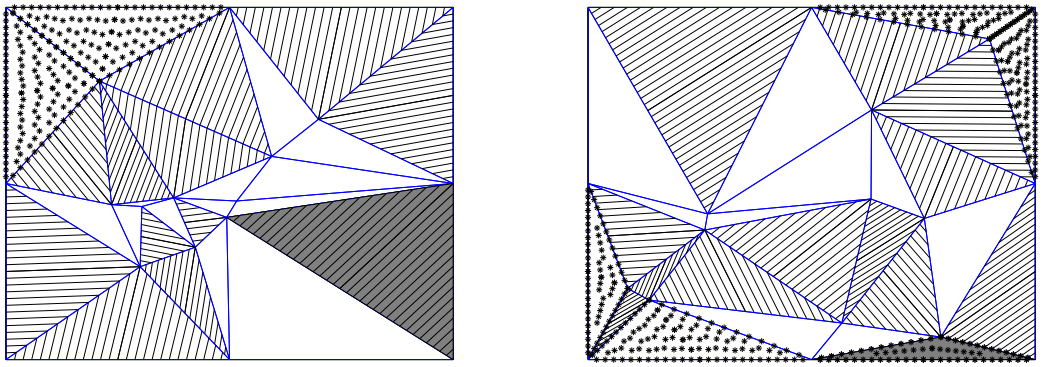
$$\frac{||P - V||}{||P - P_T||} \leq \frac{||P_L^k - V||}{||P_L^k - P_T^k||} \leq rS(L). \tag{2.34}$$

□

Lemma 9 allows us to show that the maximum circumradius of a Delaunay triangulation of a well-situated set of points is bounded. In order to do this, some additional lemmas and definitions are helpful.

Definition 9. *A simplex Δ_x in a Delaunay triangulation Δ^k of a set of points S (including the vertices of L) which has the maximum circumradius among all simplices is called a* ***maximal*** *simplex. Note that a given triangulation might have more than one maximal simplices.*

A simplex Δ_x is called a ***candidate*** *simplex if either (a) the circumcenter of Δ_x is inside or on the boundary of Δ_x, or (b) an $n-1$ dimensional face p of this simplex forms part of the boundary of L corresponding to equality in the constraint $a_i^T x \leq b_i$, and the circumcenter of Δ_x violates this constraint. Case (a) is denoted an* ***interior*** *candidate simplex, and case (b) is denoted a* ***boundary*** *candidate simplex.*



a An interior candidate simplex is maximal. b A boundary candidate simplex is maximal.

Figure 2.6: Candidate simplices in triangulation. The position of candidate simplices for a representative triangulation. Hatched triangles are interior candidate simplices, and triangles filled with stars are boundary candidate simplices. The dark shaded area is the maximal simplex.

Lemma 10. *There is a maximal simplex in a Delaunay triangulation Δ^k of a set of points S (including the vertices of L) which is a candidate simplex.*

Proof. Consider Δ_x as a maximal simplex of a Delaunay triangulation Δ^k. If Δ_x is a candidate simplex, then the lemma is true. Otherwise, define p as an $n-1$ dimensional face of this simplex, lying within an $n-1$ dimensional hyperplane H, in which V, the vertex of Δ_x that is not in H, and Z, the circumcenter of Δ_x, are on opposite sides of H, and none of the $n-1$ dimensional boundaries of L lie within H.

Then there is a simplex Δ_x^1 which is a neighbor of Δ_x that shares the face p. Define V' as the vertex of Δ_x^1 which is not in H, and Z' as the circumcenter of Δ_x^1. Since the triangulation is Delaunay

$$||Z - V|| \leq ||Z - V'|| \text{ and } ||Z' - V'|| \leq ||Z' - V||. \tag{2.35}$$

Define Z_T as the projection of both Z and Z' on H (by construction, they have the same projection), and V_T and V_T' as the projections of V and V' on H, respectively. By the

assumption, Z and V' are on one side of H, and V is on the other side. Thus, (2.35) implies

$$\|V_T - Z_T\|^2 + [\|Z_T - Z\| + \|V - V_T\|]^2 \leq \|V'_T - Z_T\|^2 + [\|V' - V'_T\| - \|Z_T - Z\|]^2. \quad (2.36)$$

Moreover, regardless of the position of Z', we may write

$$\|Z' - V'\|^2 \geq \|V'_T - Z_T\|^2 + [\|V' - V'_T\| - \|Z_T - Z'\|]^2,$$
$$\|Z' - V\|^2 \leq \|V_T - Z_T\|^2 + [\|V' - V'_T\| + \|Z_T - Z'\|]^2;$$

thus, due to (2.35),

$$\|V'_T - Z_T\|^2 + [\|V' - V'_T\| - \|Z_T - Z'\|]^2 \leq \|V_T - Z_T\|^2 + [\|V - V_T\| + \|Z_T - Z'\|]^2. \quad (2.37)$$

Adding (2.36) and (2.37) gives

$$\|Z_T - Z\|\|V - V_T\| - \|V' - V'_T\|\|Z_T - Z'\| \leq \|V - V_T\|\|Z_T - Z'\| - \|V' - V'_T\|\|Z_T - Z\|,$$

and thus

$$\|Z_T - Z\|[\|V - V_T\| + \|V' - V'_T\|] \leq \|Z_T - Z'\|[\|V - V_T\| + \|V' - V'_T\|],$$

$$\|Z - Z_T\| \leq \|Z' - Z_T\|. \quad (2.38)$$

Define W as a common vertex of Δ_x and Δ'_x, noting that W must be in H. Since $Z - Z_T$ and $Z' - Z_T$ are perpendicular to $Z_T - W$, (2.38) gives

$$\|Z - W\| \leq \|Z' - W\|. \quad (2.39)$$

It may be shown analogously that, if (2.35) is a strict inequality, then (2.39) is a strict inequality as well. Further, since Δ_x is a maximal simplex, $\|Z - W\| \geq \|Z' - W\|$; thus, the inequality (2.39) must be an equality, and Δ_x and Δ'_x are both maximal. We may also conclude that[5] (2.35) must also be an equality, which implies that Z and Z' are, in fact, the same point.

Now define F as the polygon which is equal to the union of those simplices of Δ^k whose circumcenter is Z; note that all of the simplices that make up F are maximal. If Z is inside F, the simplex which includes Z is an interior candidate simplex. If Z is not inside F then, by the above reasoning, any boundary of F which Z is on the opposite side of must also be a boundary of L, and the simplex in F which shares this boundary is a boundary candidate simplex. □

Theorem 3. *Consider Δ^k as an n dimensional Delaunay triangulation of a set of well-situated points $S \in L$ (including the vertices of L), with factor of r. Then*

$$R_{max} \leq L_2 r_1^{n-1} \quad \textit{where} \quad r_1 = \max\{rS(L), \kappa(L)\},$$

where R_{max} is the maximum circumradius, L_2 is the maximum edge length in all simplices, and $S(L)$ and $\kappa(L)$ are the skewness and aspect ratio of L.

Proof. This theorem is shown by induction on n, the dimension of the problem. For $n = 1$, the circumcenter of any simplex is located in L, and the lemma is trivially satisfied. Assuming the theorem is true for the $n-1$ dimensional case, we now show that the theorem is also true for the n dimensional case.

Consider Δ_x as a maximal simplex of the n dimensional Delaunay triangulation Δ^k, with circumcenter Z, which is also a candidate simplex (see Lemma 10). If Δ_x is an interior

[5]The logic for this conclusion is as follows: if (i) $a \leq b$ and (ii) $a < b \ \rightarrow c < d$, then, if $c = d$, then $a = b$.

candidate simple, it includes its circumcenter, and the lemma is trivially satisfied, since Z is located in L. Otherwise, Δ_x is a boundary candidate simplex, and there is a constraint $a_i^T x \leq b_i$ bounding L which is active at n vertices of Δ_x, and Z violates this constraint. Denote H_i as the $n-1$ dimensional hyperplane which contains this constraint.

Consider P as the vertex of Δ_x which is not in H_i. Define Z_T and P_T as the projections of Z and P onto H_i, and W as a vertex of Δ_x which is in H_i; then

$$||Z - P|| = ||Z - W||,$$

$$\begin{gathered}[||Z - Z_T|| + ||P - P_T||]^2 + ||Z_T - P_T||^2 = ||Z - Z_T||^2 + ||Z_T - W||^2 \\ 2||Z_T - Z||||P - P_T|| + ||P - P_T||^2 \\ = ||Z_T - W||^2 - ||Z_T - P_T||^2\end{gathered} \tag{2.40}$$

By construction Z_T is the circumcenter of an $n-1$ dimensional simplex Δ_x^1 which includes all vertices of Δ_x except P. Note that restriction of Δ^k onto the hyperplane H_i is itself an $n-1$ dimensional Delaunay triangulation. In other words, for any point $V \in S$ that is in H_i,

$$||Z_T - V|| \geq ||Z_T - W||$$

$$||P_T - V|| \geq ||Z_T - V|| - ||Z_T - P_T||$$

$$||P_T - V|| \geq ||Z_T - W|| - ||Z_T - P_T||$$

$$||Z_T - W||^2 - ||Z_T - P_T||^2 = (||Z_T - W|| + ||Z_T - P_T||)(||Z_T - W|| - ||Z_T - P_T||)$$

According to equation (2.40), $||Z_T - W|| \geq ||Z_T - P_T||$; it thus follows from the above equations that

$$||Z_T - W||^2 - ||Z_T - P_T||^2 \leq 2||Z_T - W||||P_T - V||. \tag{2.41}$$

Combining (2.40) and (2.41), we may write

$$||Z_T - Z||||P - P_T|| \leq ||Z_T - W||||P_T - V|| \tag{2.42}$$

Furthermore, by construction, $Z_T - Z$ and $P - P_T$ are perpendicular H_i to ; therefore,

$$||Z - W||^2 \leq ||Z_T - W||^2[1 + \frac{||V - P_T||^2}{||P - P_T||^2}]$$
$$||P - V||^2 = ||V - P_T||^2 + ||P - P_T||^2$$
$$||Z - W|| \leq ||Z_T - W||\frac{||P - V||}{||P - P_T||} \tag{2.43}$$

On the other hand, since S is well-situated with factor of r, according to Lemma 9, there is a point V in S which is in H_i, and

$$\frac{||P - V||}{||P - P_T||} \leq r_1. \tag{2.44}$$

Note that, if S is well situated with factor of r, the subset of points of S which lie within H_i, denoted S_i, are also well situated with factor of r. Since $||Z_T - W||$ is the circumradius of the $n-1$ dimensional simplex Δ_x^1 of the Delaunay triangulation of S_i, by the inductive hypothesis,

$$||Z_T - W|| \leq L_2 r_1^{n-2}$$

Applying (2.43) and (2.44), it thus follows from the above condition that

$$||Z - W|| \leq L_2 r_1^{n-1}$$

Since $||Z - W||$ is equal to the maximum circumradius of Δ^k, the theorem is proved. □

We now use Theorem 3 to perform a slight modification of Algorithm 2.2 in a way that ensures the set of datapoints remains well situated, with factor r, as the iteration proceeds. In this way, a bound for the maximum circumradius of the Delaunay triangulations generated by the algorithm is assured.

Algorithm 2.2 is initialized with the vertices of L. By Remark 7, this set of points is well situated. Algorithm 2.2 then (a) adds to this initial set of points a number of user specified points of interest, and then (b) adds (at step 5) a new datapoint [selected carefully, as described in the algorithm] to the existing set of datapoints at each iteration, until convergence. We now modify Algorithm 2.2 such that each time a new datapoint P is added, in both steps a and b above, an adjustment Q to the location of point P is made, if necessary, in order to ensure that set of datapoints remains well situated. This adjustment is performed as follows.

Algorithm 2.3 Assume S is a well-situated set of points, and P is a candidate point to be added to this set (after adjustment, if necessary).

1: Set Q=P.
2: Find a constraint $a_i^T x \leq b_i$ for which P is not in a well situated position. If none can be found, stop the algorithm, and return Q.
3: Replace Q by the feasible constraint projection of Q on $a_i^T x \leq b_i$ (see Definition 5).
4: Repeat from step 1 until the algorithm stops.

Note that $A_a(Q)$ includes the active constraints of P. At each step of the above algorithm, an additional constraint is added. Thus, the above algorithm stops after at most $n-1$ iterations. In Lemma 11, we show that Algorithm 2.2 still converges, even if we add Q instead of P at each iteration.

Lemma 11. *Consider S as a well-situated set of points in L, and Q as the outcome of*

Algorithm 2.3 from input P. Then,

$$\min_{x\in S} \|P - x\| \leq \rho \cdot \min_{y\in S} \|Q - y\|, \tag{2.45}$$

$$\rho = [2r_1^2\,(1 - \frac{1}{r^2})]^{-\frac{n-1}{2}}, \tag{2.46}$$

$$r_1 = \max\{rS(L), \kappa(L)\}. \tag{2.47}$$

Proof. Consider $y \in S$ as a point which minimizes $\|Q - y\|$. If a constraint $a_i^T x \leq b_i$ exists in $A_a(Q)$ which is not active at y, since S is well-situated, according to Lemma 9, there is a point $y^1 \in S$ such that $a_i^T x \leq b_i$ is active at it, and

$$\frac{\|y - y^1\|}{\|y - y_T\|} \leq r_1, \tag{2.48}$$

where y_T is the projection of y on the hyperplane $a_i^T x = b_i$. By construction,

$$\begin{aligned} \|y - Q\|^2 &= \|y - y_T\|^2 + \|y_T - Q\|^2, \\ \|y - Q\| &\geq \frac{\|y - y_T\| + \|y_T - Q\|}{\sqrt{2}}, \\ \|y^1 - Q\| &\leq \|y^1 - y_T\| + \|y_T - Q\|, \\ \frac{\|y - Q\|}{\|y^1 - Q\|} &\geq \frac{1}{\sqrt{2}} \frac{\|y - y_T\| + \|y_T - Q\|}{\|y^1 - y_T\| + \|y_T - Q\|}. \end{aligned} \tag{2.49}$$

Using (2.48) and (2.49), we have:

$$\frac{\|y - Q\|}{\|y^1 - Q\|} \geq \frac{1}{\sqrt{2}\, r_1}. \tag{2.50}$$

Using (2.50), recursively over the $k \leq n - 1$ binding constraints at point Q, we will

derive that there is a point $y^k \in S^k$ in which $A_a(Q) \subseteq A_a(y^k)$, and

$$\frac{\|y - Q\|}{\|y^k - Q\|} \geq (\frac{1}{\sqrt{2}\, r_1})^{n-1}. \tag{2.51}$$

Take $V = y^k$; then we will show that

$$\frac{\|P - V\|}{\|Q - V\|} \leq (1 - \frac{1}{r^2})^{-\frac{n-1}{2}}. \tag{2.52}$$

According to Algorithm 2.3, Q is derived by a series of successive complete feasible constraint projections of a point P onto various constraints of L which are active at Q. Assume that m feasible constraint projections are performed in during the process of Algorithm 2.3, and $\{P^1, P^2, \ldots, P^{m+1}\}$ is the series of points which are generated by Algorithm 2.3. In this way, $P^1 = P$, $P^{m+1} = Q$, and P^i for $1 < i \leq m + 1$ is the feasible-constraint-projection of P^{i-1} onto a constraint of L demoted by $a_i^T x \leq b_i$.

Define $P_L^{i,j}$ as the intermediate feasible constraint projection of P^i onto constraint $a_i^T x \leq b_i$, at step j, and $H_L^{i,j}$ as the hyperplane (with dimension less than or equal to n) implied by $A_a(P_L^{i,j})$. Then denote $P_R^{i,j}$ as a point in the intersection of H_i and $H_L^{i,j}$, that has minimum distance from $P_L^{i,j}$. (This is similar to P_R^k in Definition 5).

Since the point P^i is poorly situated position with respect to the constraint $a_i^T x \leq b_i$, and $A_a(P_L^{i,j}) \subseteq A_a(Q)$, and therefore V is in both H_i and $H_L^{i,j}$,

$$\frac{\|P_L^{i,j} - V\|}{\|P_L^{i,j} - P_R^{i,j}\|} > r$$

$$\|P_L^{i,j} - V\|^2 = \|P_L^{i,j} - P_R^{i,j}\|^2 + \|P_R^{i,j} - V\|^2$$

$$\frac{\|P_L^{i,j+1} - V\|}{\|P_L^{i,j} - V\|} \geq \frac{\|P_R^{i,j} - V\|}{\|P_L^{i,j} - V\|} \geq \sqrt{1 - \frac{1}{r^2}}$$

Moreover, at each iteration of the feasible constraint projection, a linearly independent constraint is added to the set of active constraints, therefore, step 3 of the procedure of feasible boundary projection could happened at most $n-1$ times. Thus, (2.52) is satisfied which shows (2.45) when $A_a(Q) \subseteq A_a(V)$. Furthermore, by using (2.51) and (2.52), (2.45) is satisfied when $A_a(Q) \nsubseteq A_a(V)$.

□

Theorem 4. *Algorithm 2.2, with the adjustment described in Algorithm 2.3, will converge to the global minimum of the feasible domain L if the parameter K satisfies* (2.10)*, and $p^k(x)$ is Lipschitz with a single Lipschitz constant L_p for all steps k.*

Proof. Consider S^k as the set of datapoints at step k, x_k as the global minimizer of $s^k(x)$, and x'_k as the outcome of Algorithm 2.3 for input x_k. Denote δ_k and δ_k^1 as follows:

$$\delta_k^1 = \min_{y \in S^k} \|x_k - y\|, \quad \delta_k = \min_{y \in S^k} \|x'_k - y\|.$$

Note that $S^{k+1} = S^k \cup \{x'_k\}$.

$$0 \leq \min_{z \in S^k} f(z) - f(x^*) \leq (L_p + 2Kr_{\max}^k)\delta_k^1, \tag{2.53}$$

where $r_{\max}^k$ is the maximum circumradius of a Delaunay triangulation for S^k, and L_p is the Lipschitz constant of $p^k(x)$.

Since S^k is a well-situated set with factor of r, according to Theorem 3,

$$r_{\max}^k \leq L_2 r_1^{n-1}. \tag{2.54}$$

where r_1 and L_2 are constants. Moreover, via Lemma 11,

$$\delta_k^1 \leq \rho\delta_k, \tag{2.55}$$

where ρ is a constant defined in (2.46). Thus, using (2.53), (2.54) and (2.55), it follows that

$$0 \leq \min_{z\in S^k} f(z) - f(x^*) \leq \epsilon_k \tag{2.56a}$$

$$\epsilon_k = \rho(L_p + 2KL_2 r_1^{n-1})\delta_k. \tag{2.56b}$$

Since the feasible domain L is bounded, $\delta_k \to 0$ as $k \to \infty$. Thus, Algorithm 2.2, with the adjustment described in Algorithm 2.3 incorporated, will achieve $\epsilon_k \leq \epsilon_{\text{des}}$ in finite k, where ϵ_k defined in (2.56b) for any specified $\epsilon_{\text{des}} > 0$. □

Remark 9. *The parameter $r > 1$ represents a balance between two important tendencies of Algorithm 2.2, with the adjustment described in Algorithm 2.3. For the $r \to 1$, many feasible constraint projections are performed, and thus many datapoints are computed on the boundary of F; as a result, a restrictive bound on the maximum circumradius of the triangulation is available. On the other hand, as r is made large, fewer feasible constraint projections are performed, and thus fewer datapoints are computed on the boundary of F; as a result, the bound on the maximum circumradius of the triangulation is less restrictive. A good balance between these two competing objectives seems to be given by $r = c\sqrt{n}$ where c is an $O(1)$ constant; note that Algorithm 2.2 is recovered in the $r \to \infty$ limit.*

2.6 Adapting K

The tuning parameter K in Algorithms 2.2 and 2.3 specifies the trade-off between global exploration, which is emphasized for large K, and local refinement, which is emphasized for small K. In this section, we develop a method to adjust the tuning parameter K at each iteration k in such a way as to maximally accelerate local refinement while still assuring global convergence.

The method builds on the fact that, if there exists an $\tilde{x}$ such that $p^k(\tilde{x}) - K\, e^k(\tilde{x}) \leq f(x^*)$ where $f(x^*)$ is a global minimum of $f(x)$ at each step k of Algorithm 2.2, then (2.11) is sufficient to establish convergence in Theorems 2.2 and 4, and (2.10) may be relaxed. Furthermore, it is not necessary to choose constant value for K in Algorithm 2.2, instead we may adapt K^k at each step k in such a way that $K^k \geq 0$ is bounded and $p^k(\tilde{x}) - K^k\, e^k(\tilde{x}) \leq f(x^*)$ at each step k of Algorithm 2.2.

If y_0 is a known lower bound for $f(x)$ over the feasible domain L, then if we choose K^k adaptively at each step of Algorithm 2.2 such that

$$0 \leq K^k \leq K_{\max}, \tag{2.57a}$$

$$\exists\, \tilde{x} \in L \quad p^k(\tilde{x}) - K^k\, e(\tilde{x}) \leq y_0, \tag{2.57b}$$

then the Algorithm 2.2 will converge to a global minimizer.

Note that reduced values of K^k accelerate local convergence. Thus, at each step k, we seek the smallest value of K^k which satisfies (2.57). The optimal K^k can be found simply as follows

$$K^k = \min \frac{p^k(x) - y_0}{e^k(x)}. \tag{2.58}$$

It is trivial to verify that the x which minimizes (2.58) also minimizes the corresponding

search function $p^k(x) - K^k\, e^k(x)$. Thus, an alternative definition of the search function is $s_a^k(x) = \frac{p^k(x)-y_0}{e^k(x)}$, which has a same minimizer as $s^k(x) = p^k(x) - K^k e^k(x)$ for the optimal value of K^k given in (2.58).

If at some step k, the solution of (2.58) is negative, we take K^k at that step as zero, and the adaptive search function as $s_a^k(x) = p^k(x)$.

The new search function $s_a^k(x) = \frac{p^k(x)-y_0}{e^k(x)}$ is defined piecewise, similar to the original search function. Thus, we have to solve several optimization problems with linear constraints in order to minimize $s_a^k(x)$ in L. Following similar reasoning as in Lemma 5, we can relax these constraints: for each simplex Δ_i^k, we can instead minimize $s_{a,i}^k(x) = \frac{p^k(x)-y_0}{e_i^k(x)}$ in the intersection of the circumsphere Δ_i^k and the feasible domain L.

Again in order to minimize $s_{a,i}^k(x)$ for each simplex, a good initial guess is required. In each simplex Δ_i^k, a minimizer generally has a large value of $e_i^k(x)$; therefore, the projection of the simplex's circumcenter onto the simplex itself is a good initialization point for searching for the minimum of $s_{a,i}^k(x)$. This initial point for each simplex is denoted $\hat{x}_i^k$.

As before, with this initial guess for the minimum of $s_{a,i}^k(x)$ in each simplex, we can find the global minimum using a Newton method with Hessian modification. We thus need the gradient and Hessian of the function $s_{a,i}^k(x)$:

$$\nabla(\frac{p^k(x) - y_0}{e_i^k(x)}) = \frac{\nabla(p^k(x))}{e_i^k(x)} - (p(x) - y_0)\frac{\nabla e_i^k(x)}{e_i^k(x)^2}$$

$$\nabla^2(\frac{p^k(x) - y_0}{e_i^k(x)}) = \frac{\nabla^2(p^k(x))}{e_i^k(x)} - \frac{(\nabla p^k(x))(\nabla e_i^k(x))^T}{e_i^k(x)^2} - \frac{(\nabla e_i^k(x))(\nabla p(x))^T}{e_i^k(x)^2} + (p^k(x) - y_0)(-\frac{\nabla^2 e_i^k(x)}{e_i^k(x)^2} + 2\frac{\nabla e_i^k(x)\nabla e_i^k(x)^T}{e_i^k(x)^3})$$

Algorithm 2.4 Δ-DOGS with adaptive K parameter.

This algorithm is identical to Algorithm 2.2 except for step 6.c and 6.d, which at step k initially defines the local search functions (upon which the global search function is built) as

$$s_{a,i}^k(x) = \frac{p^k(x) - y_0}{e_i^k(x)}, \tag{2.59}$$

and at step 6.d, the minimizer of $s_{a,i}^k(x)$ is calculated instead of $s_i^k(x)$. but then, if a point x is encountered during this search for which $p^k(x) < y_0$, subsequently redefines the global search function for step k as $s_a^k(x) = p^k(x)$

Remark 10. *Newton's method doesn't always converge to a global minimum. Thus, the result of the search function minimization algorithm at step k, x_k, is not necessarily a global minimizer of $s_a^k(x)$. However, the following properties are guaranteed:*

$$\text{if } s_a^k(x) = p^k(x), \text{ then } p^k(x_k) \leq y_0; \tag{2.60a}$$

$$\text{if } s_a^k(x) = \frac{p^k(x) - y_0}{e^k(x_k)}, \text{ then}$$

$$s_a^k(x_k) \leq s_a^k(\hat{x}_j^k) \ \ \forall \Delta_j^k \in \Delta^k. \tag{2.60b}$$

Recall that $\hat{x}_j^k$ is the maximizer of $e_j^k(x)$ in $\Delta_j^k(x)$.

In the following theorem we prove the convergence of Algorithm 2.4 to the global minimum of $f(x)$. Convergence is based on the conditions in (3.17); note that global minimization of the search function $s_a^k(x)$ at each iteration k is not required.

Theorem 5. *Algorithm 2.4, with the modification described in Algorithm 2.3 incorporated, will converge to the global minimum of the feasible domain L if $f(x)$ and $p^k(x)$ are twice differentiable functions with bounded Hessian, and all $p^k(x)$ are Lipschitz with the same Lipschitz constant.*

Proof. Define S^k, $r_{\max}^k$, and L_2^k as the set of datapoints, the maximum circumradius of Δ^k, and the maximum edge length of Δ^k, respectively, where Δ^k is a Delaunay triangulation

of S^k. Define x_k as the outcome of Algorithm 2.4, which at step k satisfies (3.17), and define x'_k as the outcome of Algorithm 2.3 from input x_k. According to Algorithm 2.3, $S^{k+1} = S^k \cup \{x'_k\}$. Since $f(x)$ and $p^k(x)$ are twice differentiable with bounded Hessian, constants K_f and K_{pf} exist such that

$$K_f \geq \lambda_{\max}(\nabla^2 f(x))/2, \tag{2.61a}$$

$$K_{pf} \geq \lambda_{\max}(\nabla^2 (f(x) - p^k(x))/2. \tag{2.61b}$$

Define L_p as a Lipschitz constant of $p^k(x)$ for all steps k of Algorithm 2.2. Define $y_1 \in S^k$ as the point which minimizes $\delta = \min_{x \in S^k} \|x - x_k\|$.

We will now show that

$$\min_{z \in S^k} f(z) - f(x^*) \leq \bar{\varepsilon}_k, \tag{2.62}$$
$$\bar{\varepsilon}_k = \sqrt{2 r^k_{\max} K_f L_p L_2^k \delta} + [2 r^k_{\max} \max\{K_{pf}, K_f\} + L_p]\delta,$$

where x^* is a global minimizer of $f(x^*)$.

During the iterations of Algorithm 2.4, there two possible cases for $s^k_a(x)$. The first case is when $s^k_a(x) = p^k(x)$. In this case, via (3.17a), $p^k(x_k) \leq y_0$, and therefore $p^k(x_k) \leq f(x^*)$. Since $y_1 \in S^k$, it follows that $p^k(y_1) = f(y_1)$. Moreover, L_p is a Lipschitz constant for $p^k(x)$; therefore,

$$p^k(y_1) - p^k(x_k) \leq L_p\, \delta, \quad f(y_1) - p^k(x_k) \leq L_p\, \delta,$$
$$f(y_1) - f(x^*) \leq L_p\, \delta, \quad \min_{z \in S^k} f(z) - f(x^*) \leq L_p\, \delta,$$

which shows that (2.62) is true in this case.

The other case is when $s_a^k(x) = \frac{p^k(x)-y_0}{e^k(x)}$. For this case, consider Δ_j^k as a simplex in Δ^k which includes x^*. Define $L(x)$ as the unique linear function for which $L(V_i) = f(V_i)$, where V_i are the vertices of the simplex Δ_j^k. According to Lemma 6 and (2.61a), there is an $\tilde{x} \in \Delta_j^k$

$$L(\tilde{x}_1) - K_f\, e^k(\tilde{x}_1) \leq f(x^*). \tag{2.63}$$

Since $L(x)$ is a linear function, it takes its minimum value at one of its vertices; thus,

$$\min_{z \in S^k} f(z) - f(x^*) \leq K_f\, e^k(\tilde{x}_1).$$

Recall $\hat{x}_j^k$ is the point in simplex Δ_j^k which maximizes $e_j^k(x)$ inside the closed simplex Δ_j^k; therefore, $e^k(\tilde{x}_1) \leq e^k(\hat{x}_j^k)$, and

$$\min_{z \in S^k} f(z) - f(x^*) \leq K_f\, e^k(\hat{x}_j^k). \tag{2.64}$$

On the other hand, according to Lemma 4, $2\, r_{\max}^k$ is the Lipschitz norm for the uncertainty function, $y_1 \in S^k$, and thus $e^k(y_1) = 0$; hence,

$$e^k(x_k) \leq 2\, r_{\max}^k\, \delta.$$

If $e^k(\hat{x}_j^k) \leq e^k(x_k)$,

$$\min_{z \in S^k} f(z) - f(x^*) \leq 2\, r_{\max}^k\, K_f \delta;$$

therefore, (2.62) is satisfied. Otherwise,

$$\frac{f(x^*) - y_0}{e^k(\hat{x}_j^k)} \leq \frac{f(x^*) - y_0}{e^k(x_k)}. \tag{2.65}$$

Using (2.60b) and (3.51), it follows:

$$\frac{p^k(x_k) - f(x^*)}{e^k(x_k)} \leq \frac{p^k(\hat{x}_j^k) - f(x^*)}{e^k(\hat{x}_j^k)}. \tag{2.66}$$

According Lemma 6 and (2.61b), there is an $\tilde{x}_2 \in \Delta_j^k$ such that

$$p^k(\tilde{x}_2) - K_{pf}\, e^k(\tilde{x}_2) \leq f(x^*).$$

Since $\tilde{x}_2 \in \Delta_j^k$ and L_p is a Lipschitz constant for $p^k(x)$, it follows that

$$\begin{aligned} p^k(\hat{x}_j^k) - L_p L_2^k - K_{pf}\, e^k(\hat{x}_j^k) &\leq f(x^*), \\ \frac{p^k(\hat{x}_j^k) - f(x^*)}{e^k(\hat{x}_j^k)} &\leq K_{pf} + \frac{L_p L_2^k}{e^k(\hat{x}_j^k)}. \end{aligned} \tag{2.67}$$

Using (2.66), (2.67), and the fact that L_p and $2\, r_{\max}^k$ are Lipschitz constants for $p^k(x)$ and $e^k(x)$, it follows:

$$\begin{aligned} p^k(x_k) - f(x^*) &\leq [K_{pf} + \frac{L_p L_2^k}{e^k(\hat{x}_j^k)}] e^k(x_k), \\ p^k(y_1) - f(x^*) &\leq [2 r_{\max}^k (K_{pf} + \frac{L_p L_2^k}{e^k(\hat{x}_j^k)}) + L_p]\delta. \end{aligned}$$

Note that $y_1 \in S^k$; thus, $p^k(y_1) = f(y_1) \geq \min_{z \in S^k} f(z)$, and

$$\begin{aligned} &\min_{z \in S^k} f(z) - f(x^*) \\ &\leq [2 r_{\max}^k (K_{pf} + \frac{L_p L_2^k}{e^k(\hat{x}_j^k)}) + L_p]\delta. \end{aligned} \tag{2.68}$$

Using (2.64) and (2.68), it follows:

$$\min_{z\in S^k} f(z) - f(x^*) \leq [2r^k_{\max} K_{pf} + L_p]\delta + \frac{2r^k_{\max} K_f L_2^k L_p}{\min_{z\in S^k} f(z) - f(x^*)}\delta.$$

Since x^* is a global minimizer of $f(x)$, $\min_{z\in S^k} f(z) \geq f(x^*)$, and therefore

$$[\min_{z\in S^k} f(z) - f(x^*)]^2 \leq 2r^k_{\max} K_f L_p L_2^k \delta + [2r^k_{\max} K_{pf} + L_p][\min_{z\in S^k} f(z) - f(x^*)]\delta.$$

Apply the quadratic inequality [6] , and the triangular inequality, it follows:

$$\min_{z\in S^k} f(z) - f(x^*) \leq \sqrt{2r^k_{\max} K_f L_p L_2^k \delta} + [2r^k_{\max} K_{pf} + L_p]\delta.$$

The above equation implies that (2.62) is true for this case too. Thus, (2.62) is true for all possible cases.

Moreover, by construction, S^k is a well situated set; thus, by Theorem 3,

$$r^k_{\max} \leq L_2^k r_1^{n-1}, \tag{2.69}$$

Furthermore, via Lemma 11,

$$\delta \leq \rho\, \delta_k, \tag{2.70}$$

where ρ is defined as in (2.46), and $\delta_k = \min_{x\in S^k} \|x'_k - x\|$. Note that the L_k^2 are bounded, and define L_2 as an upper bound for the L_k^2 for all k. Using (2.62), (2.69) and (2.70), it

[6] If $A, B, C > 0$, and $A^2 \leq AB + C$ then $A \leq B + \sqrt{C}$.

follows that,

$$0 \leq \min_{z \in S^{\hat{k}}} f(z) - f(x^*) \leq \epsilon_k, \quad \text{where}$$

$$\epsilon_k = A\delta_k + B\delta_k,$$

$$A = \sqrt{2\rho r^k_{\max} K_f L_p L_2^k},$$

$$B = \rho[2r^k_{\max} \max\{K_{pf}, K_f\} + L_p].$$

Since the feasible domain is bounded, $\delta_k \to 0$ as $\hat{k} \to \infty$. Moreover, A and B are two constants; therefore, $\epsilon_k \to 0$ as $k \to \infty$; thus, the global convergence of Algorithm 2.4, with Algorithm 2.3 incorporated, is assured. □

Remark 11. *Globally minimizing the search function $s^k_a(x)$ at each step k is not required in Algorithm 2.4; it is enough to have* (3.17) *to guarantee convergence. In contrast, the search function $s^k(x)$ must be globally minimized in order to guarantee convergence of Algorithm 2.2.*

Since performing Newton's method for minimizing the search function is not required for global convergence, in practice, we will perform Newton's method only in those simplices whose initial points $\hat{x}^k_j$ have small values for the adaptive search function $s^k_a(x)$. In general, performing Newton iterations in more simplices reduces the number of function evaluations required for convergence, but increases the cost of each optimization step.

2.6.1 Using an inaccurate estimate of y_0

It was shown in Theorem 5 that, using Algorithm 2.4 with a lower bound y_0 for the function, convergence to a global minimum of the function is guaranteed. However, it is observed in Section 7.4 that, if y_0 is not an accurate lower bound for the global minimum,

the speed of convergence is reduced. In this subsection, we study the behavior of Algorithm 2.4, when the estimated value of y_0 is slightly *larger* than the actual minimum of the function of interest.

Theorem 6. *Assume that $f(x)$ and $p^k(x)$ are twice differentiable functions with bounded Hessian, and all $p^k(x)$ are Lipschitz with the same Lipschitz constant. Then, for any small positive $\varepsilon > 0$, there is a finite iteration k of Algorithm 2.4, with the modification described in Algorithm 2.3 incorporated, such that a point $z \in S^k$ exists for which:*

$$f(z) - \max\{f(x^*), y_0\} \leq \varepsilon, \tag{2.71}$$

where $f(x^*)$ is the global minimum of $f(x)$.

Proof. To prove this theorem, we use the notations defined in the proof of Theorem 5. Note that, if $y_0 \leq f(x^*)$, the theorem is true based on Theorem 5. Otherwise, similar to (2.62), we will show that

$$\min_{z \in S^k} f(z) - y_0 \leq \sqrt{2r^k_{\max} K_f L_2^k L_p \delta} + [L_p + 2r^k_{\max} K_{pf}]\delta.$$

As stated previously, during the iterations of Algorithm 2.4, there two possible cases for $s^k_a(x)$. The first case is when $s^k_a(x) = p^k(x)$. In this case, via (3.17a), $p^k(x_k) \leq y_0$. Since $y_1 \in S^k$, it follows that $p^k(y_1) = f(y_1)$. Moreover, L_p is a Lipschitz constant for $p^k(x)$;

therefore,

$$p^k(y_1) - p^k(x_k) \leq L_p\,\delta,$$

$$f(y_1) - p^k(x_k) \leq L_p\,\delta,$$

$$f(y_1) - y_0 \leq L_p\,\delta,$$

$$\min_{z\in S^k} f(z) - y_0 \leq L_p\,\delta,$$

which shows that (3.47) is true in this case.

The other case is when $s_a^k(x) = \frac{p^k(x)-y_0}{e^k(x)}$. In this case, (2.60b) is satisfied. Now, similar to the proof of Theorem 5, define Δ_j^k as a simplex in Δ^k which includes a global minimizer x^*. Following the same reasoning as in the proof of Theorem 5, (2.64) is true. Moreover, $y_0 \geq f(x^*)$, and thus

$$\min_{z\in S^k} f(z) - y_0 \leq K_f\, e^k(\hat{x}_j^k).$$

In addition, if $\min_{z\in S^k} f(z) \leq y_0$, the theorem is trivial, otherwise, the above equation may be modified to

$$\frac{1}{e^k(\hat{x}_j^k)} \leq \frac{K_f}{\min_{z\in S^k} f(z) - y_0}. \tag{2.72}$$

Note that (2.67) is true in this case too. Using (2.60b), $y_0 \geq f(x^*)$, and the Lipschitz properties of $p^k(x)$ and $e^k(x)$, it follows that

$$\begin{aligned} &\min_{z\in S^k} f(z) - y_0 \\ &\leq [2r_{\max}^k(K_{pf} + \frac{L_p L_2^k}{e^k(\hat{x}_j^k)}) + L_p]\delta. \end{aligned} \tag{2.73}$$

Using (2.72), (2.73), and $\min_{z\in S^k} f(z) \geq y_0$, it follows that

$$[\min_{z\in S^k} f(z) - y_0)]^2 \leq 2r_{\max}^k K_f L_p L_2^k \delta + [2r_{\max}^k K_{pf} + L_p][\min_{z\in S^k} f(z) - f(x^*)]\delta.$$

It follows from the above equation that (2.62) is true for all possible cases. Note that (2.69) and (2.70) are true in this theorem too; thus, with similar reasoning, it follows:

$$0 \leq \min_{z\in S^{\hat{k}}} f(z) - y_0 \leq \epsilon_k, \quad \epsilon_k = A\sqrt{\delta_k} + B\delta_k,$$
$$A = \sqrt{2\rho L_2^2 r_1^{n-1} K_f L_p}, \quad B = \rho[L_p + 2L_2 r_1^{n-1} K_{pf}].$$

where $\delta_k = \min_{x\in S^k} \|x_k' - x\|$. Since the feasible domain is bounded, $\delta_k \to 0$ as $k \to \infty$; thus, $\epsilon_k \to 0$ as $k \to \infty$. □

Remark 12. *Theorem 6 shows that if an inaccurate guess for the lower bound y_0 of the function $f(x)$ is used, Algorithm 2.4 will converge to a point whose function value is equal to or below the estimate y_0. In other words, Algorithm 2.4 will first converge to a point whose function value is less than y_0, and then perform only local refinements thereafter, taking $s^k(x) = p^k(x)$ for later iterations.*

2.7 Parallelization

Parallel computing is one of the most powerful tools of modern numerical methods. In expensive optimization problems, performing an optimization of the type discussed here on a single CPU is relatively slow. The algorithms presented above only accommodate serial computations, with one function evaluation performed in each step.

The present algorithms are intended for problems with expensive function evalua-

tions. Other than these function evaluations, the most expensive part of the present algorithms are the search function minimizations. Constructing the Delaunay triangulation is also somewhat expensive in high-dimensional problems; however, this is made negligible in comparison with the other parts of the algorithm when an Incremental method is used to update the Delaunay triangulation from one step to the next.

The search function minimization allows for straightforward parallel implementation. As described before, we must minimize the local search functions $s_i^k(x)$ [or, in Algorithm 2.4, $s_{a,i}^k(x)$] within each simplex of the Delaunay triangulation at step k; this task is embarrassingly parallel.

The other expensive part (which will be the most expensive part in many applications) is the function evaluations themselves. We may modify Algorithms 2.2 and 2.4 to perform n_p function evaluations in parallel. To accomplish this, we need to identify n_p new points to evaluate at each step.

During Algorithm 2.2 or 2.4, x_k is derived by minimizing $s^k(x) = p^k(x) - Ke^k(x)$ or $s_a^k(x) = \frac{p^k(x)-y_0}{e^k(x)}$. Note that, at each step, the uncertainty function $e^k(x)$ is independent from the interpolation $p^k(x)$ and the function values themselves. Thus, we can modify $e^{k+1}(x)$ without performing the cost function evaluation at x_k. This idea may be implemented as follows

Note that minimizing $s^{k_i}(x)$ for $0 < i < n_p$ is a relatively easy task, since $s^{k_i}(x) = s^{k_{i-1}}(x)$ in most of the simplices, and the incremental implementation of the Delaunay triangulation can be used to flag the indices of those simplices that have been changed by adding x_i^k to $S_{i-1}^k(x)$ (see [22]).

Remark 13. *It is easy to show that the modification proposed in Algorithm 2.5 preserves the convergence properties of Algorithms 2.2 and 2.4. The parameter c at step* 4.*d plays*

Algorithm 2.5 In this algorithm, a modification for Algorithm 2.2 is presented is which, at each step k, identifies n_p new points to evaluate in parallel at each step. Note that this approach extends immediately to Algorithm 2.4.

Take the set of initialization points S^0 as all M of the vertices of the feasible domain L together with one or more user-specified points of interest on the interior of L. Evaluate the function $f(x)$ at each of these initialization points in parallel. Perform a Delaunay triangulation for S^0. Set $k = 0$.

Calculate (or, for $k > 0$, update) an appropriate interpolating function $p^k(x)$ through all points in S^k.

Calculate x_k^0 as the minimizer of $s^k(x)$ (see step 3 and 4 of Algorithm 2.2). This task may be done in parallel for each simplex.

Replace x_k^0 with the outcome of Algorithm 2.3 from input x_k^0, then take $S_1^k = S^k \cup \{x_k^0\}$.

For each substep $i \in \{1, 2, \ldots, n_p - 1\}$, do the following:

a. Incrementally calculate the Delaunay triangulation for data points for S_i^k in order to derive the new uncertainty function $e^{k_i}(x)$.

b. Derive x_k^i as a global minimizer of $s^{k_i}(x) = p^k(x) - Ke^{k_i}(x)$.

c. Calculate $\delta_i^k = \min_{y \in S_i^k} \|x_k^i - y\|$,
and $\delta_0^k = \min_{y \in S^k} \|x_k^0 - y\|$.

d. If $\delta_i^k \leq c\delta_0^k$ for some c with $0 < c \leq 1$, replace x_k^i with a global minimizer of $e^{k_i}(x)$.

e. Replace x_k^i with the outcome of Algorithm 2.3 from input x_k^i.

f. Take $S_{i+1}^k(x) = S_i^k \cup \{x_k^i\}$.

Take $S^{k+1} = S_{n_p}^k$, and evaluate the function at
$\{x_k^0, x_k^1, \ldots, x_k^{n_p - 1}\}$ in parallel.

6. Repeat from step 2 until $\delta_0^k \leq \delta_{des}$.

a significant role in the convergence rate; large c forces more of the function evaluations to be related strictly to global exploration, whereas small c allows function evaluations to potentially get dense in regions away from the global minimum. Intermediate values of c are thus preferred, as discussed further in §2.8.5.

2.8 Results

To evaluate the performance of our algorithms, we applied them to the minimization of the some representative test functions (see [36]):

- **One dimension**

  Weierstrass function: [7] Taking $N \gg 1$ ($N = 300$ in the simulations performed here),

  $$f(x) = \sum_{i=0}^{N} \frac{1}{2^i} \cos(3^i \pi\, x) \tag{2.74}$$

- **Two dimensions**

  Parabolic function:

  $$f(x,\, y) = x^2 + y^2 \tag{2.75}$$

  Schwefel fuction: Defining $c = 418.982887$,

  $$f(x,\, y) = c - x \sin\left(\sqrt{|x|}\right) - y \sin\left(\sqrt{|y|}\right) \tag{2.76}$$

[7] The parameters of the Weierstrass function used in this chapter do not satisfy the condition assuring nondifferentiability everywhere that Weierstrass originally identified; however, according to [37], these parameters indeed assure nondifferentiability of the Weiertrass function everywhere as $N \to \infty$.

Rastrigin function: Defining $A = 2$,

$$f(x,\, y) = 2\,A + x^2 + y^2 + -A\cos(2\pi\, x) - A\cos 2\pi\, y \tag{2.77}$$

Rosenbrock function: Defining $p = 10$,

$$f(x,\, y) = (1 - x^2) + p\,(y - x^2)^2 \tag{2.78}$$

- **Higher dimensions**

  Rastrigin function: Defining $A = 2$,

$$f(x_i) = A\,n + \sum_{i=1}^{n}\left[x_i^2 - A\cos(2\pi\, x_i)\right] \tag{2.79}$$

  Rosenbrock function: Defining $p = 10$,

$$f(x_i) = \sum_{i=1}^{n-1}\left[(1 - x_i)^2 + p\,(x_{i+1} - x_i^2)^2\right] \tag{2.80}$$

Except where otherwise stated, each simulation was initialized solely with function evaluations at each vertex of the feasible domain; that is, no user-specified points were used during the initialization. What follows is a description of the results of the simulations performed.

Also, unless otherwise stated, each test result reported incorporates Algorithm 2.3 into Algorithms 2.2 and 2.4, with parameter $r = 2\sqrt{n}$.

2.8.1 Nondifferentiable 1D case

The Weierstrass function is a classical example of a nondifferentiable function, for which derivative-based optimization approaches are ill-suited. Note that the proofs of convergence of the present algorithms, as developed above, don't even apply in this case; however, it is seen that the algorithms developed converge quickly regardless.

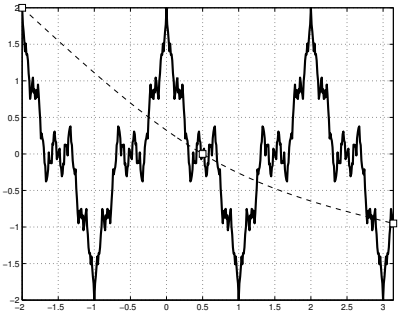
We sought a global minimum of this test function over the domain $[-2, \pi]$. For this test function (only), the set of initial data points was taken as $S^0 = \{-2, 0.5, \pi\}$. The result using Algorithm 2.2 with $K = 0$ is illustrated in Figure 2.7a, the result using Algorithm 2.2 with $K = 100$ is illustrated in Figure 2.7b, and the result using Algorithm 2.4 (with adaptive K, taking $y_0 = -2$, which is the known lower bound of $f(x)$) is illustrated in Figure 2.7c. The optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$.

It is seen in the $K = 0$ case that the optimization routine terminated prematurely, and the global minimum of the problem was not identified; it is thus seen that the global exploration part of the algorithm (for $K > 0$) is important. It is seen in the $K = 100$ case that global convergence was achieved, and that 34 function evaluations were required for convergence. It is seen in the adaptive K case that global convergence was achieved more rapidly, requiring only 17 function evaluations for convergence.

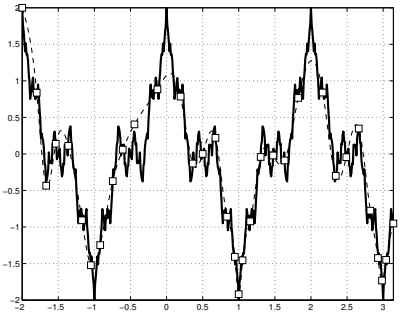
2.8.2 Box constraints

2D parabola

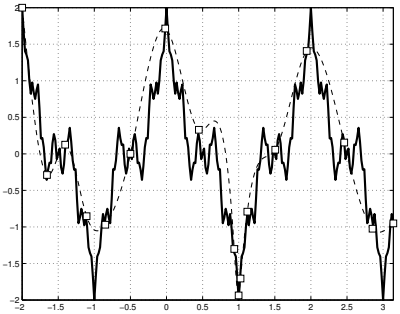
The first 2D test function considered is a parabola (2.75). This simplistic test was made to benchmark the effectiveness of the algorithm on a trivial convex optimization problem. The function considered has a global minimizer at (0, 0) in the feasible domain



a Algorithm 2.2 with $K = 0$,



b Algorithm 2.2 with $K = 100$,



c Algorithm 2.4 with $y_0 = -2$.

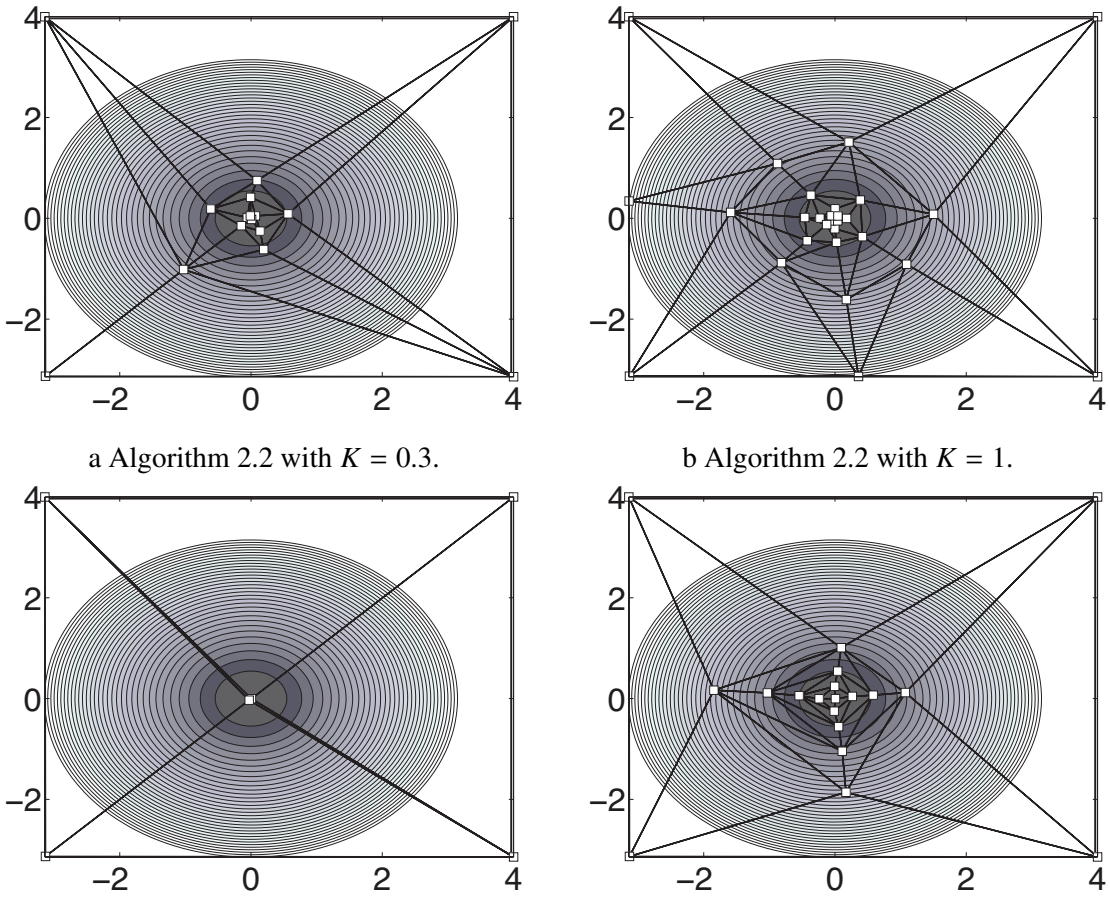
Figure 2.7: Δ-DOGS on Weierstrass function. Implementation of Algorithms 2.2 and 2.4 for the Weierstrass function (2.74). Actual function (solid), function evaluations (squares), and interplant after the final function evaluation (dashed).

$x_i \in [-\pi, 4]$ (the center of the domain is shifted away from the origin to make the minimum nontrivial to find). Again, the optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. For this problem, for both small K ($K = 0.3$, see Figure2.8a) and larger K ($K = 1$, see Figure 2.8b), global convergence was achieved; 16 function evaluations were required for convergence of this simple function with $K = 0.3$, and 29 function evaluations were required for convergence with $K = 1$. For the larger value of K, a bit more global exploration is evident. Taking $K = 0$ in the present case again results in premature termination of the optimization algorithm, at the very first step, and the global minimum in not identified

Given exact knowledge of y_0, the behavior of Algorithm 2.4 for this simple test function is remarkable, and the algorithm converges in only 6 function evaluations (that is, two function evaluation after evaluating the function at the vertices of L). As shown in Figure 2.8d, taking y_0 as a bound on the minimum, $y_0 = -0.1$, the algorithm requires a few more iterations (19 function evaluations are needed). In this case, Algorithm 2.4 actually performs a function evaluation very near the global minimizer within the first two iterations, similar to the case when $y_0 = 0$; however, the algorithm continues to explore a bit more, until it confirms that no other minima with reduced function values exist near this point.

2D Schwefel

The second 2D test function considered is the Schwefel function (2.76), which is characterized by nine local minima over the domain considered, $x_i \in [0, 500]$, with the global minimizer at (420.968746, 420.968746), and global minimum of $f(x^*) = 0$. The optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 1$. As shown in Figure 2.9a, for $K = 0.03$, the algorithm fails to converge to the global minimum, as not enough global



a Algorithm 2.2 with $K = 0.3$.

b Algorithm 2.2 with $K = 1$.

c Algorithm 2.4 with $y_0 = 0$.

d Algorithm 2.4 with $y_0 = -0.1$.

Figure 2.8: Δ-DOGS on 2D parabola. Location of function evaluations using Algorithms 2.2 and 2.4 applied to the 2D parabola (2.75).

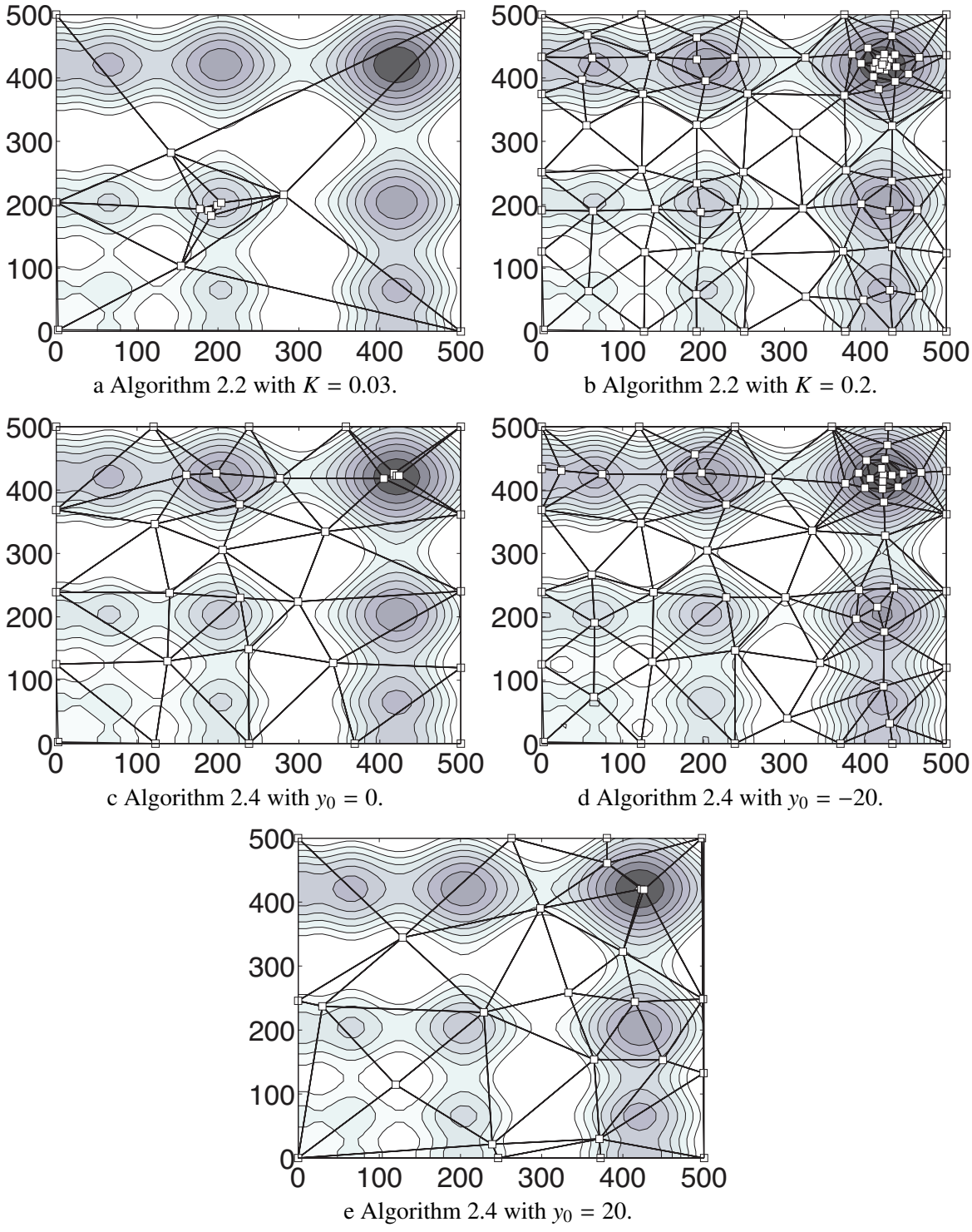
exploration is performed. As shown in Figure 2.9b, taking $K = 0.2$, and thus performing more global exploration, the algorithm succeeds in finding the global minimum after 87 function evaluations. As shown in Figure 3.8d, using adaptive K based on accurate knowledge of global minimum $y_0 = 0$, the result is similar to the $K = 0.3$ case, but only 36 function evaluations are performed; convergence is seen to be especially rapid once the neighborhood of the global minimum was identified. As shown in Figure 2.9d, using adaptive K based on a bound on the global minimum, $y_0 = -20$, the algorithm continues to explore a bit more, now requiring 64 function evaluations for convergence. As shown in

Figure 2.9e, using adaptive K based on an inaccurate guess of the bound on the global minimum, $y_0 = 20$, convergence is similar to the case with $y_0 = 0$ (Figure 3.8d), with actually a bit faster convergence (now requiring 26 function evaluations for convergence), because global exploration is suspended (that is, K is driven to zero) once function values below y_0 are discovered, with the algorithm thereafter focusing solely on local refinement; recall from Theorem 6 that, in this case, the algorithm may stop any time after function values below y_0 are discovered, and convergence to a neighborhood of the global minimum is not assured.

2D and 3D Rastrigin

The third test function considered is the Rastrigin function. We first consider the 2D case (2.77), which is characterized by 36 local minima over the domain considered, $x_i \in [-2, \pi]$, with the global minimum at the origin. The results for this test function are presented in Figures 2.10b, 2.10a, and 2.10c, and are similar to the Schewefel test case. Algorithm 2.2 fails to converge to the global minimum when $K = 10$, which is too small for this problem, and more extensive global exploration was performed when $K = 20$, in which case convergence was achieved in 63 function evaluations. More efficient convergence was obtained when Algorithm 2.4 (adaptive K) was used with an accurate value of $y_0 = 0$, requiring only 30 function evaluations for convergence.

In Figure 2.11, we consider the 3D case (2.79), which is characterized by 216 local minima over the domain considered, $x_i \in [-2, \pi]$, with the global minimum at the origin. We applied Algorithm 2.4 with an accurate value of $y_0 = 0$, and terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. During the first iteration after the initialization, a point was obtained with function value close to the global minimum; however, several more iterations



a Algorithm 2.2 with $K = 0.03$.

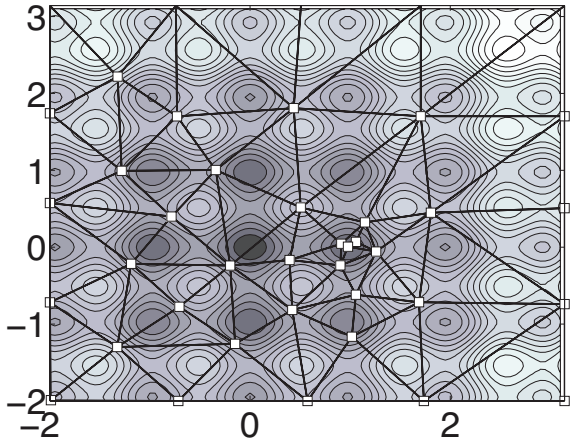
b Algorithm 2.2 with $K = 0.2$.

c Algorithm 2.4 with $y_0 = 0$.

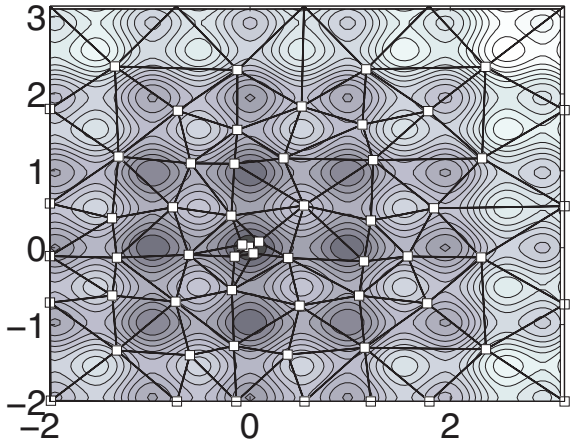
d Algorithm 2.4 with $y_0 = -20$.

e Algorithm 2.4 with $y_0 = 20$.

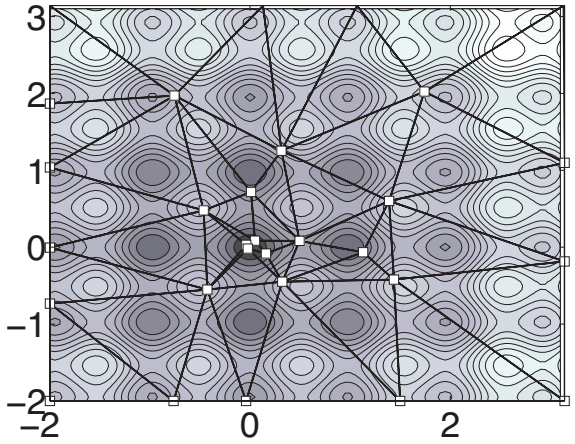
Figure 2.9: Δ-DOGS on 2D Schwefel function. Location of function evaluations in Algorithms 2.2 and 2.4 on the 2D Schwefel function (2.76).



a Algorithm 2.2 with $K = 10$.

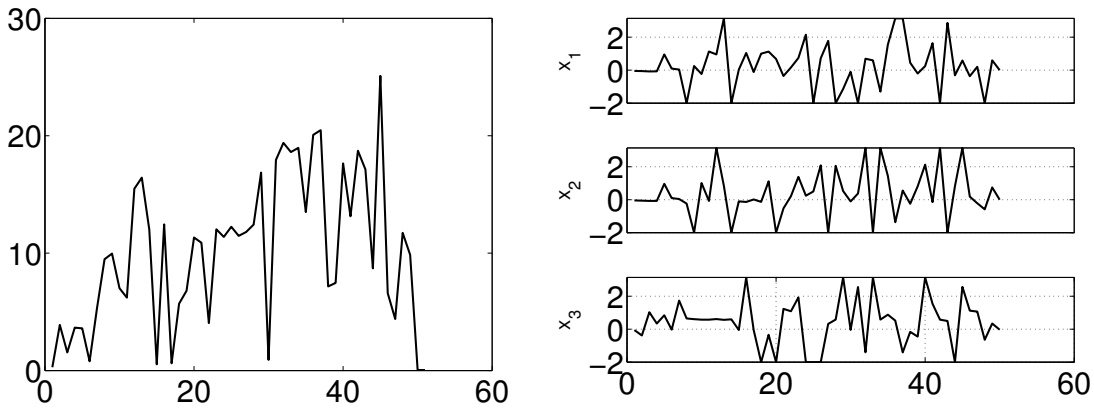


b Algorithm 2.2 with $K = 20$.



c Algorithm 2.4 with $y_0 = 0$.

Figure 2.10: Δ-DOGS on 2D Rastrigin function. Location of function evaluations in Algorithms 2.2 and 2.4 on the 2D Rastrigin function (2.77).



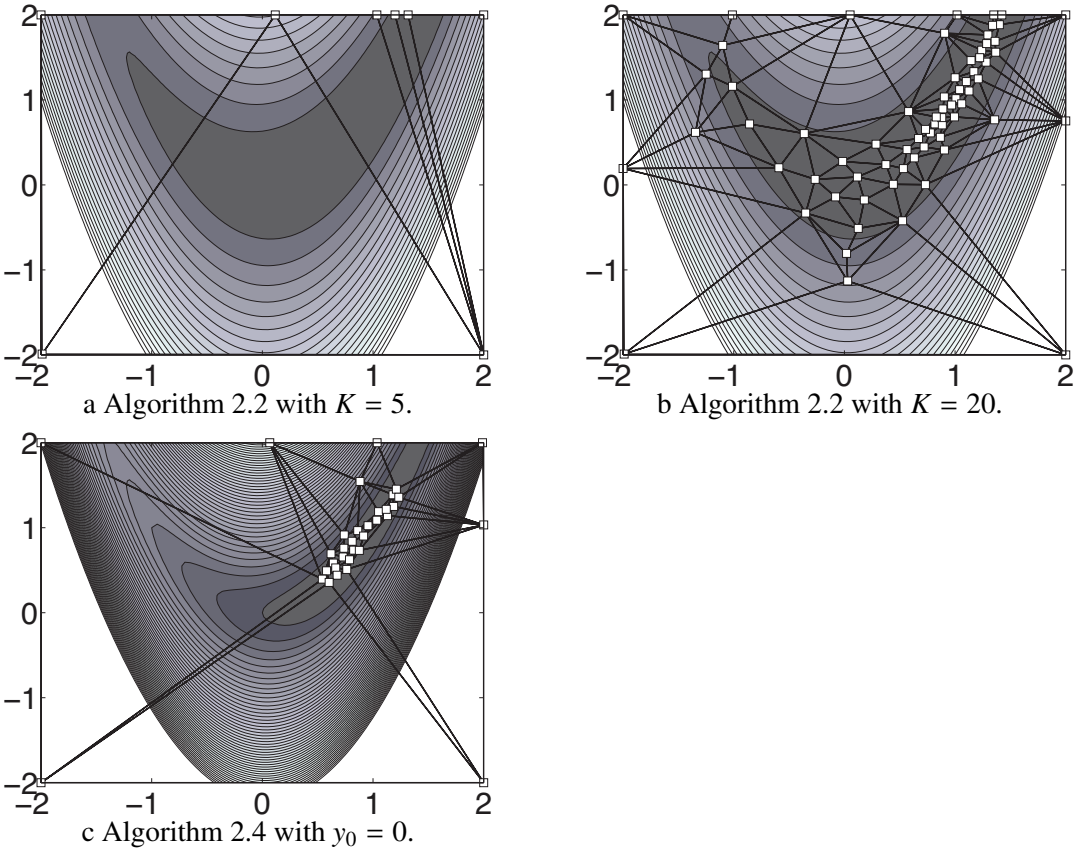
a Function values at the evaluation points. b Coordinates of the evaluation points.

Figure 2.11: Implementation of Algorithm 2.4 with $y_0 = 0$ on the 3D Rastrigin function (2.79).

were required until the algorithm stopped after 50 iterations (i.e., including the vertices of L, 58 function evaluations).

2D and 3D Rosenbrock

The next test function considered is the Rosenbrock function. We first consider the 2D case (2.78), which is characterized by a single minimum over the domain considered, $x_i \in [-2\ 2]$, with the global minimum at $(1,\ 1)$, and a challenging, nearly flat valley along the curve $y = x^2$ where the global minimum lies. In this test function, the optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$; since the valley is so flat in this case, the accuracy of the converged solution is a strong function of the termination criterion, and significantly relaxing this criterion leads to inaccurate results. As shown in Figure 2.12a, for $K = 5$, the algorithm fails to converge to the global minimum, as not enough global exploration is performed. As shown in Figure 2.12b, a larger value of the tuning parameter, $K = 20$, facilitates more thorough global exploration over the domain, with the function evaluations concentrating along the valley, and convergence is achieved in 70

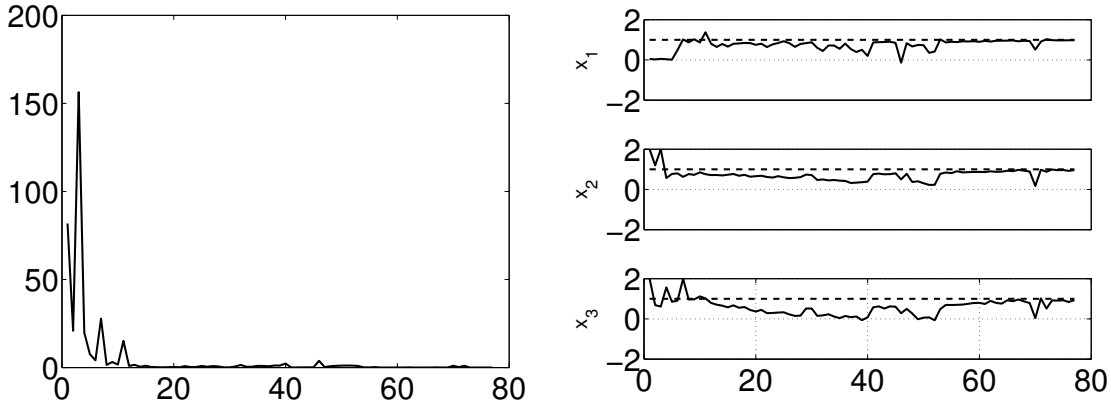


a Algorithm 2.2 with $K = 5$.

b Algorithm 2.2 with $K = 20$.

c Algorithm 2.4 with $y_0 = 0$.

Figure 2.12: Δ-DOGS on 2D Rosenbrock function. Location of function evaluations in Algorithms 2.2 and 2.4 on the 2D Rosenbrock function (2.78).



a Function values at the evaluation points. b Coordinates of the evaluation points.

Figure 2.13: Δ-DOGS on 3D Rosenbrock function. Implementation of Algorithm 2.4 with $y_0 = 0$ on the 3D Rosenbrock function (2.80).

function evaluations. As shown in Figure 2.12c, applying Algorithm 2.4 (adaptive K) using an accurate value of $y_0 = 0$ focused the function evaluations even better along the valley of the function, and convergence is achieved in only 34 function evaluations.

In Figure 2.13, we consider the 3D case (2.80), which is again characterized by a single minimum over the domain considered, $x_i \in [-2\,2]$, with the global minimum at (1, 1, 1), and a challenging region near the 3D curve $x_3 = x_2^2 = x_1^4$ where the function is nearly "flat". We applied Algorithm 2.4 with an accurate value of $y_0 = 0$, and terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. Similar to the 2D case, after a brief exploration of the feasible domain, the algorithm soon concentrates function evaluations near the $x_3 = x_2^2 = x_1^4$ curve where the reduced function values lie, as shown in Figure 2.13b, and convergence is achieved in 76 iterations.

2.8.3 General linear constraints

The above tests were all performed using simple box constraints (2.1b), $a \leq x \leq b$. We now test the performance of Algorithm 2.4 with $y_0 = 0$ when more general linear

constrains (2.1a) are applied, $Ax \leq b$.

We first consider the 2D Rastrigin function (2.77) with the following linear constraints:

$$-2 \leq x, \quad x \leq \pi, \quad -2 \leq y, \tag{2.81a}$$

$$y \leq \pi, \quad x \leq y, \quad x + y \leq 1. \tag{2.81b}$$

During the initialization step, after finding the vertices of L, it is determined that three constraints are redundant. Thus, the feasible domain (in general, a convex polyhedron) is actually a simplex in this case, bounded by other three constraints. The global minimum in this case lies on one of the constraint boundary; as shown in Figure 2.14, Algorithm 2.4 converges after initially exploring the feasible domain with 17 function evaluations.

Next, consider the 2D Rosenbrock function (2.78) with the following linear constraints:

$$-2 \leq x \leq 2, \quad -2 \leq y \leq 2, \tag{2.82a}$$

$$-2.2 \leq x + y, \quad x + y \leq 2.2. \tag{2.82b}$$

During the initialization step, it is determined that none of the constraints are redundant, since each constraint is active at exactly two vertices. As shown in Figure 2.15, the feasible domain in this case is a convex polygon with six vertices. The global minimum in this case lies near, but not on the constraint boundary. As expected, the results are quite similar to the case with box constraints (see Figure 2.12), and the global minimum is found with 27 function evaluations.

Finally, we considered the 3D Rastrigin and Rosenbrock functions, (2.79) and

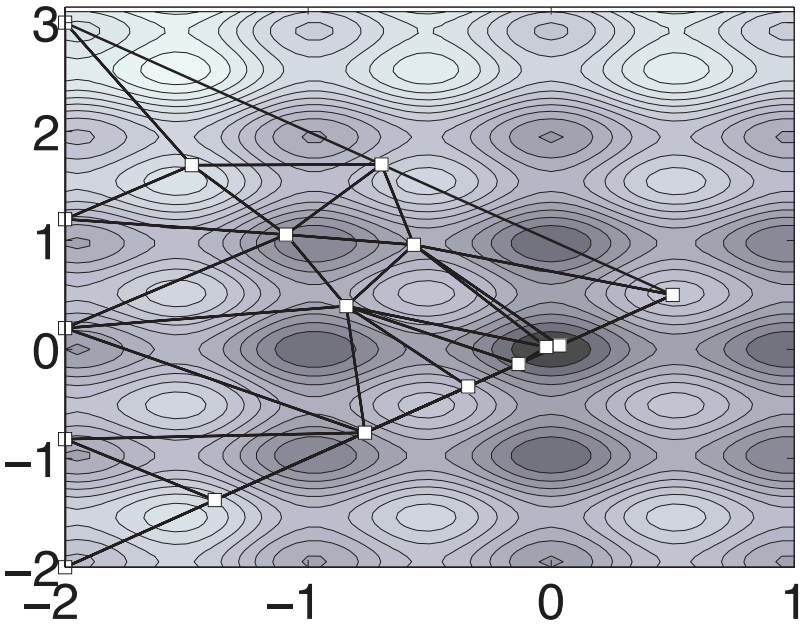


Figure 2.14: Rastrigin function in 2D with linear constraints

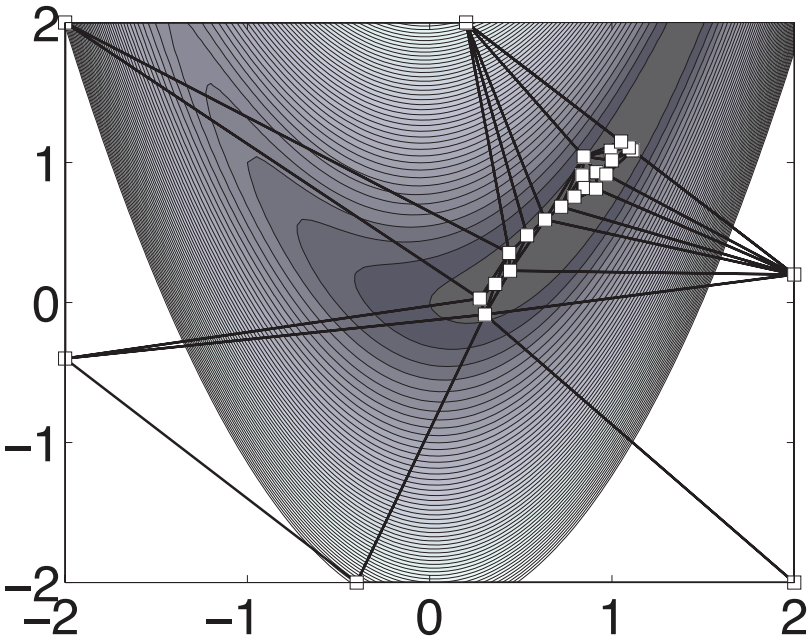


Figure 2.15: Rosenbrock function in 2D with linear constraints

(2.80), with the following linear constraints:

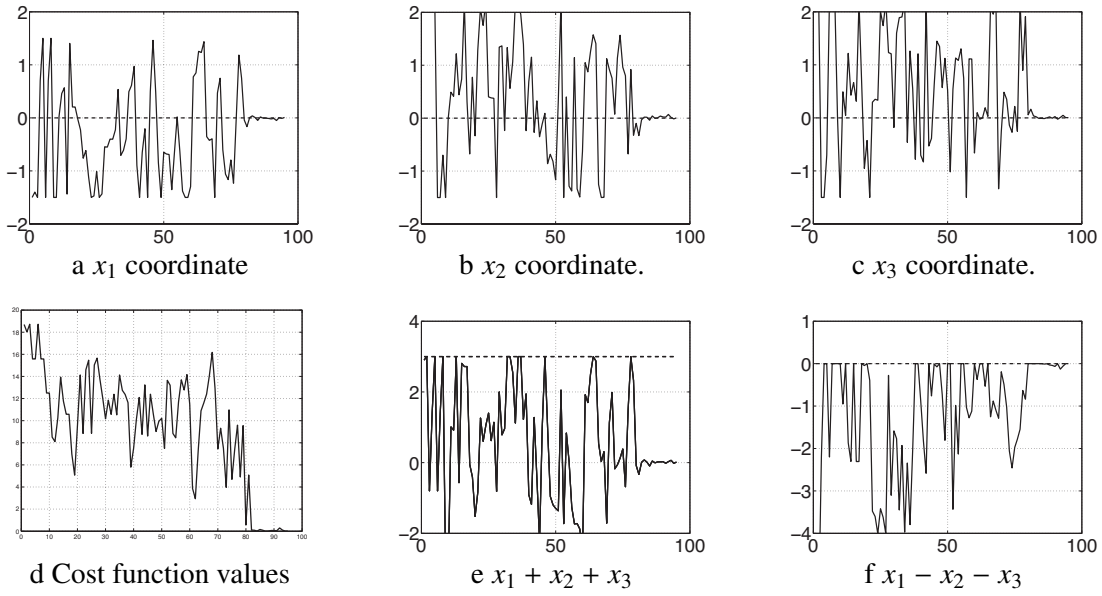


Figure 2.16: Implementation Algorithm 2.4 on the 3D Rastrigin function (2.79) with linear constraints.

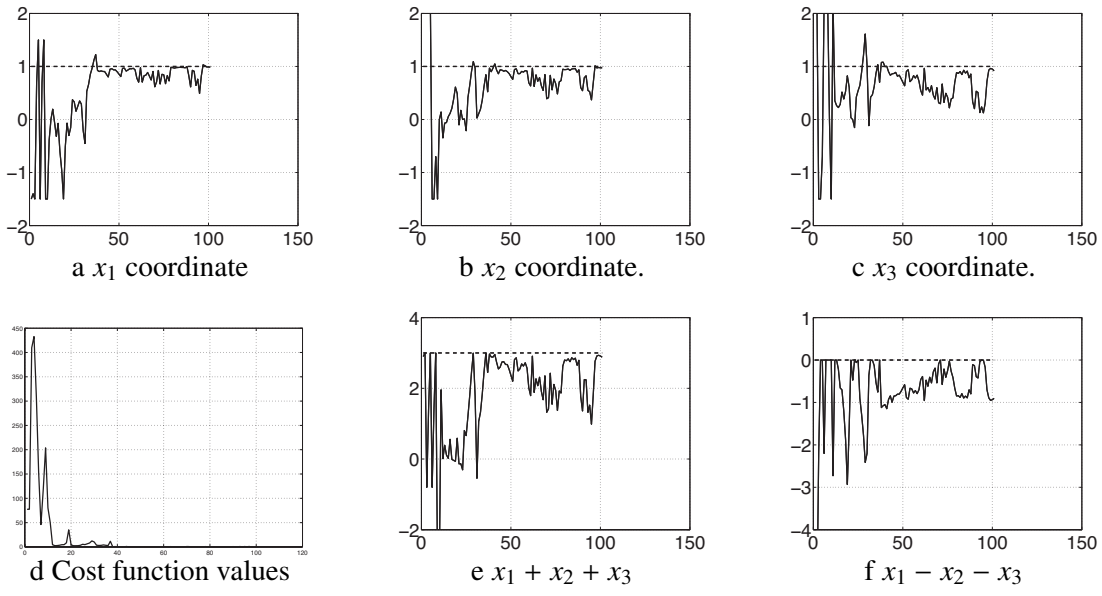


Figure 2.17: [Δ-DOGS on 3D Rosenbrock function with linear constraints. As in Figure 2.16 for the 3D Rosenbrock function (2.80).

$$-2 \leq x_i \leq 2 \quad \text{for } 1 \leq i \leq 3, \tag{2.83a}$$

$$x_1 + x_2 + x_3 \leq 3, \tag{2.83b}$$

$$x_1 - x_2 - x_3 \leq 0. \tag{2.83c}$$

During the initialization step, it is determined that the constraint $x_1 \leq 2$ is the only redundant constraint, since each of the other constraints is active at at least three of the vertices. The feasible domain in this case is a convex polyhedron with 10 vertices; it turns out that the constraint in (2.83) is active at six vertices, so one of the faces of this polyhedron is a hexagon. As shown in Figures 2.16 and 2.17, the behavior of Algorithm 2.4 is similar to the corresponding tests with box constraints discussed previously. Note, of course, that all function evaluations performed by Algorithm 2.4 are evaluated at feasible points.

2.8.4 Feasible constraint projections

We now explore the role of the feasible boundary projections introduced in Definition 5, and incorporated into Algorithm 2.3, on the convergence of Algorithm 2.2 with $K = 1$, focusing specifically on the impact of the r parameter, taking $r = 1.05$, $r = 5$ and $r = 30$. We perform this test using the 2D parabolic function (2.75) subject to the following linear constraints:

$$x \leq 0.1, \quad -1.1 \leq y, \quad y - x \leq 0.5. \tag{2.84}$$

The location of the function evaluations for different values of r is shown in Figure 2.18.

Recall that $1 < r < \infty$, with the $r \to \infty$ limit suppressing all feasible constraint projections. It is observed that, for small values of r, the algorithm tends to explore more

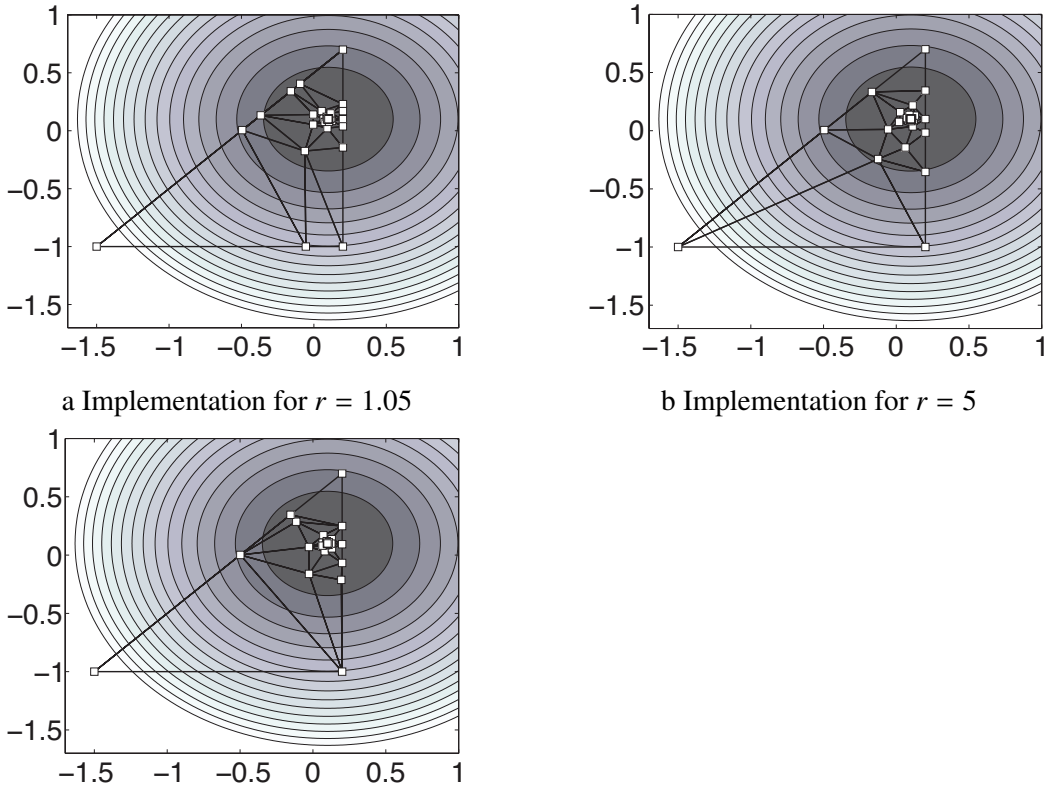
on the boundaries of the feasible domain, and for large values of r, the triangulation is more irregular, with certain function evaluations clustered in a region far from the global minimum. Intermediate values of r are thus preferred.

Figure 2.19 plots the maximum circumradius $r^k_{\max}$ of the Delaunay triangulation Δ^k, as well as the upper bound for $r^k_{\max}$, during the optimization using Algorithm 2.2 with Algorithm 2.3 incorporated using different values of r. The maximum circumradius $r^k_{\max}$ is seen to be reduced when smaller values of r are used; however, the cases with $r = 1.05$ and $r = 5$ are not noticeably different in this respect. Another important observation is that the bound on the maximum circumradius, given by (2.54), is also reduced when smaller values of r are used; however, this bound is seen to be quite conservative in this example.

2.8.5 Parallel performance

We now test the performance of the constant-K version of Algorithm 2.5 on the Weierstrass function (2.74), over the domain $[-2, \pi]$ using $n_p = 3$ processors, taking $K = 15$ and, in turn, $c = 0$, $c = 0.5$ and $c = 1$. The optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. Algorithm 2.5 fails to converge to the global minimum when $c = 0$, as multiple function evaluations are performed at a single step k that are close to each other in this case, which causes premature termination of the algorithm. Algorithm 2.5 converges to the global minimum for $c = 0.5$ in 18 function evaluations, and for $c = 1$ in 21 function evaluations; it is thus seen that intermediate values of c are preferred. In the $c = 0.5$ case, after the initialization, 6 iterations were executed, with 7 function evaluations performed in parallel during each iteration.

Testing Algorithm 2.2 with $K = 15$ on the same problem, it is seen that fewer (in this case, 13) function evaluations are required by the serial version of the algorithm,

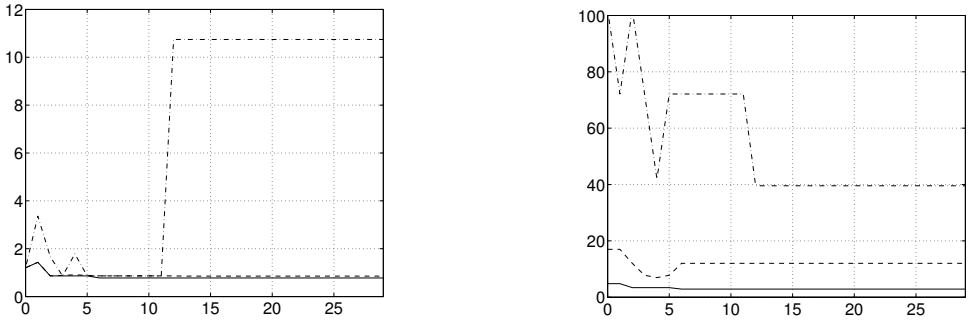


a Implementation for $r = 1.05$

b Implementation for $r = 5$

c Implementation for $r = 30$

Figure 2.18: Role of feasible boundary projection. The location of function evaluations by Algorithm 2.2, with Algorithm 2.3 incorporated, for different values of r. The cost function is simple quadratic function whose minimizer is an interior point within a feasible domain given by a right triangle.



a Actual value of $r^k_{\max}$ b Theoretical upper bound for $r^k_{\max}$

Figure 2.19: The actual value and the theoretical bound for the maximum circumradius $r^k_{\max}$ of Δ^k, as a function of k, for the optimizations illustrated in Figure 2.18. Solid line, dashed line, and dot-dashed line are the results for $r = 1.05$, $r = 5$ and $r = 30$ respectively.

as the location of each new function evaluation is based on the trends revealed in all of the previous function evaluations. However, the total number of iterations that need to be executed in this case is increased from 6 to 10, thus demonstrating the benefit of the parallel algorithm when performing function evaluations in parallel is possible.

2.9 Conclusions

In this chapter, we have presented a new response surface method for derivative-free optimization of nonconvex functions within a feasible domain L bounded by linear constraints. The chapter developed five algorithms:

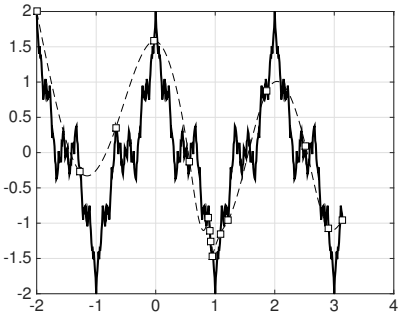
- Algorithm 2.1 showed how to initialize the problem, identifying the vertices of L, eliminating the redundant constraints, and projecting the equality constraints out of the problem.
- Algorithm 2.2 presented the essential strawman form of the method. It uses any well-behaved interpolation function of the user's choosing, and a synthetic piecewise-quadratic uncertainty function built on the framework of a Delaunay triangulation.

A search function given by a linear combination of the interpolation and a model of the uncertainty is minimized at each iteration. The search function itself is piecewise smooth, and may in fact be nonconvex within certain simplices of the Delaunay triangulation. A valuable feature of the algorithm is that global minimization of the search function within each simplex is, in fact, not required at each iteration; convergence to the global minimum can be guaranteed even if the algorithm only locally minimizes the search function within each simplex at each iteration. Unfortunately, this simple algorithm contains an important technical flaw: it does not ensure that the triangulation remains well behaved as new datapoints are added.

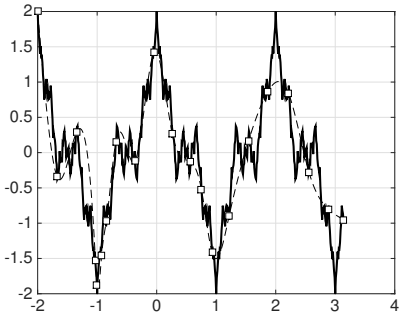
- Algorithm 2.3 showed how to correct the technical flaw of Algorithm 2.2 by performing feasible constraint projections, when necessary, to ensure that the triangulation remains well behaved, with bounded circumradii, as new datapoints are added.
- Algorithm 2.4 showed how to use an estimate for the lower bound of the function to maximally accelerate local refinement while still ensuring convergence to the global minimum.
- Algorithm 2.5 showed how to efficiently parallelize the function evaluations on n_p processors at each step of the algorithm.

Four adjustable parameters were identified in the above algorithms, and their effects quantified in numerical tests:

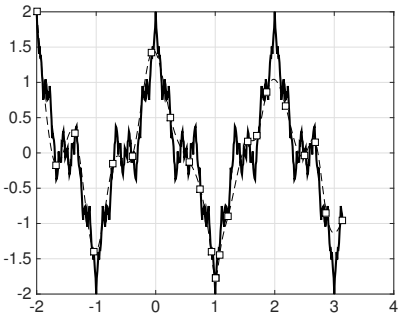
- Algorithm 2.2 introduced a tuning parameter $K > 0$, which governs the balance between global exploration and local refinement. For sufficiently large K applied to smooth functions (that is, Lipschitz, twice-differentiable, and bounded Hessian), convergence to a neighborhood of the global minimum is guaranteed in a finite number of iterations. For larger values of K, exploration becomes essentially uniformly



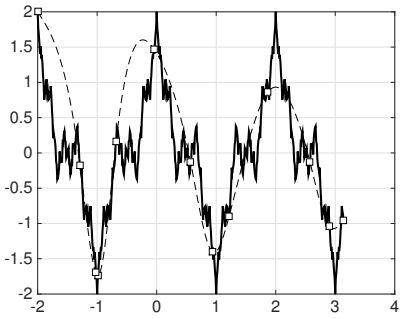
a Parallel Algorithm with $c = 0$, $n_p = 3$.



b Parallel Algorithm with $c = 0.5$, $n_p = 3$.



c Parallel Algorithm with $c = 1$, $n_p = 3$.



d Serial Algorithm

Figure 2.20: Parallel Δ-DOGS. Comparison of Algorithm 2.5, for various values of c, and Algorithm 2.2, all with $K = 15$, on the Weierstrass function considered previously. Evaluated points (squares), function of interest (solid line), interpolation at last step (dashed line)

.

over L.

- Algorithm 2.3 introduced a tuning parameter $r > 1$ which controls how frequently feasible constraint projections are performed. Small values of r leads to function evaluations accumulating on the boundaries of L, and a more uniform triangulation with reduced maximum circumradius. Large values of r leads to fewer function evaluations on the boundaries of L, and less uniform triangulations. Intermediate values of r are thus preferred.
- Algorithm 2.4 uses a tuning parameter y_0, which is an estimate for the global minimum. Convergence of Algorithm 2.4 was found to be remarkably rapid when y_0 was

an accurate estimate of the global minimum, both for smooth functions, and even certain nonsmooth functions, like Weierstrass, characterized by exploitable trends of the global shape of function. (For problems without such exploitable trends, the algorithm was well behaved, exploring essentially uniformly over the feasible domain.) When y_0 is less than the global minimum, convergence of Algorithm 2.4 to a global minimizer is guaranteed, though more global exploration is typically performed in the process. When y_0 is greater than the global minimum, convergence of Algorithm 2.4 to a value less than or equal to y_0 is guaranteed, with some local refinement performed thereafter.

- Algorithm 2.5 uses a tuning parameter c which controls how much closer evaluation points are allowed to get during the parallel substeps of the iteration. Global convergence is guaranteed for $0 < c \leq 1$. Small values of c allow function evaluations to get dense far from the global minimum during the parallel substeps, and slows convergence. Large values of c force the algorithm to focus primarily on global exploration, again slowing convergence. Intermediate values of c are thus preferred.

The algorithms described above were tested in Matlab, and Python and C++ implementations of these algorithms are currently being developed; for more information regarding availability of these codes, please contact the authors via email. Of course, as with any derivative-free optimization algorithm, there is a curse of dimensionality, and optimization in only moderate dimensional problems (i.e., $n < 10$) is expected to be numerically tractable; a key bottleneck of the present code as the dimension of the problem increases is the overhead required with the enumeration of the triangulation. The parallel version of the algorithm is expected to be particularly efficient in cases requiring substantial global exploration; in cases focusing primarily on local refinement, the speed up provided

by performing function evaluations in parallel is anticipated to be reduced.

In next chapter, we extend the algorithms developed here to convex domains bounded by arbitrary convex constraints. In Part 3 of this work, we extend the algorithms developed here to approximate function evaluations, each of which is obtained via statistical averaging over a finite number of samples.

Acknowledgements

This chapter is published in Journal of Global Optimization in the following article: P. Beyhaghi, D. Cavaglieri, T.R. Bewley, “Delaunay-based derivative-free optimization via global surrogates, part I: linear constraints", *Journal of Global Optimization*, (2015): 1-52. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Delaunay-based Optimization for convex domain: Δ-DOGS(C)

3.1 Introduction

In this chapter, a new derivative-free optimization algorithm is presented to minimize a (possibly nonconvex) function subject to convex constraints[1] on a bounded feasible region in parameter space:

$$\text{minimize } f(x) \text{ with } \; x \in L = \{x | c_i(x) \leq 0, \forall i = \{1, 2, \ldots, m\}\}, \tag{3.1}$$

where $c_i(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ are assumed to be convex, and both $f(x)$ and the $c_i(x)$ assumed to be twice differentiable. Moreover, the feasible domain L is assumed to be bounded with a nonempty interior (note that this assumption is only technical: if a given feasible domain

[1]The representation of a convex feasible domain as stated in (3.1) is the standard form used, e.g., in [19], but is not completely general. Certain convex constraints of interest, such as those implied by linear matrix inequalities (LMIs), can not be represented in this form. The extension of the present algorithm to feasible domains bounded by LMIs will be considered in future work.

has an empty interior, relaxing these constraints by ϵ generates a feasible domain with a nonempty interior; further related discussion on this matter is deferred to the last paragraph of §3.5.2.).

The algorithms developed in Chapter I were restricted to problems with linear constraints, as the domain searched was limited to the convex hull of the initial datapoints, which in Part I was taken as all vertices of the (there, polyhedral) feasible domain. Another potential drawback of the approach taken in Part I was the expense of the initialization of the algorithm: 2^n initial function evaluations were needed in the case of box constraints, and many more initial function evaluations were needed when there were many linear constraints. This chapter addresses both of these issues.

Constrained optimization problems have been widely considered with local optimization algorithms in both the derivative-based and the derivative-free settings. For global optimization algorithms, the precise nature of the constraints on the feasible region of parameter space is a topic that has received significantly less attention, as many global optimization methods (see for e.g., [12, 38, 39, 8, 40, 11 ,41, 42]) have very similar implementations in problems with linear and nonlinear constraints.

There are three classes of approaches for Nonlinear Inequality Problems (NIPs) using local derivative-based methods. Those in the first class, called sequential quadratic programming methods (see [43, 44]), impose (and, successively, update) a local quadratic model of the objective function $f(x)$ and a local linear model of the constraints $c_i(x)$ in order to estimate the local optimal solution at each step. These models are defined based on the local gradient and Hessian of the objective function $f(x)$, and the local Jacobian of the constraints $c_i(x)$, at the datapoint considered at each step. Those in the second class, called quadratic penalty methods (see [45, 46, 47]), perform some function evaluations outside

of the feasible domain, with a quadratic term added to the cost function which penalizes violations of the feasible domain boundary, and solves a sequence of subproblems with successively stronger penalization terms in order to ultimately solve the problem of interest. Those in the third class, called interior point methods (see [43]), perform all function evaluations inside the feasible domain, with a log barrier term added to the cost function which penalizes proximity to the feasible domain boundary (the added term goes to infinity at the domain boundary), and solves a sequence of subproblems with successively weaker penalization terms in order to ultimately solve the problem of interest.

NIPs are a subject of significant interest in the derivative-free setting as well. One class of derivative-free optimization methods for NIPs is called direct methods, which includes the well-known General Pattern Search (GPS) [48] methods which restrict all function evaluations to lie on an underlying grid which is successively refined. GPS methods were initially designed for unconstrained problems, but have been modified to address box-constrained problems [49], linearly-constrainted problems, and smooth nonlinearly-constrained problems [50]. Mesh Adaptive Direct Search (MADS) algorithms [51, 52, 53, 54] are modified GPS algorithms that handle non-smooth constraints. GPS and MADS algorithms have been extended in [13] to handle coordination with grids (that is, non-Cartesian grids) given by n-dimensional sphere packings, which significantly improves efficiency in high dimensional problems.

The leading class of derivative-free optimization algorithms today is known as Response Surface Methods. Methods of this class leverage an underlying inexpensive model, or "surrogate", of the cost function. Kriging interpolation is often used to develop this surrogate [12]; this convenient choice provides both an interpolant and a model of the uncertainty of this interpolant, and can easily handle extrapolation from the convex hull of the

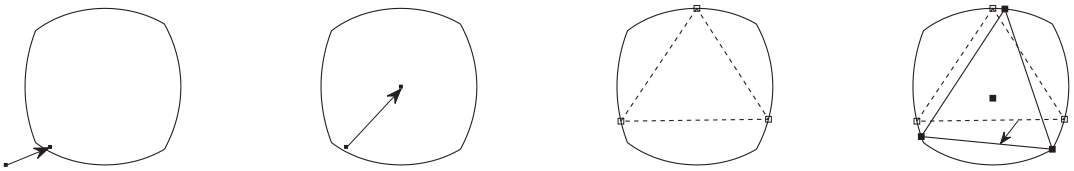
data points out to the (curved) boundaries of a feasible domain bounded by nonlinear constraints. Chapter 2 summarized some of the numerical issues associated with the Kriging interpolation method, and developed a new Response Surface Method based on *any* well-behaved interpolation method, such as polyharmonic spline interpolation, together with a synthetic model of the uncertainty of the interpolant built upon a framework provided by a Delaunay triangulation.

Unfortunately, the uncertainty function used in chapter 2 of this study is only defined within the convex hull of the available datapoints, so the algorithms described in Part I do not extend immediately to more general problems with convex constraints. The present chapter develops the additional machinery necessary to make this extension effectively, by appropriately increasing the domain which is covered by convex hull of the datapoints as the algorithm proceeds. As in chapter 2, we consider optimization problems with expensive cost function evaluations but computationally inexpensive constraint function evaluations; we further assume that the computation of the surrogate function has a low computational cost. The algorithm developed in chapter 3 of this study has two significant advantages over those developed in chapter 2: (a) it solves a wider range of optimization problems (with more general constraints), and (b) the number of initial function evaluations is reduced (this is significant in relatively high-dimensional problems, and in problems with many constraints).

The chapter is structured as follows. Section 3.2 discusses briefly how the present algorithm is initialized. Section 3.3 presents the algorithm. Section 3.4 analyzes the convergence properties of the algorithm. In Section 3.5, the optimization algorithm proposed is applied to a number of test functions with various different constraints in order to quantify its behavior. Conclusions are presented in Section 3.6.

3.2 Initialization

In contrast to the algorithms developed in Part I, the algorithm developed here is initialized by performing initial function evaluations at only $n + 1$ feasible points. There is no need to cover the entire feasible domain by the convex hull of the initial datapoints; however, it is more efficient to initialize with a set of $n + 1$ points whose convex hull has the maximum possible volume.



a Step 1. b Step 2. c Step 3. d Steps 4-6.

Figure 3.1: Representation of Algorithm 3.1 on an illustrative example.

Before calculating the initial datapoints to be used, a feasible point x_f which satisfies all constraints must be identified. The feasible domain considered [see (3.1)] is $L = \{x|c_i(x) \leq 0, 1 \leq i \leq m\}$, where the $c_i(x)$ are convex. The convex feasibility problem (that is, finding a feasible point in a convex domain) is a well-known problem in convex optimization; a number of effective methods are presented in [45, 55]. In this chapter, the convex feasibility problem is solved by minimizing the following quadratic penalty function

$$P_1(x) = \sum_{i=1}^{m} \max(c_i(x), 0)^2, \tag{3.2}$$

where $P_1(x)$ is simply the quadratic penalty function used by quadratic penalty methods for solving NIPs. Since the $c_i(x)$ are convex, $P_1(x)$ is also convex; thus, if the feasible domain is nonempty, any local minimizer of $P_1(x)$ is feasible. Note that the feasible domain is empty if the minimum of $P_1(x)$ is greater than 0.

Note that $P_1(x)$ is minimized via an iterative process which uses the Newton direction at each step, together with a line search, in order to guaranty the Armijo condition. Hessian modification [28] may be required to find the Newton's direction, since $P_1(x)$ is not strictly convex. An example of this procedure to find a feasible point from an initial infeasible point is illustrated in Figure 3.1a.

The point x_f generated above is not necessarily an interior point of L. Note that the interior of L is nonempty if and only if the MFCQ (Mangasarian-Fromovitz constraint qualification) holds at x_f (see Proposition 3.2.7 in [56]). Checking the MFCQ at point x_f is equivalent to solving a linear programming problem [57], which either (a) generates a direction towards the interior of L from x_f (from which a point x_T on the interior of L is easily generated), or (b) establishes that the interior is empty. The optimization algorithm developed in this chapter is valid only in case (a).

Starting from this interior feasible point x_T, the next step in the initialization identifies another feasible point that is, in a sense, far from all the boundaries of feasibility. This is achieved by minimizing the following logarithmic barrier function:

$$P_2(x) = -\sum_{i=1}^{m} \log(-c_i(x)). \tag{3.3}$$

It is easy to verify that $P_2(x)$ is convex, and has a unique global minimum. Note that since initial point is an interior point, $P_2(x)$ can be defined at it. This function is also minimized via Newton's method; the line search at each step of this minimization is confined to the interior of the feasible domain. The outcome of this procedure, denoted X_0, is illustrated in Figure 3.1b.

After finding the interior point X_0 from the above procedure, a regular simplex[2] Δ

[2]A simplex is said to be regular if all of its edges are equal.

whose body center is X_0 is constructed. Finding the coordinates of a regular simplex is well-known problem in computational geometry (see, e.g., §8.7 of [58]). The computational cost of finding a regular simplex is $O(n^2)$, which is insignificant compared with the rest of the algorithm.

Before continuing the initialization process, a new concept is introduced which is used a few times in this chapter.

Definition 10. *The* prolongation *of point M from a feasible point O onto the feasibility boundary ∂L, is the unique point on the ray from O that passes through M which is on the boundary of feasibility. In order to find the prolongation of M from O on L, first the following 1D convex programming problem has to be solved.*

$$\begin{aligned} &\max_{\alpha \in \mathbb{R}} \quad \alpha, \\ &\textit{subject to} \quad c_i(O + \alpha\,(M - O)) \leq 0, \quad \forall\, 1 \leq i \leq m. \end{aligned} \tag{3.4}$$

Then, $N = O + \alpha\,(M - O)$, the prolongation point. It is easy to observe that N is located on the boundary of L. Since $\alpha = 0$ is a feasible point for (3.2)*, it can be solved using the interior point method [43]. The computational cost of this subproblem is not significant if the computational cost of the constrained functions $c_i(x)$ are negligible.*

Based on the above definition, the initialization process is continued by prolongation of the vertices of the regular simplex from X_0 onto L. As illustrated in Figure 3.1c, the simplex so generated by this prolongation has a relatively large volume, and all of its vertices are feasible. The algorithm developed in the remainder of this chapter incrementally increases the convex hull of the available datapoints towards the edges of the feasible domain itself. This process is generally accelerated if the volume of the initial feasible

simple is maximized.

The feasible simplex generated above is not the feasible simplex of maximal volume. As illustrated in Figure 3.1d, we next perform a simple iterative adjustment to the feasible vertices of this simplex to locally maximize its volume[3]. That is, denoting $\not{V}\{\Delta\{V_1, V_2, \ldots, V_{m+1}\}\}$ as the volume of an m-dimensional simplex with corners $\{V_1, V_2, \ldots, V_{m+1}\}$, we consider the following problem:

$$\begin{aligned} &\text{maximize } \not{V}\{\Delta\{V_1, V_2, \ldots, V_{n+1}\}\}, \\ &\text{where} \{V_1, V_2, \ldots, V_{n+1}\} \in L. \end{aligned} \tag{3.5}$$

The problem of finding $p > 2$ points in a convex subset of $\mathbb{R}^2$ which maximizes its enclosing area is a well-known problem in the fields of interpolation, data compression, and robotic sensor networks. An efficient algorithm to solve this problem is presented in [59]. Note that (3.5) differs from the problem considered in [59] in three primary ways:

- (3.5) is defined in higher dimensions ($n \geq 2$);
- the boundary of the domain L is possibly nondifferenible, whereas the problem in [59] assumed the boundary is twice differentiable;
- (3.5) is easier in the sense that a simplex is considered, not a general convex polyhedron.

As a result of these differences, a different strategy is applied to the present problem, as described below.

Consider $V_1, V_2, \ldots, V_{n+1}$ as the vertices of a simplex Δ_x. The volume of Δ_x may be

[3]The problem of globally maximizing the volume of a feasible simplex inside a convex domain is, in general, a nonconvex problem. We do not attempt to solve this global maximization, which is unnecessary in our algorithm.

written:

$$\mathcal{V}(\Delta_x) = \frac{\mathcal{V}(\Delta'_x)L_{V_k}}{n}, \tag{3.6}$$

where Δ'_x is the $n-1$ dimensional simplex generated by all vertices except V_k, H_k is the $n-1$ dimensional hyperplane containing Δ'_x, and L_{V_k} is the perpendicular distance from the vertex V_k to the hyperplane H_k.

By (3.6), L_{V_k} must be maximized to maximize the volume of the simplex if the other vertices are fixed. Furthermore, it is easily verified that the perpendicular distance of a point p to the hyperplane H_k, characterized by $a_k^T x = b_k$, is equal to $|(a_k^T p - b_k)/(a_k^T a_k)|$; thus,

$$V_k = \underset{p \in L}{\operatorname{argmax}} |a_k^T p - b_k|. \tag{3.7}$$

Solving the optimization problem (3.7) is equivalent to finding the maximum of two convex optimization problems with linear objective functions. The method used to solve these two convex problems is the primal-dual-barrier method explained in detail in [60] and [43].

Based on the tools developed above, (3.5) is solved via the following procedure:

Algorithm 3.1 Completion of initialization of Δ-DOGS(C)

Find a point X in $L = \{x | c_i(x) \leq 0, 1 \leq i \leq m\}$ by minimizing $P_1(x)$ defined in (3.2); then, goes to the interior of L by checking the MFCQ condition.
Starting from X; then, goes to another feasible point X_0 which is far from the constraint boundaries by minimizing $P_2(x)$ defined in (3.3).
Generate a uniform simplex with body center X_0; denote the vertices of this simplex $X_1, X_2, \ldots, X_{n+1}$.
Determine $V_1, V_2, \ldots, V_{n+1}$ as the prolongations of $X_1, X_2, \ldots, X_{n+1}$ from X_0 to the boundary of L.
For $k = 1$ to $n+1$, modify V_k to maximize the distance from the hyperplane which passes through the other vertices by solving (3.7).
If all modification at step 6 are small, stop the algorithm; otherwise repeat from 5.

Definition 11. *The simplex Δ_i obtained via Algorithm 3.1, is referred to the* initial simplex.

Definition 12. *For each vertex V_k of the initial simplex, the hyperplane H_k, characterized by $a_k^T x = b_k$, passes through all vertices of the initial simplex except V_k. Without loss of generality, the sign of the vector a_k may be chosen such that, for any point $x \in L$, $a_k^T x \leq a_k V_k$. Now define the enclosing simplex Δ_e as follows:*

$$\Delta_e = \{x \,|\, a_i^T x \leq a_i^T V_i,\ 1 \leq i \leq n+1\};$$

note that the feasible domain L is a subset of the enclosing simplex Δ_e.

Lemma 12. *Consider $V_1, V_2, \ldots, V_{n+1}$ as the vertices of the initial simplex Δ_i, and $P_1, P_2, \ldots, P_{n+1}$, as those of the enclosing simplex Δ_e given in Definition 12; then*

$$p^k = \sum_{j=1}^{n+1} V_j - nV_k. \tag{3.8}$$

Proof. Define p^k according to (3.8). Then, for all $i \neq k$,

$$a_i^T p^k = \sum_{j=1}^{n+1} a_i^T V_j - n a_i^T V_k.$$

According to Definition 12, $a_i^T V_j = b_i$, $\forall\, j \neq i$; thus, above equation is simplified to:

$$a_i^T p^k = a_i^T V_i.$$

Thus, n of the independent constraints on the enclosing simplex given in Definition 12 are binding at p^k, and thus p^k is a vertex of Δ_e. □

Definition 13. *Consider $P_1, P_2, \ldots, P_{n+1}$ as the vertices of the enclosing simplex Δ_e, and O*

as its body center. The vertices of the exterior simplex Δ_E are defined as follows:

$$E_i = O + \kappa(P_i - O) \tag{3.9}$$

where $\kappa > 1$ is called the extending parameter.

The relative positions of the initial, enclosing, and exterior simplices are illustrated in Figure 3.2. It follows from (3.8) and (3.9) that $\sum V_i = \sum P_i = \sum E_i$; that is, the initial, enclosing, and exterior simplices all have the same body center, denoted O.

Remark 14. *In this chapter, the following condition is imposed on the extending parameter κ:*

$$\kappa \geq \frac{2 \max_{y \in L} \|y - O\|}{n \min_{1 \leq i \leq n+1} \|V_i - O\|}, \tag{3.10}$$

where O is the body center of the initial simplex, and $\{V_1, V_2, \ldots, V_{n+1}\}$ are the vertices of the initial simplex. In §3.4, it is seen that this condition is necessary to ensure convergence of the optimization algorithm presented in §3.3. In general, the value of $\max_{y \in L}\{\|y - O\|\}$ in not known and is difficult to obtain[4]; however, the following upper bound is known,

$$\max_{y \in L} \|y - O\| \leq n \max_{1 \leq i \leq n+1} \|V_i - O\|.$$

Thus, by choosing κ as follows, (3.10) *is satisfied:*

$$\kappa = \frac{2 \max_{1 \leq i \leq n+1} \|V_i - O\|}{\min_{1 \leq i \leq n+1} \|V_i - O\|}. \tag{3.11}$$

Definition 14. *The vertices of the initial simplex Δ_i, together with its body center O, form*

[4]This is a maximization problem for a convex function in a convex compact domain. This problem is studied in [61].

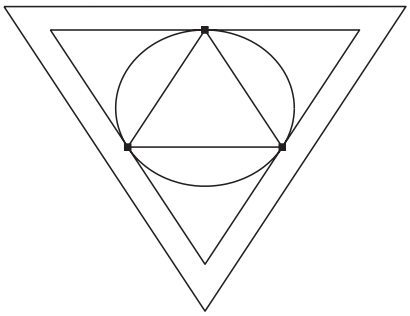


Figure 3.2: Illustration of the initial, enclosing and exterior simplices for an elliptical feasible domain.

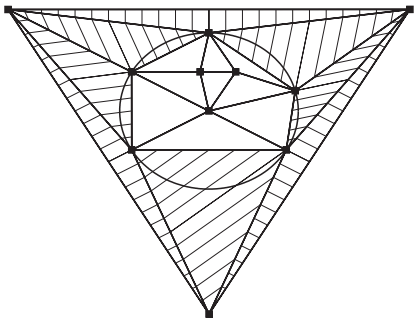


Figure 3.3: Representation of the boundary (hashed) and interior (non-hashed) simplices in the Delaunay triangulation of a set of feasible evaluation points together with the three vertices of the exterior simplex.

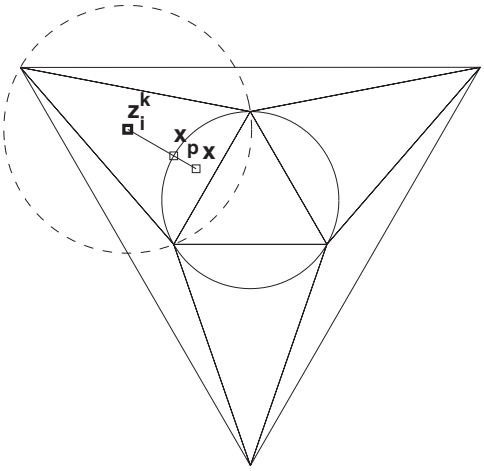


Figure 3.4: Illustration of a convex boundary projection: the solid circle indicates the feasible domain, x is the initial point, z_i^k is the circumcenter of a boundary simplex Δ_i^k containing x, the dashed circle indicates the corresponding circumsphere, and x_p is the feasible boundary projection.

the initial evaluation set S_E^0. The union of S_E^0 and the vertices of the exterior simplex form the initial triangulation set S_T^0.

Remark 15. *After constructing S_E^0 and S_T^0, a Delaunay triangulation Δ^0 over S_T^0 is calculated. If the body center O and a vertex E of the exterior simplex Δ_E are both located in any simplex of the triangulation Δ^0; then E′ is defined as the intersection of segment OE with the boundary of L, and E′ is added to both S_E^0 and S_T^0, and the triangulation is updated. After at most $n+1$ modifications of this sort, the body center O is not located in the same simplex of the triangulation as any of the vertices of the exterior simplex. As a result, the number of points in S_E^0 and S_T^0 are at most $2n+3$, and $3n+4$ respectively. The sets S_E^0 and S_T^0 are used to initialize the optimization algorithm presented in the following section.*

3.3 Description of the Optimization Algorithm

In this section, we present an algorithm to solve the optimization problem defined in (3.1). We assume that calculation of the constraint functions $c_i(x)$ are computationally inexpensive compared to the function evaluations $f(x)$, and that the gradient and Hessian of $c_i(x)$ are available.

Before presenting the optimization algorithm itself, some preliminary concepts are first defined.

Definition 15. *Consider Δ^k as a Delaunay triangulation of a set of points S_T^k, which includes the vertices of the exterior simplex Δ_E. As illustrated in Figure 3.3, there are two type of simplices:*

a. boundary *simplices, which include at least one of the vertices of the exterior simplex Δ_E, and*

b. interior *simplices, which do not include any vertices of the exterior simplex* Δ_E.

Definition 16. *For any boundary simplex* $\Delta_x \in \Delta^k$ *which includes only one vertex of the exterior simplex* Δ_E*, the* $n-1$ *dimensional simplex formed by those vertices of* Δ_x *of which are not in common with the exterior simplex is called a* face *of* Δ_x.

Definition 17. *For each point* $x \in L$*, the constraint function* $g(x)$ *is defined as follows:*

$$g(x) = \max_{1 \le j \le m} \{c_j(x)\}. \tag{3.12}$$

For each point x in a face $F \in \Delta^k$*, the linearized constraint function with respect to the face* F*,* $g_L^F(x)$*, is defined as follows:*

$$c_{j,L}^F(x) = \sum_{i=1}^{n} w_i c_j(V_i), \tag{3.13a}$$

$$g_L^F(x) = \max_{1 \le j \le m} \{c_{j,L}^F(x)\}, \tag{3.13b}$$

where the weights $w_i \ge 0$ *are defined such that*

$$x = \sum_{i=1}^{r} w_i V_i \quad with \quad \sum_{i=1}^{r} w_i = 1.$$

Definition 18. *Consider x as a point which is located in the circumsphere of a boundary simplex in* Δ^k*. Identify* Δ_i^k *as that boundary simplex of* Δ^k *which includes x in its circumesphere and which maximizes* $g(z_i^k)$*. The prolongation (see Footnote 10) of* z_i^k *from x onto the boundary of feasibility is called a* feasible boundary projection *of the point x, as illustrated in Figure 3.4.*

Algorithm 3.2 Δ-DOGS(C)

As described in previous section, prepare the problem for optimization by (a) executing Algorithm 3.1 to find the initial simplex Δ_i, (b) identifing the exterior simplex Δ_E as described in Definition 13.
Take S_E^0 and S_T^0 as the initial evaluation set and the initial triangulation set, respectively. Evaluate $f(x)$ at all points in S_E^0. Set $k = 0$.
Calculate (or, for $k > 0$, update) an interpolating function $p^k(x)$ that passes through all points in S_E^k.
Perform (or, for $k > 0$, incrementally update) a Delaunay triangulation Δ^k over all points in S_T^k.
For each simplex Δ_i^k of the triangulation Δ^k which includes at most one vertex of the exterior simplex, define F_i^k as the corresponding face of Δ^k if Δ_i^k is a boundary simplex, or take $F_i^k = \Delta_i^k$ itself otherwise. Then:

a. Calculate the circumcenter z_i^k and the circumradius r_i^k of the simplex F_i^k.

b. Define the **local uncertainty function** $e_i^k(x)$ as

$$e_i^k(x) = (r_i^k)^2 - (x - z_i^k)^T(x - z_i^k). \tag{3.14}$$

c. Define the **local search function** $s_i^k(x)$ as follows: if Δ_k^i is an interior simplex, take

$$s_i^k(x) = \frac{p^k(x) - y_0}{e_i^k(x)}; \tag{3.15}$$

otherwise, take

$$s_i^k(x) = \frac{p^k(x) - y_0}{-g_L^{F_i^k}(x)}, \tag{3.16}$$

where y_0 is an estimate (provided by the user) for the value of the global minimum.

d. Minimize the local search function $s_i^k(x)$ in F_i^k.

If a point x at step 3 is found in which $s^k(x) \leq y_0$, then redefine the search $s_i^k(x) = p^k(x)$ in all simplices, and take x_k as the minimizer of this search function in L; otherwise, take x_k as the minimizer of the local minima identified in step 3d.
If x_k is not inside the circumsphere of any of the boundary simplices, define $x'_k = x_k$; otherwise, define x'_k as the feasible boundary projection (see Definition 18) of x_k. Perform a function evaluation at x'_k, and take $S_E^{k+1} = S_E^k \cup \{x'_k\}$ and $S_T^{k+1} = S_T^k \cup \{x'_k\}$.
Repeat from step 3 until $\min_{x \in S^k}\{\|x'_k - x\|\} \leq \delta_{des}$.

Remark 16. *Algorithm 2.2 generalizes the adaptive K algorithm developed in chapter 2 of this work to convex domains. We could similarly extend the constant K algorithm developed in chapter 2 by modifying* (3.15) *and* (3.16) *as follows: if Δ_k^i is an interior simplex, take*

$$s_i^k(x) = p^k(x) - K\, e_i^k(x);$$

otherwise, take

$$s_i^k(x) = p^k(x) + K\, g_L^{F_i^k}.$$

Such a modification might be appropriate if an accurate estimate of the Hessian of $f(x)$ is available, but an accurate estimate of the lower bound y_0 of $f(x)$ is not.

Remark 17. *At each step of Algorithm 2.2, a feasible point x'_k is added to L. If Algorithm 3.2 is not terminated at finite k, since the feasible domain L is compact, the sequence of x'_k will have at least one convergent subsequence, by the Bolzano Weierstrass theorem. Since this subsequence is convergent, it is also Cauchy; therefore, for any $\delta_{des} > 0$, there are two integers k and $m < k$ such that $\|x'_m - x'_k\| \leq \delta_{des}$. Thus, $\min_{x \in S^k}\{\|x'_k - x\|\} \leq \delta_{des}$; thus, the termination condition will be satisfied at step k. As a result, Algorithm 3.2 will terminate in a finite number of iterations.*

Definition 19. *At each step of Algorithm 3.2, and for each point $x \in L$, there is a simplex $\Delta_i^k \in \Delta^k$ which includes x. The global uncertainty function $e^k(x)$ is defined as $e^k(x) = e_i^k(x)$. It is shown in Part I (Lemmas* 3 *and* 4*) that $e^k(x)$ is continuous and Lipchitz with Lipchitz constant of r_{max}^k, the maximum circumradius of Δ^k, and that the Hessian of $e^k(x)$ inside each simplex, and over each face F of each simplex, is $-2\,I$.*

There are three principle difference between Algorithm 3.2 above and the corresponding algorithm proposed in chapter 2 for the linearly-constrained problem:

- In the initialization, instead of calculating the objective function at all of the vertices of feasible domain (which is not possible if the constraints are nonlinear), the present algorithm is instead initialized with between $n + 2$ and $2n + 3$ function evaluations.
- The local search function is modified in the boundary simplices.
- The feasible *boundary* projections used in the present work are analogous to (but slightly different from) Algorithm 3 of chapter 2, which applies one or more feasible *constraint* projections.

3.3.1 Minimizing the search function

As with Δ-DOGS, the most expensive part of Algorithm 3.2, separate from the function evaluations themselves, is step 3 of the algorithm. The cost of this step is proportional to the total number of simplices S in the Delaunay triangulation. As derived in [25], a worst-case upper bound for the number of simplices in a Delaunay triangulation is $S \sim O(N^{\frac{n}{2}})$, where N is the number of vertices and n is the dimension of the problem. As shown in [26, 27], for vertices with a uniform random distribution, the number of simplices is $S \sim O(N)$. This fact limits the present algorithm to be applicable only for relatively low DOF problems (say, n<10). In practice, the most limiting part of step 3 is the memory requirement imposed by the computation of the Delaunay triangulations. Thus, the present algorithm itself is applicable only to those problems for which Delaunay triangulations can be performed amongst the datapoints.

In this section, we describe some details and facts that simplifies certain other aspects of step 3 (besides the Delaunay triangulations). In general, there are two type of simplices in Δ^k whose definition of the search function is different.

In the interior simplices Δ_i^k, the function $s_i^k(x) = \frac{p^k(x)-y_0}{e_i^k(x)}$ has to be minimized in Δ_i^k.

One important property of $e^k(x)$ which makes this problem easier is $e^k(x) = \max_{j\in S} e^k_j(x)$ (Lemma 2), and as it shown in Lemma 5, if $s^k(x)$ is minimized in L rather than Δ^k_i, the position of point x_k would not be changed. Another important issue is the method of initialization for the minimization of $s^k_i(x)$. To accomplish this, we choose a point $\hat{x}^k_i$ which maximizes $e^k_i(x)$ as the initial point for the minimization algorithm. Note that, if the circumcenter of this simplex is included in this simplex, the circumcenter is the point that maximizes $e^k_i(x)$; otherwise, this maximization is a simple quadratic programming problem. The computational cost of calculating $\hat{x}^k_i$ is similar to that of calculating x^k_c. After the initializations described above, the subsequent minimizations are performed using Newton's method with Hessian modification via modified Cholesky factorization (see [28]) to minimize $s^k_i(x)$ within L. It will be shown (see Theorem 7) that performing Newton's method is actually not necessary to guaranty the convergence of Algorithm 3.2, and having a point whose value $s^k(x)$ is less than the initial points is sufficient to guaranty the convergence; however, performing Newton minimizations at this step can improve the speed of convergence. In practice, we will perform Newton minimizations only in a few simplices whose initial points have small values for $s^k(x)$.

For the boundary simplices Δ^k_i, the function $s^k_i(x)$ is defined only over the face F^k_i of Δ^k_i. The minimization in each boundary simplex is initialized at the point $\tilde{x}^k_i$ which maximizes the piecewise linear function $-g_L^{F^k_i}(x)$; this maximization may be written as a linear programming problem with the method described in §1.3, p. 7, in [62]. Similarly, it can be seen that initialization with these points is enough to guaranty the convergence; thus, it is not necessary to perform the Newton method after this initialization. In our numerical implementation, the best of these initial points are considered for the simulation; however, the implementation of Newton minimizations on these faces could further improve the

speed of convergence.

Remark 18. *As in Algorithm 4 of chapter 2, since Newton's method doesn't always converge to a global minimum, x_k is not necessarily a global minimizer of $s^k(x)$. However, the following properties are guaranteed:*

$$\text{if } s^k(x) = p^k(x), \text{ then } p^k(x_k) \leq y_0; \tag{3.17a}$$

$$\text{otherwise } s^k(x_k) \leq s_j^k(\hat{x}_j^k) \ \ \forall \Delta_j^k \in \Delta^k, \tag{3.17b}$$

$$\text{and } s^k(x_k) \leq s_j^k(\tilde{x}_j^k) \ \ \forall F_j^k \in \Delta^k. \tag{3.17c}$$

Recall that $\hat{x}_j^k$ is the maximizer of $e_j^k(x)$ in the interior simplex $\Delta_j^k \in \Delta^k$, and $\tilde{x}_j^k$ is the maximizer of $-g_L^F(x)$ over the face F_j^k of the boundary simplex $\Delta_j^k \in \Delta^k$. These properties are all that are required by Theorem 7 in order to establish convergence.

3.4 Convergence analysis

In this section, the convergence properties of Algorithm 3.2 are analyzed. The structure of the convergence analysis is similar to that in chapter 2.

In this section, the following conditions for the function of interest, $f(x)$, the constraints $c_i(x)$, and the interpolating functions $p^k(x)$ are imposed.

Assumption 1. *The interpolating functions $p^k(x)$, for all k, are Lipchitz with the same Lipchitz constant L_p.*

Assumption 2. *The function $f(x)$ is Lipchitz with Lipchitz constant L_f.*

Assumption 3. *The individual constraint functions $c_i(x)$ for $1 \leq i \leq m$ are Lipchitz with Lipchitz constant L_c.*

Assumption 4. *A constant K_{pf} exists in which*

$$K_{pf} > \lambda_{max}(\nabla^2(f(x) - p^k(x))/2), \quad \forall x \in L \, and \, k > 0.$$

Assumption 5. *A constant K_f exists in which*

$$K_f > \lambda_{max}(\nabla^2(f(x))/2), \; \forall x \in L.$$

Assumption 6. *A constant K_g exists in which*

$$K_g > \lambda_{max}(\nabla^2(c_i(x))/2), \; \forall x \in L.$$

Before analyzing the convergence of Algorithm 3.2, some preliminary definitions and lemmas are needed.

Definition 20. *According to the construction of the initial simplex, its body center is an interior point in L; thus, noting* (3.12)*, $g(O) < 0$. Thus, the quantity $L_O = \max_{y \in L} \|y - O\|/(-g(O))$ is a bounded positive real number.*

Lemma 13. *Each vertex of the boundary simplices of Δ^k, for all steps of Algorithm 3.2, is either a vertex of the exterior simplex or is located on a boundary of L.*

Proof. We will prove this lemma by induction on k. By construction, O is not in any boundary simplex of Δ^0, and all other points of S_E^k are on the boundary of L; therefore, the lemma is true for the base case $k = 0$. Assuming the lemma is true for the case $k - 1$, we now show that it follows that the lemma is also true for the case k.

At step k of Algorithm 3.2, we add a point x'_k to the triangulation set S_T^k. As described step 5 of Algorithm 3.2, this point arises from one of two possible cases.

In the first case, x_k is not located in the circumsphere of any boundary simplex, and a feasible boundary projection is not performed; as a result, $x'_k = x_k$ is an interior point of L. In this case, the incremental update of the Delaunay triangulation at step k does not change the boundary simplices of Δ^{k-1} (see, e.g., section 2.1 in [23]), and thus the lemma is true in this case.

In the other case, x_k is located in the circumsphere of a boundary simplex, and a feasible boundary projection is performed; thus, x'_k is on the boundary of L. Consider Δ_x as one of the new boundary simplices which is generated at step k. By construction, x_k is a vertex of Δ_x. Define F as the $n-1$ dimensional face of Δ_x which does not include x_k. Based on the incremental construction of the Delaunay triangulation, F is a face of another simplex Δ'_x at step $k-1$. Since Δ_x is a boundary simplex, F includes a vertex of the exterior simplex; thus, Δ'_x is a boundary simplex in Δ^{k-1}; thus, each vertex of F is either a boundary point or a vertex of the exterior simplex; moreover, x'_k is a boundary point. As a result, Δ_x satisfies the lemma. For those boundary simplices which are not new, the lemma is also true, by the induction hypothesis. Thus, the lemma is also true in this case. □

Definition 21. *For any point $x \in L$, the* interior projection *of x, donated by x_I, is defined as follows: If x is located inside or on the boundary of an interior simplex, put $x_I = x$; otherwise, x_I is taken as the point of intersection, closest to x, of the line segment Ox with the boundary of the union of the interior simplices. Thus, x_I is located on a face of the triangulation.*

Lemma 14. *Consider x as a point in L which is not located in the union of the interior simplices at step k of Algorithm 3.2. Define x_I as the interior projection of x (see Definition 21). Then,*

$$p^k(x_I) - f(x) \leq e^k(x_I)\{K_{pf} + L_f K_g L_O\} - L_f L_O g_L^F(x_I), \tag{3.18}$$

where $e^k(x)$ is the uncertainty function, $g_L^F(x)$ is the linearized constraint function (see Definition 17) with respect to a face F which includes x_I, and L_f, K_{pf}, K_g, and L_O are defined in Assumptions 10, 4 and 6 and Definition 20, respectively.

Proof. By Definition 21, x_I is a point on the line segment Ox; thus,

$$x_I = x\frac{\|O - x_I\|}{\|O - x\|} + O\frac{\|x - x_I\|}{\|O - x\|}.$$

Define $\{V_1, V_2, \ldots, V_n\}$ as the vertices of F, and $G_j^k(x) = c_{j,L}^F(x) - c_j(x) - K_g e^k(x)$, where $c_{j,L}^F$ is the linear function defined in (3.13a). First we will show that $G_j^k(x)$ is strictly convex in F. Since $c_{j,L}^F(x)$ is a linear function of x in F, and $\nabla^2 e^k(x) = -2\,I$ (see (3.14)), it follows that

$$\nabla^2 G_j^k(x) = -\nabla^2\{c_j(x)\} + 2\,K_g\,I. \tag{3.19}$$

According to Assumption 6 $\nabla^2 G_j^k(x) > 0$ in F; thus, it is strictly convex; therefore, its maximum in F is located at one of the vertices of F. Furthermore, by construction, $G_j^k(V_i) = 0$; thus, $G_j^F(x_I) \leq 0$, and

$$\begin{aligned} c_j(x_I) &\geq c_{j,L}^F(x_I) - K_g e^k(x_I) \;\; \forall \;\; 1 \leq j \leq m, \\ g(x_I) &\geq g_L^F(x_I) - K_g e^k(x_I), \end{aligned} \tag{3.20}$$

where $g(x)$ is defined in Definition 17. Since the $c_i(x)$ are convex, $g(x)$ is also convex; thus,

$$g(x_I) \leq g(x)\frac{\|O - x_1\|}{\|O - x\|} + g(O)\frac{\|x - x_I\|}{\|O - x\|}. \tag{3.21}$$

Since $x \in L$, $g(x) \leq 0$. Since O is in the interior of L, $g(O) < 0$; from (3.20) and (3.21), it

thus follows that

$$g_L^F(x_I) - K_g e^k(x_I) \leq g(O)\frac{\|x - x_I\|}{\|O - x\|},$$
$$\|x - x_I\| \leq \frac{\|x - O\|}{-g(O)}\{K_g e^k(x_I) - g_L^F(x_I)\}.$$

By Definition 20, this leads to

$$\|x - x_I\| \leq L_O\{K_g e^k(x_I) - g_L^F(x_I)\}. \tag{3.22}$$

Now define $T(x) = p^k(x) - f(x) - K_{pf} e^k(x)$. By Assumption 4 and Definition 19, similar to $G_j^k(x)$, $T(x)$ is also strictly convex in F, and $T(V_i) = 0$; thus,

$$p^k(x_I) - K_{pf} e^k(x_I) \leq f(x_I). \tag{3.23}$$

Furthermore, L_f is a Lipchitz constant for $f(x)$; thus,

$$f(x_I) - f(x) \leq L_f\|x - x_I\|. \tag{3.24}$$

Using (3.22),(3.23), and (3.24), (3.18) is satisfied. □

Remark 19. *If x is a point in an interior simplex, it is easy to show [as in the derivation of* (3.23)*] that* (3.18) *is modified to*

$$p^k(x) - f(x) \leq K_{pf} e^k(x). \tag{3.25}$$

Lemma 15. *Consider F as a face of the union of the interior simplices at step k of Algorithm 3.2, x as a point on F, and $g_F^L(x)$ as the linearized constraint function on F as defined*

in Definition (17). *Then,*

$$-g_F^L(x) \leq L_c \sqrt{e^k(x)}. \tag{3.26}$$

Proof. By Assumption 11, the $c_i(x)$ are Lipchitz with constant L_c. Thus, by (3.13a), the $c_{i,F}^L(x)$ are Lipchitz with the same constant. Furthermore, the maximum of a finite set of Lipchitz functions with constant L_c is Lipchitz with the same constant (see Lemma 2.1 in [63]). Define $\{V_1, V_2, \ldots, V_n\}$ as the vertices of F; then, by Lemma 13, V_i is on the boundary of L; thus, $g^L(V_i) = 0$ for all $1 \leq i \leq n$, and

$$-g_F^L(x) \leq L_c \min_{1 \leq i \leq n} \{\|x - V_i\|\}. \tag{3.27}$$

Now define $\{w_0, w_1, \ldots, w_n\}$ as the weights of x in F (that is, $x = \sum_{j=0}^n w_j V_j$ where $w_i \geq 0$ and $\sum_{j=0}^n w_j = 1$), z as the circumcenter of F, and r as the circumradius of F. Then, for each j,

$$r^2 = \|V_j - z\|^2 = \|V_j - x\|^2 + \|x - z\|^2 + 2\,(V_j - x)^T(x - z).$$

Multiplying the above equations by w_j and taking the sum over all j, noting that $\sum_{j=0}^n w_j = 1$, it follows that

$$r^2 = \sum_{j=0}^n w_j \|V_j - x\|^2 + \|x - z\|^2 + 2 \sum_{j=1}^n w_j (V_j - x)^T(x - z).$$

Since $\sum_{j=0}^{n} w_j(V_j - x) = 0$, this simplifies to

$$r^2 = \sum_{j=0}^{n} w_j \|V_j - x\|^2 + \|x - z\|^2,$$

$$\sum_{j=0}^{n} w_j \|V_j - x\|^2 = r^2 - \|x - z\|^2,$$

$$\sum_{j=0}^{n} w_j \|V_j - x\|^2 = e^k(x), \tag{3.28}$$

$$e^k(x) \geq \min_{0 \leq i \leq n} \{\|x - V_i\|\}^2. \tag{3.29}$$

Since both side of (3.29) are positive numbers, we can take square root of the both sides; moreover, L_c is a positive number.

$$L_c \sqrt{e^k(x)} \geq L_c \min_{1 \leq i \leq r} \{\|x - V_i\|\}. \tag{3.30}$$

Using (3.27) and (3.30), (3.26) is satisfied. □

Note that, by (3.28), the global uncertainty function $e^k(x)$ defined in (3.14) and Definition 19 is simply the weighted average of the squared distance of x from the vertices of the simplex that contains x.

The other lemma which is essential for analysis the convergence of Algorithm 3.2, is that the maximum circumradius of Δ^k is bounded.

Lemma 16. *Consider Δ^k as a Delaunay triangulation of a set of triangulation points S_T^k at step k. The maximum circumradius of Δ^k, r_{max}^k, is bounded as follows:*

$$r_{max}^k \leq \kappa L_1 \sqrt{1 + \left(\kappa \frac{L_1}{2(\kappa - 1)\delta_O}\right)^2}, \tag{3.31}$$

where L_1 is the maximum edge length of the enclosing simplex Δ_e, κ is the extending parameter (Definition 13), and δ_O is the minimum distance from O (the body center of the exterior simplex) to the boundary of Δ_e.

Proof. Consider Δ_x as a simplex in Δ^k which has the maximum circumradius. Define z as the circumcenter of Δ_x, and x as a vertex of Δ_x which is not a vertex of the exterior simplex. If z is located inside of the exterior simplex Δ_E, then,

$$r^k_{\max} = \|z - x\| \leq L_1\kappa, \tag{3.32}$$

which shows lemma in this case. If z is located outside of Δ_E, then define z_p as the point closest to z in the exterior simplex. That is,

$$\begin{aligned} z_p &= \text{argmin}_{y\in\Delta_E}\|z - y\| \\ \text{subject to } & a^T_{i,E}\, y \leq b_{i,E}, \forall\ 1 \leq i \leq n+1, \end{aligned} \tag{3.33}$$

where $a^T_{i,E}\, y \leq b_{i,E}$ defines the i'th face of the exterior simplex. Define $A_a(Z_p)$ as the set of active constraints at Z_p in the constraints of (3.33), and v as a vertex of the exterior simplex in which all constraints in $A_a(z_p)$ are active at z_p; then, using the optimality condition at Z_p, it follows that

$$\begin{aligned} (z - z_p)^T(z_p - v) = 0, \\ \|z - v\|^2 = \|z - z_p\|^2 + \|v - z_p\|^2. \end{aligned} \tag{3.34}$$

Now consider p as the intersection of the line segment zx with the boundary of the exterior

simplex; then,

$$\|z - x\| = \|z - p\| + \|p - x\|. \tag{3.35}$$

Furthermore, by construction,

$$\|z - z_p\| \leq \|z - p\|. \tag{3.36}$$

Since p is on the exterior simplex Δ_E, and x is inside of the enclosing simplex Δ_e, it follows that

$$\|p - x\| \geq (\kappa - 1)\delta_O. \tag{3.37}$$

Note that x is a vertex of the simplex Δ_x, and v is a point in S^k. Since the triangulation Δ^k is Delaunay, v is located outside of the interior of the circumsphere of Δ_x. Recall, z is the circumcenter of Δ_x; thus,

$$R_x = \|x - z\| \leq \|z - v\|, \tag{3.38}$$

where R_x is the circumcenter of L. Using (3.34), (3.35) and (3.38) it follows that

$$\|z - p\|^2 + \|p - x\|^2 + 2\|z - p\|\|p - x\| \leq \|z - z_p\|^2 + \|v - z_p\|^2. \tag{3.39}$$

By (3.36), (3.37) and (3.39), it follows that

$$2(\kappa - 1)\delta_O\|z - z_p\| \leq \|z - z_p\|^2.$$

Note that $\|v - z_p\| \leq \kappa L_1$; thus,

$$\|z - z_p\| \leq \kappa^2 \frac{L_1^2}{2(\kappa - 1)\delta_O}. \tag{3.40}$$

Combining (3.34) and (3.40), noting that $r^k_{max} \leq \|z - v\|$, (3.31) is satisfied. □

The final lemma which is required to establish the convergence of Algorithm 3.2, given below, establishes an important property of the feasible boundary projection.

Lemma 17. *Consider Δ^k as a Delaunay triangulation of Algorithm 3.2 at step k, and x'_k as the feasible boundary projection of x_k. If the extending parameter κ satisfies* (3.10)*, then, for any point $V \in S^k$,*

$$\frac{\|V - x'_k\|}{\|V - x_k\|} \geq \frac{1}{2}. \tag{3.41}$$

Proof. If a feasible boundary projection is not performed at step k, $x'_k = x_k$, and (3.41) is satisfied trivially. Otherwise, consider Δ^k_i in Δ^k as the boundary simplex, with circumcenter Z and circumradius R, which is used in the feasible boundary projection (see Definition 18). First, we will show that, for a value of κ that satisfies (3.10), Z is not in L. This fact is shown by contradiction; thus, first assume that Z is in L. Define A as a shared vertex of Δ^k_i and the exterior simplex, and $V_1, V_2, \ldots, V_{n+1}$ as the vertices of the initial simplex Δ_i (see Definition 11). Since the body center of the exterior simplex $O \in S^k_T$, and Δ^k is Delaunay,

$$\begin{gathered}
\|O - Z\| \geq \|A - Z\|, \\
\|O - Z\| \geq \min_{y \in L}\{\|A - y\|\}, \\
\|O - Z\| + \|O - y\| \geq \min_{y \in L}\{\|A - y\|\} + \|O - y\|, \\
2 \max_{y \in L}\{\|O - y\|\} \geq \|O - A\|, \\
\|O - A\| \geq n\kappa \min_{1 \leq j \leq n+1}\{\|O - V_j\|\}, \\
2 \max_{y \in L}\{\|y - O\|\} \geq n\kappa \min_{1 \leq j \leq n+1}\{\|V_j - O\|\}.
\end{gathered} \tag{3.42}$$

However, (3.42) is in contradiction with (3.10); thus, Z is not in L; therefore, by Definition

18, x'_k is on the line segment Zx_k.

Now we will show (3.41). Consider V_p as the orthogonal projection of V onto the line through x_k, x'_k and Z; then, we may write $V_p = \beta x_k$ and, by construction:

$$\|V - x_k\|^2 = \|V - V_p\|^2 + \|V_p - x_k\|^2,$$
$$\|V - x'_k\|^2 = \|V - V_p\|^2 + \|V_p - x'_k\|^2,$$
$$\|V - Z\|^2 = \|V - V_p\|^2 + \|V_p - Z\|^2.$$

Since Z is outside of L, x_k is in L, x'_k is on the boundary of L, and they are collinear ; then, there is a real number $0 \leq \alpha \leq 1$, such that $x'_k = Z + \alpha\,(x_k - X)$, which leads to

$$\|V_p - x_k\|^2 = \|V_p - Z\|^2 + 2(Z - V_p)^T(x_k - Z) + 2\|x_k - Z\|^2,$$
$$\|V_p - x'_k\|^2 = \|V_p - Z\|^2 + 2\alpha(Z - V_p)^T(x_k - Z) + 2\alpha^2\|x_k - Z\|^2.$$

Therefore, above equations are simplified to.

$$\|V - x_k\|^2 = \|V - Z\|^2 + 2(Z - V_p)^T(x_k - Z) + 2\|x_k - Z\|^2,$$
$$\|V - x'_k\|^2 = \|V - Z\|^2 + 2\alpha(Z - V_p)^T(x_k - Z) + 2\alpha^2\|x_k - Z\|^2.$$

By defining $\beta = \frac{(Z-V_p)^T(x_k-Z)}{\|x_k-Z\|\|V-Z\|}$ and $\gamma = \frac{\|x_k-Z\|}{\|V-Z\|}$; the above equation is simplified to:

$$\frac{\|V - x'_k\|^2}{\|V - x_k\|^2} = \frac{1 + 2\,\alpha\beta\gamma + \alpha^2\gamma^2}{1 + 2\,\beta\gamma + \gamma^2}. \tag{3.43}$$

By construction, x_k is inside the circumsphere of Δ_i^k. Moreover, Δ^k is a Delaunay triangulation for S^k, and $V \in S^k$; thus, V is outside the interior of circumsphere of Δ_i^k. As a result,

$||Z - x_k|| \leq ||Z - V||$. Moreover, γ is trivially positive; thus, $0 \leq \gamma \leq 1$. Now we identify lower and upper bounds for β:

$$-||Z - V_p||||x_k - Z|| \leq (Z - V_p)^T(x_k - Z) \leq ||Z - V_p||||x_k - Z||,$$

$$||Z - V_p|| \leq ||Z - V||,$$

$$-||Z - V||||x_k - Z|| \leq (Z - V_p)^T(x_k - Z) \leq ||Z - V||||x_k - Z||,$$

$$-1 \leq \beta \leq 1.$$

The right hand side of (3.43) is a function of α, β, γ. Moreover, it is shown that $0 \leq \alpha \leq 1$, $-1 \leq \beta \leq 1$, and $0 \leq \gamma \leq 1$. In order to show (3.41), It is sufficient to prove that the minimum of this three dimensional function, denoted by $\Gamma(\alpha, \beta, \gamma)$, in the box that characterized its variable is $\frac{1}{4}$.

The function Γ is a fractional linear function of β, thus,

$$\begin{aligned}
\Gamma(\alpha, \beta, \gamma) &\geq \min\{\Gamma(\alpha, -1, \gamma), \Gamma(\alpha, 1, \gamma)\}, \\
\Gamma(\alpha, 1, \gamma) &= (\frac{1 + \alpha\gamma}{1 + \gamma})^2 \geq \frac{1}{(1 + \gamma)^2} \geq \frac{1}{4}, \\
\Gamma(\alpha, -1, \gamma) &= (\frac{1 - \alpha\gamma}{1 - \gamma})^2 \geq (\frac{1 - \gamma}{(1 - \gamma)})^2 \geq 1, \\
\Gamma(\alpha, \beta, \gamma) &\geq \frac{1}{4}.
\end{aligned} \tag{3.44}$$

Using (3.43) and (3.44), (3.41) is verified. □

Finally, using the above lemmas, we can prove the convergence of Algorithm 3.2 to the global minimum in the feasible domain L.

Theorem 7. *If Algorithm 3.2 is terminated at step k, and $y_0 \leq f(x^*)$, then there are real*

bounded numbers A_1, A_2, A_3 such that

$$\min_{z \in S^k} f(z) - f(x^*) \leq A_1\, \delta_{des} + A_2\, \sqrt{\delta_{des}} + A_3\, \sqrt[4]{\delta_{des}}. \tag{3.45}$$

Proof. Define S_E^k, S_T^k, $r_{\max}^k$, and L_2^k as the evaluation set, the triangulation set, the maximum circumradius of Δ^k, and the maximum edge length of Δ^k, respectively, where Δ^k is a Delaunay triangulation of S_T^k. Define x_k as the outcome of Algorithm 3.2 though step 4, which at step k satisfies (3.17), and x_k' as the feasible boundary projection of x_k on L.

Define $y_1 \in S_E^k$ as the point which minimizes $\delta = \min_{x \in S_E^k} \|x - x_k\|$, and the parameters $\{A, B, C, D, E\}$ as follows:

$$A = \max\{K_f + L_f K_g L_O, L_f L_O\}, \tag{3.46a}$$

$$B = \max\{K_{pf} + L_f K_g L_O, L_f L_O\}, \tag{3.46b}$$

$$C = \max\{4\, A\, r_{\max}^k, L_p + 4\, B\, r_{\max}^k\}, \tag{3.46c}$$

$$\begin{aligned} D = 2\, \max\{&A\, L_c\, \sqrt{2 r_{\max}^k}, B\, L_c\, \sqrt{2\, r_{\max}^k}, \\ &\sqrt{L_c\, L_2^k\, L_p\, A\, r_{\max}^k}\}, \end{aligned} \tag{3.46d}$$

$$E = \sqrt{2\, L_c\, L_2^k\, L_p\, A}\, \sqrt[4]{2\, r_{\max}^k}. \tag{3.46e}$$

We will now show that

$$\min_{z \in S^k} f(z) - f(x^*) \leq C\delta + D\, \sqrt{\delta} + E\, \sqrt[4]{\delta}, \tag{3.47}$$

where x^* is a global minimizer of $f(x^*)$.

During the iterations of Algorithm 3.2, there two possible cases for $s^k(x)$. The first case is when $s^k(x) = p^k(x)$. In this case, via (3.17a), $p^k(x_k) \leq y_0$, and therefore

$p^k(x_k) \leq f(x^*)$. Since $y_1 \in S_E^k$, it follows that $p^k(y_1) = f(y_1)$. Moreover, L_p is a Lipchitz constant for $p^k(x)$; therefore,

$$p^k(y_1) - p^k(x_k) \leq L_p\,\delta, \quad f(y_1) - p^k(x_k) \leq L_p\,\delta,$$
$$f(y_1) - f(x^*) \leq L_p\,\delta, \quad \min_{z \in S_e^k} f(z) - f(x^*) \leq L_p\,\delta.$$

which shows that (2.62) is true in this case.

The other case is when $s^k(x) = (p^k(x) - y_0)/e^k(x)$ in the interior simplices, and $s^k(x) = (p^k(x) - y_0)/(-g_L^F(x))$ on the faces of Δ^k. For this case, we will show that (2.62) is true when x^* is not in an interior simplex. When x^* is in an interior simplex, (2.62) can be shown in an analogous manner.

Define F_i^k as a face in Δ^k which includes x_I^*, the interior projection (see Definition 18) of x^*, and Δ_i^k as an interior simplex which includes x_I^*. Define $\hat{x}_i^k$ and $\tilde{x}_i^k$ as the points which maximize $e_i^k(x)$ and $-g_L^{F_i^k}(x)$ in Δ_i^k and F_i^k, respectively, where $e_i^k(x)$ and $-g_L^{F_i^k}(x)$ are the local uncertainty function in Δ_i^k and the linearized constraint function (see Definition 17) in F_i^k.

According to Lemma 14 and (3.46b),

$$p^k(x_I^*) - f(x^*) \leq B\,e^k(x_I^*) - B\,g_L^{F_i^k}(x_I^*),$$
$$p^k(x_I^*) - f(x^*) \leq B\,e_i^k(\hat{x}_i^k) - B\,g_L^{F_i^k}(\tilde{x}_i^k). \tag{3.48}$$

Further, by replacing the linear interpolation $p^k(x)$ with the linear function $L_i^k(x)$, in which $L_i^k(x) = f(x)$ at the vertices of F, and using $\min_{z \in S^k} f(z) \leq L^k(x_I^*)$ and the fact that the

Hessian of $L_i^k(x)$ is zero, and noting (3.46a), (3.48) may be modified to:

$$\min_{z\in S_E^k}\{f(z)\} - f(x^*) \le A\, e_i^k(\hat{x}_i^k) - A\, g_L^{F_i^k}(\tilde{x}_i^k). \tag{3.49}$$

If the search function at x_k is defined by (3.15), take $\hat{e}_k = e^k(x_k)$; on the other hand, if it is defined by (3.16), take $\hat{e}_k = -g_L^{F_j^k}(x_k)$ where F_j^k is the face of Δ^k which includes x_k.

Since $2\, r_{\max}^k$ is a Lipchitz constant for $e^k(x)$ (see Lemma 4 in chapter 2), by using Lemma 15 above, it follows

$$\begin{aligned} \hat{e}_k &\le \max\{2\, r_{\max}^k \delta, L_c \sqrt{2\, r_{\max}^k \delta}\}, \\ \hat{e}_k &\le 2\, r_{\max}^k \delta + L_c \sqrt{2\, r_{\max}^k \delta} \end{aligned} \tag{3.50}$$

If $e_i^k(\hat{x}) - g_L^{F_i^k}(\tilde{x}_i^k) \le 2\,\hat{e}_k$, then by (3.49) and (3.50), (2.62) is satisfied. Otherwise, dividing $f(x^*) - y_0 \ge 0$ by this expression with the opposite inequality,

$$\frac{2\, f(x^*) - 2\, y_0}{e_i^k(\hat{x}) - g_L^{F_i^k}(\tilde{x}_i^k)} < \frac{f(x^*) - y_0}{\hat{e}_k}. \tag{3.51}$$

Using (2.60b), (3.17c) and (3.51)[5]

$$\begin{gathered} \frac{p^k(x_k) - y_0}{\hat{e}_k} \le \frac{p^k(\hat{x}_i^k) - y_0}{e_i^k(\hat{x}_j^k)}, \quad \frac{p^k(x_k) - y_0}{\hat{e}_k} \le \frac{p^k(\tilde{x}_i^k) - y_0}{-g_L^{F_i^k}(\hat{x}_j^k)}, \\ \frac{p^k(x_k) - f(x^*)}{\hat{e}_k} \le \frac{p^k(\hat{x}_i^k) + p^k(\tilde{x}_i^k) - 2\, f(x^*)}{e_i^k(\hat{x}_j^k) - g_L^{F_i^k}(\hat{x}_j^k)}. \end{gathered}$$

According to Assumption 9, L_p is a Lipchitz constant for $p^k(x)$; noting $\max\{\|\hat{x}_i^k - x_I^*\|, \|\tilde{x}_i^k -$

[5] If $x \le a/b$, $x \le c/d$, and $b, d > 0$ then $x \le (a+c)/(b+d)$.

$x_I^*\|\} \leq L_2^k$ and (3.48), the above equation thus simplifies to

$$\frac{p^k(x_k) - f(x^*)}{\hat{e}_k} \leq 2\,B + \frac{2\,L_2^k\,L_p}{e_i^k(\hat{x}_j^k) - g_L^{F_i^k}(\hat{x}_j^k)},$$

$$f(y_1) - p^k(x_k) \leq L_p\delta,$$

$$f(y_1) - f(x^*) \leq L_p\delta + \hat{e}_k\{2\,B + \frac{2\,L_2^k L_p}{e_i^k(\hat{x}_j^k) - g_L^{F_i^k}(\hat{x}_j^k)}\}.$$

Using (3.50), (3.49) and $f(x^*) \leq \min_{z\in S^k}\{f(z)\} \leq f(y_1)$, it follows:

$$\min_{z\in S^k}\{f(z)\} - f(x^*) \leq$$

$$L_p\delta + \hat{e}_k\{2\,B + \frac{2\,L_2^k L_p A}{\min_{z\in S^k}\{f(z)\} - f(x^*)}\},$$

$$[\min_{z\in S^k}\{f(z)\} - f(x^*)]^2 \leq$$

$$2\,L_2^k L_p A\hat{e}_k + \{L_p\delta + 2\,B\,e_k\}[\min_{z\in S^k}\{f(z)\} - f(x^*)].$$

Perform the quadratic inequality[6] on the above, and the triangular inequality[7] on the square root of (3.50), it follows that:

$$\min_{z\in S^k}\{f(z)\} - f(x^*) \leq \sqrt{2\,L_2^k\,L_p\,A\,\hat{e}_k} + [L_p\delta + 2\,B\,\hat{e}_k],$$

$$\sqrt{\hat{e}_k} \leq \sqrt{2\,r_{\max}^k}\,\sqrt{\delta} + \sqrt{L_c}\,\sqrt[4]{2\,r_{\max}^k\delta}.$$

By using above equations and (3.50), (2.62) is satisfied.

According to Lemma 16, $r_{\max}^k$ is bounded above; since L_2^k is also bounded, since the feasible domain is bounded, it follows that C, D and E are bounded real numbers.

[6] If $A, B, C > 0$, and $A^2 \leq C + BA$ then $A \leq \sqrt{C} + B$.

[7] If $x, y > 0$, then $\sqrt{x+y} \leq \sqrt{x} + \sqrt{y}$.

Furthermore, according to Lemma 17,

$$\min_{z \in S^k} \|x'_k - y\| \geq \frac{\delta}{2}. \tag{3.52}$$

Since $\|x'_k - y\| = \delta_{\text{des}}$, we have

$$\begin{aligned} &\min_{z \in S^k}\{f(z)\} - f(x^*) \leq \varepsilon_k, \\ &\text{where} \quad \varepsilon_k = 2C\delta_{\text{des}} + D\sqrt{2\delta_{\text{des}}} + E\sqrt[4]{2\delta_{\text{des}}} \end{aligned} \tag{3.53}$$

Thus, (3.45) is satisfied for $A_1 = 2\,C$, $A_2 = \sqrt{2}\,D$, and $A_3 = \sqrt[4]{2}\,D$. □

In the above theorem, it is shown that a point can be obtained whose function value is arbitrarily close to the global minimum, as long as Algorithm 3.2 is terminated with a sufficiently small value for δ_{des}. This result establishes convergence with respect to the function values for a finite number of steps k. In the next theorem, we present a property of Algorithm 3.2 establishing its convergence to a global minimizer in the limit that $k \to \infty$.

Theorem 8. *If Algorithm 3.2 is not terminated at step* 6*, then the sequence* $\{x'_k\}$ *has an* ω*-limit point* [8] *which is a global minimizer of* L*.*

Proof. Define z_k as a point in S^k that has the minimum objective value. By construction, $f(z_r) \leq f(z_l)$, if $r > l$; thus, $f(z_k)$ is monotonically non-increasing. Moreover, according to Theorem 7,

$$\begin{aligned} &f(z_k) - f(x^*) \leq A_1\delta_k + A_2\sqrt{\delta_k} + A_3\sqrt[4]{\delta_k}, \\ &\text{where} \quad \delta_k = \min_{x \in S^k}\{\|x - x_k\|\}, \end{aligned} \tag{3.54}$$

[8] The point $\hat{x}$ is an ω-limit point (see [64]) of the sequence x_k, if a subsequence of x_k converges to $\hat{x}$.

where $f(x^*)$ is the global minimum. According to Remark 17, for any arbitrarily small $\delta > 0$, there is a k such that $\min_{x \in S^k}\{\|x - x_k\|\} \leq \delta$; thus,

$$f(z_k) - f(x^*) \leq A_1\delta + A_2\sqrt{\delta} + A_3\sqrt[4]{\delta}. \tag{3.55}$$

Thus, for any $\varepsilon > 0$; there is a k such that $0 \leq f(z_k) - f(x^*) \leq \varepsilon$. Now consider x_1 as an *omega*-limit point of the sequence $\{z_k\}$; thus, there is a subsequence $\{a_i\}$ of $\{z_k\}$ that converges to x_1. Since $f(x)$ is continuous,

$$f(x_1) = \lim_{i \to \infty} f(a_i) = f(x^*), \tag{3.56}$$

which establishes that the x_1 is a global minimizer of $f(x)$. □

Remark 20. *Theorems 7 and 8 ensure that Algorithm 3.2 converges to the global minimum if $y_0 \leq f(x^*)$; however, if $y_0 > f(x^*)$, in a manner analogous to Theorem 6 in chapter 2, it can be shown that Algorithm 3.2 converges to a point where the function value is less than or equal to y_0.*

3.5 Results

In this section, we apply Algorithm 3.2 to some representative examples in two and three dimensions with different types of convex constraints to analyze its performance. Three different test functions for $f(x)$, where $x = [x_1, x_2, \ldots, x_n]^T$, are considered:

Parabolic function:

$$f(x) = \sum_{i=1}^{n} x_i^2. \tag{3.57}$$

Rastrigin function: Defining $A = 2$,

$$f(x) = A\,n + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi\, x_i) \right]. \tag{3.58}$$

Shifted Rosenbrock function[9]: Defining $p = 10$,

$$f(x) = \sum_{i=1}^{n-1} \left[(x_i)^2 + p\,(x_{i+1} - x_i^2 - 2\,x_i)^2 \right]. \tag{3.59}$$

In the unconstrained setting, the global minimizer of all three test functions considered is the origin.

Another essential part of the class of problems considered in (3.1) is the constraints that are imposed; this is, in fact, a central concern of the present chapter.

Note that all simulations in this section are stopped when $\min_{y \in S^k}\{\|y - x_k\|\} \leq 0.01$.

3.5.1 2D with circular constraints

We first consider 2D optimization problems in which the feasible domain is a circle. In problems such as this, in which there is only one constraint ($m = 1$), $g_F^L(x) = 0$ for all faces F at all steps. Thus, no searches are performed on the faces; rather, all searches are performed in the interior simplices, with some points x_k (that is, any point x_k that lands within the circumsphere of a boundary simplex) projected out to the feasible domain boundary.

[9]This is simply the classical Rosenbrock function with its unconstrained global minimizer shifted to the origin.

In this subsection, two distinct circular constraints are considered:

$$(x_1 + 1.5)^2 + (x_2 + 1.5)^2 \leq 5.5, \tag{3.60a}$$

$$(x_1 + 1.5)^2 + (x_2 + 1.5)^2 \leq 3.8. \tag{3.60b}$$

The constraint (3.60a) includes the origin (and, thus, the global minimizer of the test functions considered), whereas the constraint (3.60b) does not. For the problems considered in this subsection, we take $\kappa = 1.4$ for all simulations performed, which satisfies (3.10).

It is observed that, for test functions (3.57) and (3.58), the global minimizer in the domain that satisfies (3.60b) is $x^* = [-0.1216; -0.1216]$, and the global minima are $f(x^*) = 0.0301$ and $f(x^*) = 0.8634$, respectively. For the Rosenbrock function (3.59), the global minimizer is at $x^* = [-0.0870; -0.1570]$, and the global minimum is $f(x^*) = 0.0085$ [it is easy to check that this point is KKT (see [19]), and that the problem is convex].

Algorithm 3.2 is implemented for six optimization problems constructed with the above test functions and constraints. In each problem, the parameter y_0 in Algorithm 3.2 is chosen be a lower bound for $f(x^*)$. Two different cases are considered: one with $y_0 = f(x^*)$, and one with $y_0 < f(x^*)$.

The first two problems consider the parabolic test function (3.57), first with the constraint (3.60a), then with the constraint (3.60b). For these problems, in the case with $y_0 = f(x^*)$ (that is, $f(x^*) = 0$ for constraint (3.60a), and $f(x^*) = 0.0301$ for constraint (3.60b)), a total of 8 and 7 function evaluations are performed by the optimization algorithm for the constraints given by (3.60a) and (3.60b), respectively (see Figures 3.5a and 3.5g). This remarkable convergence rate is, of course, due to the facts that the global minimum is known and both the function and the curvature of the boundary are smooth.

For the two problems related to the parabolic test function in the case with $y_0 < f(x^*)$ (we take $y_0 = -0.1$ in both problems), slightly more exploration is performed before termination; a total of 15 and 13 function evaluations are performed by the optimization algorithm for the constraints given by (3.60a) and (3.60b), respectively (see Figures 3.5d and 3.5j). Another observation is that, for the constraint given by (3.60b), more function evaluations on the boundary are performed; this is related to the fact that the function value on the boundary is reduced along a portion of the boundary.

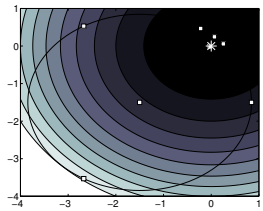
The next two problems consider the Rastrigin test function (3.58), first with the constraint (3.60a), then with the constraint (3.60b). With the constraint (3.60a), this problem has 16 local minima in the interior of the feasible domain, and 10 constrained local minima on the boundary of L; with the constraint (3.60b), it has 12 local minima in the interior of the feasible domain, and 4 constrained local minima on the boundary of L. For these problems, in the case with $y_0 = f(x^*)$ (that is, $f(x^*) = 0$ for constraint (3.60a), and $f(x^*) = 0.8634$ for constraint (3.60b)), a total of 41 and 38 function evaluations are performed by the optimization algorithm for the constraints given by (3.60a) and (3.60b), respectively (see Figures 3.5b and 3.5h). As expected, more exploration is needed for this test function than for a parabola. Another observation for the problem constrained by (3.60b) is that two different regions are characterized by rather dense function evaluations, one in the vicinity of the global minimizer, and the other in a neighborhood of a local minima (at $x_1 = [-1; 0]$) where the cost function value ($f(x_1) = 1$) is relatively close to the global minimum $f(x^*) = 0.8634$.

For the two problems related to the Rastrigin test function in the case with $y_0 < f(x^*)$ (we take $y_0 = -0.2$ for the problem constrained by (3.60a), and $y_0 = 0.5$ for the problem constrained by (3.60b)), more exploration is, again, performed before termination;

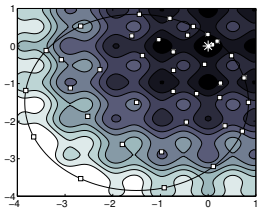
a total of 70 and 64 function evaluations are performed by the optimization algorithm for the constraints given by (3.60a) and (3.60b), respectively (see Figures 3.5e and 3.5k). For the problem constrained by (3.60b), three different regions are characterized by rather dense function evaluations: in the vicinity of $[-1;\ 0]$, in the vicinity of $[0;\ -1]$, and in the vicinity of the global minimizer at $[-0.1216; -0.1216]$.

The last two problems of this subsection consider the Rosenbrock test function (3.59), first with the constraint (3.60a), then with the constraint (3.60b). This challenging problem is characterized by a narrow valley, with a relatively flat floor, in the vicinity of the curve $x_2 = x_1^2$. For these problems, in the case with $y_0 = f(x^*)$ (that is, $f(x^*) = 0$ for constraint (3.60a), and $f(x^*) = 0.0085$ for constraint (3.60b)), a total of 29 and 31 function evaluations are performed by the optimization algorithm for the constraints given by (3.60a) and (3.60b), respectively (see Figures 3.5c and 3.5i).

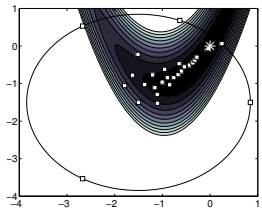
For the two problems related to the Rosenbrock test function in the case with $y_0 < f(x^*)$ (we take $y_0 = -0.5$ in both problems), more exploration is performed before termination; a total of 47 and 54 function evaluations are performed by the optimization algorithm for the constraints given by (3.60a) and (3.60b), respectively (see Figures 3.5f and 3.5l). As with the previous problems considered, exact knowledge of $f(x^*)$ significantly improves the convergence rate. Additionally, such knowledge confines the search to a smaller region around the curve $y = x^2$, and fewer boundary points are considered for function evaluations.



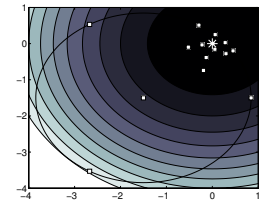
a Parabola, in (3.60a), $y_0 = f(x^*)$.



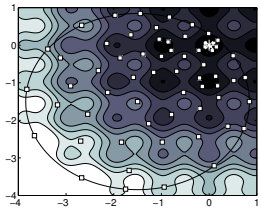
b Rastrigin, in (3.60a), $y_0 = f(x^*)$.



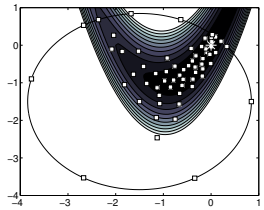
c Rosenbrock, in (3.60a), $y_0 = f(x^*)$.



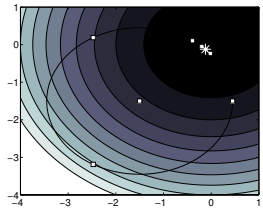
d Parabola, in (3.60a), $y_0 < f(x^*)$.



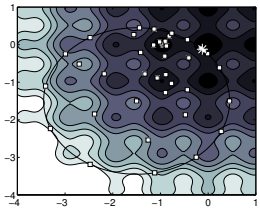
e Rastrigin, in (3.60a), $y_0 < f(x^*)$.



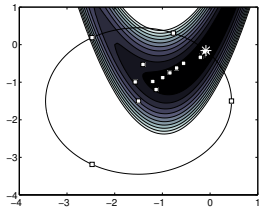
f Rosenbrock, in (3.60a), $y_0 < f(x^*)$.



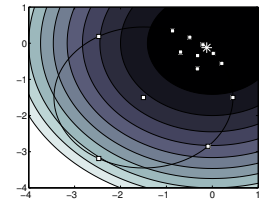
g Parabola, in (3.60b), $y_0 = f(x^*)$.



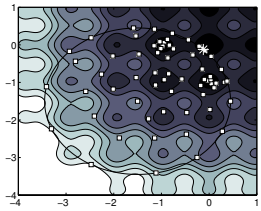
h Rastrigin, in (3.60b), $y_0 = f(x^*)$.



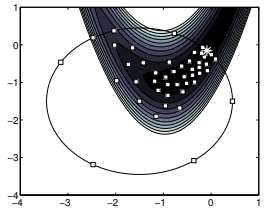
i Rosenbrock, in (3.60b), $y_0 = f(x^*)$.



j Parabola, in (3.60b), $y_0 < f(x^*)$.

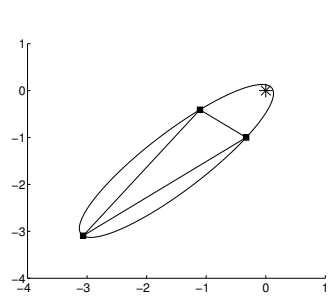


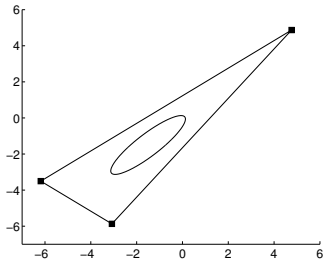
k Rastrigin, in (3.60b), $y_0 < f(x^*)$.

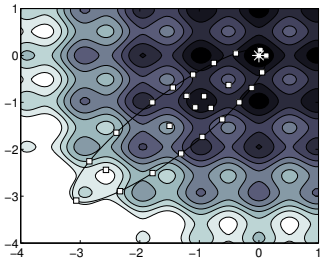


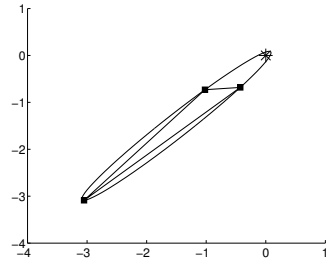
l Rosenbrock, in (3.60b), $y_0 < f(x^*)$.

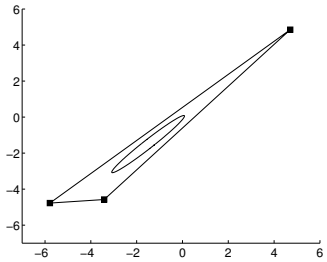
Figure 3.5: Δ-DOGS(C) on 2D circle. Location of function evaluations when applying Algorithm 3.2 to the 2D parabola (3.57), Rastrigin function (3.58), and Rosenbrock function (3.59), in a circle.


a Initial simplex for (3.61a).


b Exterior simplex for (3.61a).


c Evaluation points for constraint (3.61a).


d Initial simplex for (3.61b).


e Exterior simplex for (3.61b).

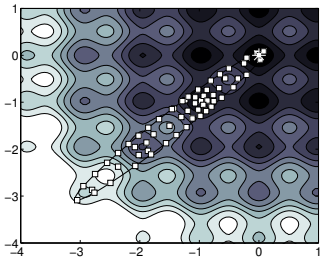

f Evaluation points for constraint (3.61b).

Figure 3.6: Δ-DOGS(C) on 2D ellipce. Implementation of Algorithm 3.2 with $y_0 = 0$ on the 2D Rastrigin problem (3.58) within the ellipse (3.61).

3.5.2 2D with elliptical constraints

The next two constraints considered are diagonally-oriented ellipses with aspect ratios 4 and 10:

$$(x + y + 3)^2 + 16\,(x - y)^2 \leq 10, \tag{3.61a}$$

$$(x + y + 3)^2 + 100\,(x - y)^2 \leq 10. \tag{3.61b}$$

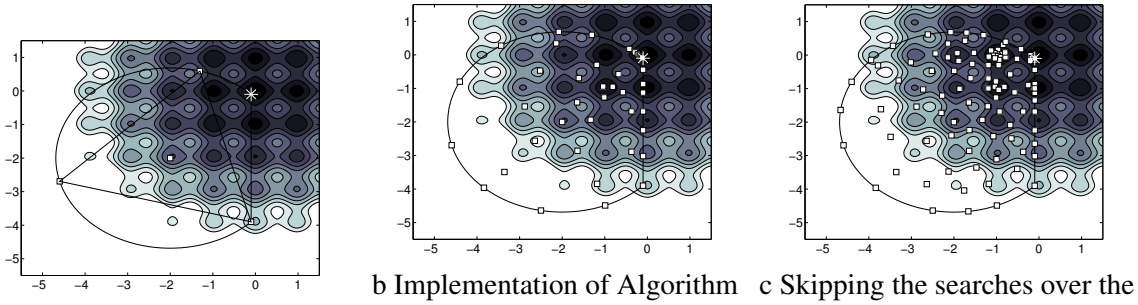
These constraints are somewhat more challenging to deal with than those considered in §3.5.1. As in §3.5.1, $g_F^L(x) = 0$ for all faces F at all steps; thus, no searches are performed on the faces.

The Rastrigin function has 4 unconstrained local minimizers inside each of these ellipses. Additionally, there are 6 constrained local minimizers on the boundary (3.61a);

note that there are no constrained local minimizers on the boundary (3.61b).

In this subsection, Algorithm 3.2 is applied to the 2D Rastrigin function (3.58) for the two different elliptical constraints considered. The unconstrained global minimizer of this function is at $x^* = [0, 0]$, where the global minimum is $f(x^*) = 0$; this unconstrained global minimizer is contained within the feasible domain for both (3.61a) and (3.61b). We take $y_0 = f(x^*) = 0$ for all simulations reported in this subsection.

For the problems considered in this subsection, we take $\kappa = 2$ for all simulations performed, which satisfies (3.10). Note that, though the aspect ratio of the feasible domain is relatively large (especially in (3.61b)), the minimum distance of the body center of the initial simplex from its vertices is relatively large; thus, a relatively small value for extending parameter κ may be used. As seen in Figure 3.6, for the cases constrained by (3.61a) and (3.61b), the simulations are terminated with 24 and 60 function evaluations, respectively. This indicates that the performance of the algorithm depends strongly on the aspect ratio of the feasible domain, as predicted by the analysis presented in §3.4, as the parameter K_g in Assumption 6 increases with the square of the aspect ratio of the elliptical feasible domain [see the explicit dependence on K_g in (3.46) and (2.62)]. In the case constrained by (3.61b), Figure 3.6d shows that the global minimizer lies outside of the smallest face of the initial simplex, and is situated relatively far this face. In this case, the reduced uncertainty $e^k(x)$ across this small face (a result of the fact that its vertices are close together) belies the substantially larger uncertainty present in the feasible domain far outside the initial simplex, in the vicinity of the global minimizer; this, in effect, slows convergence. Note that this situation only occurs when the curvature of the boundary is large. An alternative approach to address problems with large aspect ratio is to eliminate the highly-constrained coordinate direction(s) altogether, solve a lower-dimensional optimization problem, then locally



a Location of the initial simplex. b Implementation of Algorithm 3.2. c Skipping the searches over the faces.

Figure 3.7: Δ-DOGS(C) on combined constrained. Implementation of Algorithm 3.2 with $y_0 = f(x^*)$ on the 2D Rastrigin function (3.58) with constraints (3.62), with and without searching over the faces.

optimize the original problem in the vicinity of the lower-dimensional optimal point. Note that, through the construction of the enclosing simplex (see Definition 12), the highly-constrained coordinate directions can easily be identified and eliminated.

3.5.3 2D with multiple constraints

In this subsection, Algorithm 3.2 is implemented for problems characterized by the union of multiple linear or nonlinear constraints on the feasible domain, which specifically causes corners (that is, points at which two or more constraints are active) in the feasible domain. In such problems, the value of $g_F^L(x)$ is nonzero at some faces of the Delaunay triangulations; Algorithm 3.2 thus requires searches over the faces of the triangulations. The example considered in this subsection quantifies the importance of this process of searching over the faces of the triangulations in such problems.

The test function considered in this subsection is the 2D Rastrigin function (3.58),

and the feasible domain considered is as follows:

$$c_1(x) = (x_1 + 2)^2 + (x_2 + 2)^2 \leq 2 \times 1.9^2, \tag{3.62a}$$

$$c_2(x) = x_1 \leq -0.1. \tag{3.62b}$$

The Rastrigin function in the feasible domain characterized by (3.62) has 18 local minima inside the feasible domain, and 7 constrained local minima on the boundary of the feasible domain.

Note that the unconstraint global minimizer of (3.58) is not in the domain defined by the constraints (3.62).

The global minimizer of (3.58) within the feasible domain defined by (3.62) is $x^* = [-0.1; -0.1]$, and the global minimum is $f(x^*) = 0.7839$.

In order to benchmark the importance of the search over the faces of the triangulations in Algorithm 3.2, two different optimizations on the problem described above (both taking $y_0 = f(x^*)$ and $\kappa = 1.4$) are performed. In the first, we apply Algorithm 3.2. In the second, the same procedure is applied; however, the search over the faces of the triangulation is skipped.

As shown in Figure 3.7b, Algorithm 3.2 requires only 35 function evaluations before the algorithm terminates, and a point in the vicinity of the global minimizer is found. In contrast, as shown in Figure 3.7c, when the search over the faces of the triangulation is skipped in Algorithm 3.2, 104 function evaluations are required before the algorithm terminates. In problems of this sort, in which the constrained global minimizer is at a corner, Algorithm 3.2 requires searches over the faces in order to move effectively along the faces into the corner.

Note also in Figure 3.7b that, when a point in the corner is the global minimizer, Algorithm 3.2 converges slowly along the constraint boundary towards the corner. A potential work-around to this problem might be to switch to a derivative-free local optimization method (e.g. [52, 13, 50]) when a point in the vicinity of the solution near a corner is identified.

3.5.4 3D Problems

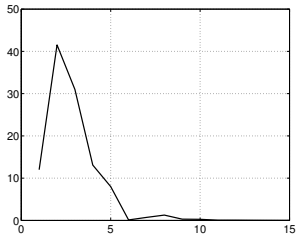
To benchmark the performance of Algorithm 3.2 on higher-dimensional problems, it was applied to the 3D parabolic (3.57), 3D Rastrigin (3.58) and 3D Rosenbrock (3.59) test functions with a feasible domain that looks approximately like the D-shaped feasible domain illustrated in Figure 3.7 rotated about the horizontal axis:

$$c_1(x) = \sum_{i=1}^{3} (x_i + 2)^2 \leq 3 \times 1.95^2, \tag{3.63a}$$

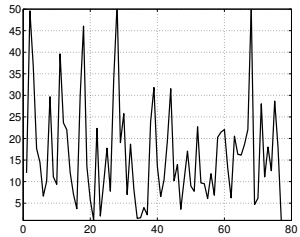
$$c_2(x) = x_1 \leq -0.05. \tag{3.63b}$$

The constrained global minimizer x^* for the 3D parabolic and 3D Rastrigin test functions in this case is at the corner $[-0.05; -0.05; -0.05]$, where both constraints are active. The constrained global minimizer x^* for the 3D Rosenbrock test functions is at $[-0.05; -0.0975; -0.1875]$, where one of the constraints $[c_2(x)]$ is active.

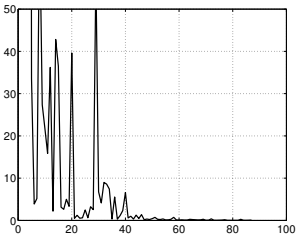
Algorithm 3.2 with $y_0 = 0$ and $\kappa = 1.4$ was applied to the problems described above. As shown in Figure 3.8, the parabolic, Rastrigin, and Rosenbrock test cases terminated with 15, 76 and 87 function evaluations, respectively. As mentioned in the last paragraph of §3.5.3, convergence in the first two of these test cases is slowed somewhat from what would be achieved otherwise due to the fact that the constrained global minimum lies in



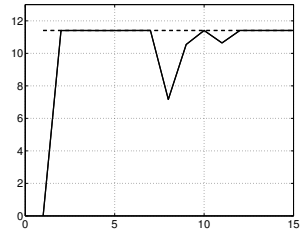
a Function values for Parabola (3.57).



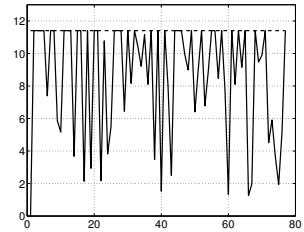
b Function values for Rastrigin (3.58).



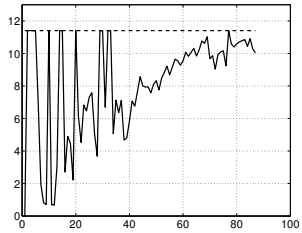
c Function values for Rosenbrock (3.59).



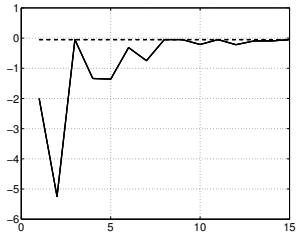
d $c_1(x)$ at evaluated points for Parabola.



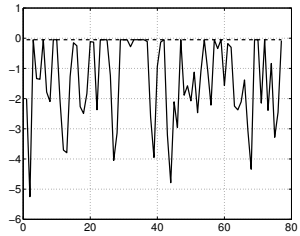
e $c_1(x)$ for Rastrigin.



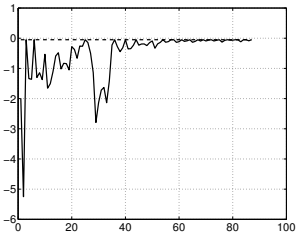
f $c_1(x)$ for Rosenbrock.



g $c_2(x)$ at evaluated points for Parabola.

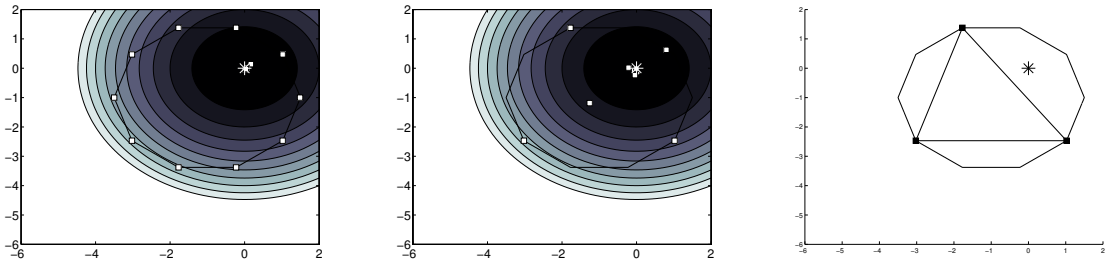


h $c_2(x)$ for Rastrigin.



i $c_2(x)$ for Rosenbrock.

Figure 3.8: Δ-DOGS(C) in 3D. Algorithm 3.2 applied to the 3D parabolic (3.57), Rastrigin (3.58), and Rosenbrock (3.59) test functions, with a feasible domain defined by (3.63).



a Δ-DOGS. b Δ-DOGS(C) c Initial simplex

Figure 3.9: Δ-DOGS versus Δ-DOGS(C). Implementation of Algorithm 3.2 of the present chapter, and Algorithm 4 of chapter 2, to a parabolic test function constrained by a decagon.

a corner of the feasible domain. Note that the Rastrigin case is fairly difficult, as this function is characterized by approximately 140 local minima inside the feasible domain, and 50 local minima on the boundary of the feasible domain; nonetheless, Algorithm 3.2 identifies the nonlinearly constrained global minimum in only 76 function evaluations.

3.5.5 Linearly constrained problems

In this section, Algorithm 3.2 is compared with Algorithm 4 in chapter 2 of this work (the so-called "Adaptive K" algorithm) for a linearly constrained problem with a smooth, simple objective function, but a large number of corners of the feasible domain.

The test function considered is the 2D parabola (3.57), and the set of constraints applied is

$$\begin{aligned} a_i &= \begin{bmatrix} \cos(2\pi i/m) & \sin(2\pi i/m) \end{bmatrix}^T, \\ b_i &= 2.5\,\cos(\pi/m) - \begin{bmatrix} 1 & 1 \end{bmatrix} a_i, \\ a_i^T x &\leq b_i, \quad 1 \leq i \leq m. \end{aligned} \tag{3.64}$$

The feasible domain characterized by (3.64) is a uniform m-sided polygon with m vertices.

The feasible domain contains the unconstrained global minimum at $[0, 0]$, and $y_0 = f(x^*) = 0$ is used for both simulations reported.

The initialization of Algorithm 4 in chapter 2 requires m initial function evaluations, at the vertices of the feasible domain; the case with $m = 10$ (taking $r = 5$, as discussed in §4 of chapter 2) is illustrated in Figure 3.9a, and is seen to stop after 15 function evaluations (that is, 5 iterations after initialization at all corners of the feasible domain).

In contrast, as illustrated Figure 3.9c, Algorithm 3.2 of the present chapter (taking $\kappa = 1.4$), applied to the same problem, stops after only 10 function evaluations (that is, 6 iterations after initialization at the vertices of the initial simplex and its body center). Note that one feasible boundary projection is performed in this simulation, as the global minimizer is not inside the initial simplex.

These simulations show that, for linearly-constrained problems with smooth objective functions, it is not necessary to calculate the objective function at all vertices of the feasible domain. As the dimension of the problem and/or the number of constraints increases, the cost of such an initialization might become large. Using the algorithm developed in the present chapter, it is enough to initialize with between $n + 2$ and $2n + 3$ function evaluations.

3.5.6 Role of the extending parameter κ

The importance of the extending parameter κ is now examined by applying Algorithm 3.2 to the Rastrigin function (3.58) within a linearly-constrained domain:

$$x \leq 0.1, \tag{3.65a}$$

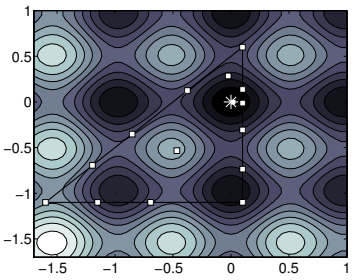
$$-1.1 \leq y, \tag{3.65b}$$

$$y - x \leq 0.5. \tag{3.65c}$$

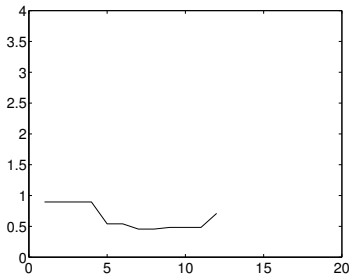
The Rastrigin function (3.58) has 3 local minimizers inside this triangle, and there are not any constrained local minimizers on the boundary of this triangle.

The feasible domain includes the unconstrained global minimizer in this case; thus, the constrained global minimizer is $[0, 0]$. Two different values of the extending parameter are considered, $\kappa = 1.1$ and $\kappa = 100$. It is observed (see Figure 3.10) that, for $\kappa = 1.1$, the algorithm converges to the vicinity of the solution in only 12 function evaluations; for $\kappa = 100$, 16 function evaluations are required, and the feasible domain is explored more.

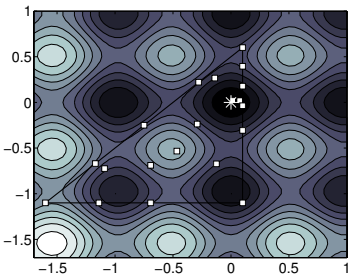
The main reason for this phenomenon is related to the value of the maximum circumradius (see Figures 3.10b and 3.10d). In the $\kappa = 100$ case, Algorithm 3.2 performs a few unnecessary function evaluations, close to existing evaluation points, at those iterations during which the maximum circumradius of the triangulation is large. The $\kappa = 1.1$ case better regulates the maximum circumradius of the triangulation, thus inhibiting such redundant function evaluations from occurring.



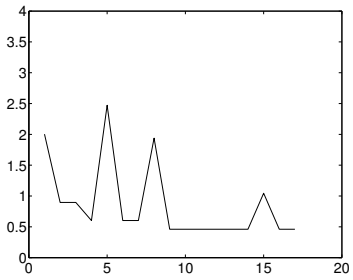
a Evaluation points for $\kappa = 1.1$.



b The maximum circumradius for $\kappa = 1.1$.



c Evaluation points for $\kappa = 100$.



d The maximum circumradius for $\kappa = 100$.

Figure 3.10: Role on the extension factor of the exterior simplex. Implementation of Algorithm 3.2 with $y_0 = 0$ on 2D Rastrigin problem (3.58) in a triangle characterized by (3.65) with $\kappa = 1.1$ and $\kappa = 100$.

3.5.7 Comparison with other Derivative-free methods

In this section, we briefly compare our new algorithm with two modern benchmark methods among derivative free optimization algorithms. The first method considered is MADS (mesh adaptive direction search; see [53]), as implemented in the NOMAD software package [65]. This method (NOMAD with $2n$ neighbors) is a local derivative-free optimization algorithm which can handle constraints on the feasible domain. The second method considered is SMF (surrogate management framework; see [12]). This algorithm is implemented in [66] for airfoil design[10]. This method is a hybrid method that combines a generalized pattern search with a krigining based optimization algorithm.

The test problem considered here for the purpose of this comparison is the min-

[10]Though this code is not available online, we have obtained a copy of it by personal communication from Prof. Marsden.

imization of the 2D Rastrigin function (3.58), with $A = 3$, inside a feasible domain L characterized by the following constraints:

$$c_1(x) = x \leq 2, \quad c_2(x) = (x-2)^2 + (y+2)^2 \leq 8. \tag{3.66}$$

In this test, for Algorithm 3.2, the value of $y_0 = f(x^*) = 0$ was used, and the termination condition, similar to the previous sections, is taken by $\min_{x \in S^k} \|x - x_k\| \leq 0.01$. Results are shown in Figure 3.11.

Algorithm 3.2 performed 19 function evaluations (see Figure 3.11a), which includes some initial exploration of the feasible domain, and more dense function evaluations in the vicinity the global minimizer. Note that, since Algorithm 3.2 uses interpolation during its minimization process, once it discovers a point in the neighborhood of the global solution, convergence to the global minimum is achieved rapidly.

The SMF algorithm similarly performs both global exploration and local refinement. However, the number of function evaluations required for the same test problem increases to 66 (see Figure 3.11b). The reasons for this appear to be essentially twofold. First, the numerical performance of the polyharmonic spline interpolation together with our synthetic uncertainty function appear to behave favorably as compared with Kriging, as also observed in chapter 2 of this work. Second, the search step of the SMF [66] was not designed for nonlinearly constrained problems. In other words, the polling step is the only step of SMF that deals with the complexity of the constraints.

Unlike Algorithm 3.2, MADS is not designed for global optimization; thus, there is no guaranty of convergence to the global minimum. In this test, two different initial points were considered to illustrate the behavior of MADS. Starting from $x_0 = [1.9; -\pi]$ (see Figure 3.11c), MADS converged to a local minimizer after 19 function evaluations; however,

this local minimum is not the global solution. Starting from $x_0 = [1; -1]$ (see Figure 3.11d), MADS converged to the global minimizer at $[0, 0]$ after 59 function evaluations.

In the above tests, we have counted those function evaluations that are performed within L, as the constraint functions $c_i(x)$ are assumed to be computationally inexpensive. Note that Algorithm 3.2 does not include any function evaluations outside of L.

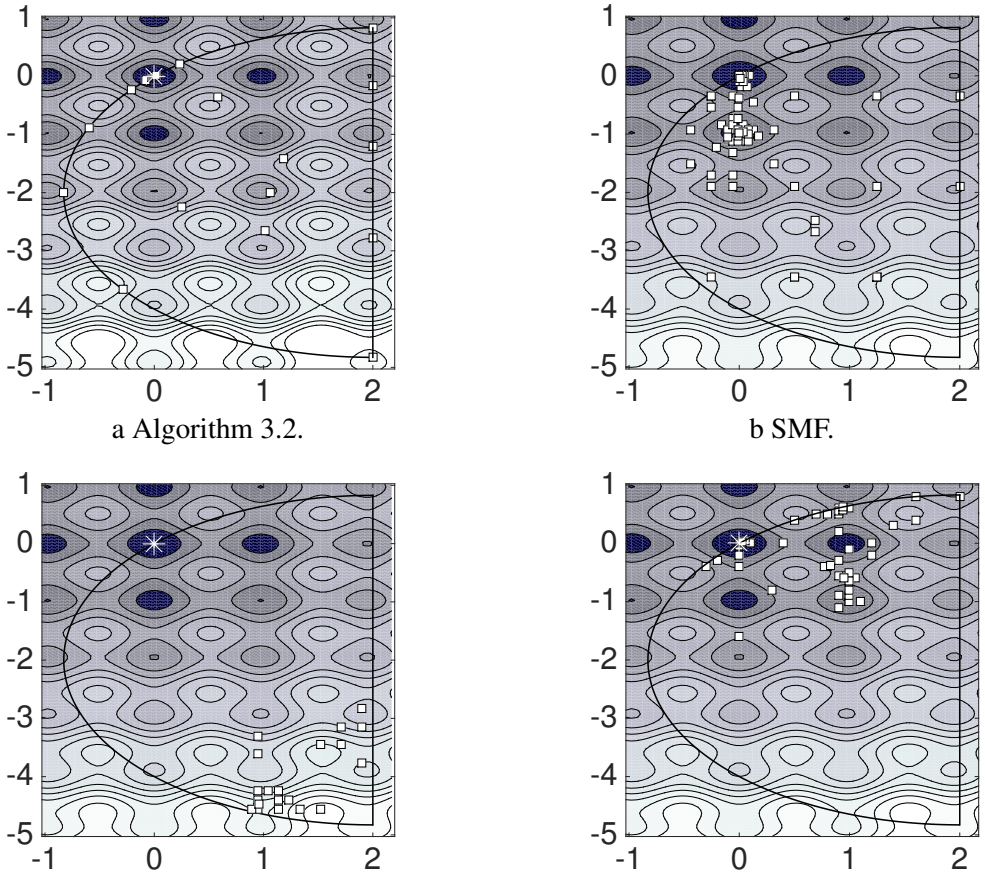
Although this particular test shows a significant advantage for Algorithm 3.2 compared to the SMF and MADS tests, more study is required to analyze properly the performance of this algorithm. An important fact about this algorithm is that it's performance is dependent on the interpolation strategy used; this dependence will be studied in future work. A key advantage of Algorithm 3.2 is that it lets the user choose the interpolation strategy to be implemented; the best choice might well be problem dependent.

3.6 Conclusions

In this chapter, the derivative-free global optimization algorithms developed in chapter 2 of this study were extended to the optimization of nonconvex functions within a feasible domain L bounded by a set of convex constraints. The chapter focused on extending Algorithm 4 of chapter 2 (the so-called Adaptive K algorithm) to convex domains; we noted in Remark 16 that the extension of Algorithm 2 of chapter 2 (the so-called Constant K algorithm) may be extended in an analogous fashion.

We developed three algorithms:

- Algorithm 3.1 showed how to initialize the optimization by identifying $n + 1$ points in L which maximize the volume of the initial simplex generated by these points. An enclosing simplex was also identified which includes L.
- The initial simplex identified by Algorithm 3.1, together with its body center with up



a Algorithm 3.2. b SMF.

c MADS start with $x_0 = [1.9; -\pi]$. d MADS start with $x_0 = [1; -1]$

Figure 3.11: Comparison of Δ-DOGS(C) with SMF and MADS. Implementation of (a) Algorithm 3.2, (b) SMF, and (c,d) MADS on minimization of the 2D Rastrigin function (3.58) inside the feasible domain characterized by (3.66).

to $n + 1$ additional points to better separate the interior simplices from the boundary simplices in the Delaunay triangulations to be used.

- Algorithm 3.2 then presented the optimization algorithm itself, which modified the Algorithms of chapter 2 in order to extend the convex hull of the evaluation points to cover the entire feasible domain as the iterations proceed. An important property of this algorithm is that it ensures that the triangulation remains well behaved (that is, the maximum circumradius remains bounded) as new datapoints are added.

In the algorithm developed, there are two adjustable parameters. The first parameter is y_0; this is an estimate of the global minimum, and has a similar effect on the optimization algorithm as seen in Algorithm 4 of chapter 2. The second parameter is $\kappa > 1$, which quantifies the size of the exterior simplex. It is shown in our analysis (§3.4) that, for $1 < \kappa < \infty$, the maximum circumradius is bounded. In practice, it is most efficient to choose this parameter as small as possible while respecting the technical condition (3.10) required in order to assure convergence.

The performance of the algorithm developed is shown in our results (§3.5) to be good if the aspect ratio of L of the feasible domain is not too large. Specifically, the curvature of each constraint $c_i(x)$ is seen to be as significant to the overall rate of convergence as the smoothness of the objective function $f(x)$ itself. It was suggested in §3.5 that, in feasible domains with large aspect ratios, those directions in parameter space that are highly constrained can be identified and eliminated from the global optimization problem; after the global optimization is complete, local refinement can be performed to optimize in these less important directions.

Another challenging aspect of the algorithm developed is its reduced rate of convergence when the global minimizer is near a corner of the feasible domain. This issue can be

addressed by switching to a local optimization method when it is detected that the global minimizer is near such a corner (that is, when multiple values of $c_i(x)$ are near zero).

In future work, we extend the algorithms developed to problems in which function evaluations are inexact, and can be improved with additional computational effort. Moreover, this algorithm will be applied on a wide range of test problems. In particular, the performance of the present algorithm depends on the particular interpolation strategy used (unlike various other response surface methods, this method is not limited to a specific type of interpolation strategy).

Acknowledgment

This chapter is published in Journal of Global Optimization as a different article as: P. Beyhaghi, T.R. Bewley, “Delaunay-based derivative-free optimization via global surrogates, part II: convex constraints", *Journal of Global Optimization*, (2016): 1-33. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Implementation of Cartesian grids to accelerate Delaunay-based derivative-free optimization

4.1 Introduction

In this chapter, a derivative-free optimization algorithm is presented to minimize a (possibly nonconvex) function subject to bound constraints [1]:

$$\text{minimize } f(x) \text{ with } \; x \in L = \{x | a \leq x \leq b\}, \;\; a < b. \tag{4.1}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $a, b \in \mathbb{R}^m$.

The algorithm developed here is a modification of the original Delaunay-based optimization derivative-free algorithm developed in chapter 2. In this chapter, we will assume

[1] Taking a and b as vectors, $a \leq b$ implies that $a_i \leq b_i \; \forall i$.

that there is a target value f_0 which is achievable ($\exists\, x \in L$ such that $f(x) \leq f_0$), and the goal is to find a point such that $f(x) \leq f_0$. As done in Algorithm 2 of [67], the present algorithm can easily be extended to problems for which a target value of f_0 is unavailable. For simplicity, in this chapter, we will assume that this target value f_0 is known.

The structure of this chapter is as follows: Section 4.2 presents the modified algorithm, and all the tools that are needed. Section 4.3 analyzes the convergence properties of the new optimization algorithm, and the technical conditions that are required to guarantee its convergence to the desired solution. Section 4.4 applies both the original and the modified optimization algorithm to a representative example. Some conclusions are presented in Section 4.5.

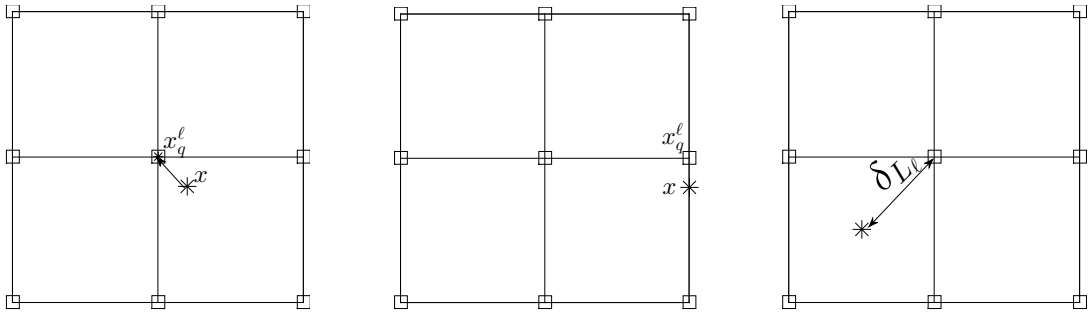
4.2 Delaunay-based optimization algorithm using cartesian grid

In this section, we present in Algorithm 4.1, a new optimization algorithm, dubbed in Δ-DOGS(Z), and its essential elements. Before explaining Algorithm 4.1, some preliminary concepts are required.

Definition 22. *The Cartesian grid of level ℓ for the feasible domain $L = \{x | a \leq x \leq b\}$, denoted by L_ℓ, is defined as follow:*

$$L_\ell = \left\{ x | x = a + \frac{1}{N}(b-a) \otimes z, \quad z \in \{0, 1, \ldots, N\} \right\}, \text{ where } N = 2^\ell.$$

*The **quantizater** of a point x on a grid L_ℓ, denoted x_q^ℓ, is a point on the grid which has the minimum distance from x. Note that this quantization process might have multiple*



a Quantization of an interior point

b Quantization of a boundary points

c Maximum quantization error of the grid

Figure 4.1: Representation of a 2D grid with level $\ell = 1$, the quantization process, and the the maximum quantization error. The gridpoints are shown by open squares in the figures. The point x is shown by the star in left and middle figures, and its quantization process is illustrated by an arrow. In middle figure, it is observed that the point x_q is on the boundary in which x_q is located.

solutions; in this case, any of these solutions is acceptable. The maximum quantization radius (i.e., in the language of sphere packing theory, the "covering radius") of the grid δ_{L_ℓ} is defined as follows:

$$\delta_N = \max_{x \in L_\ell} \|x - xq\| = \frac{\|b - a\|}{2N}. \tag{4.2}$$

Remark 21. *There are three important property for the the Cartesian grid which are used in our optimization algorithm.*

a. The grid of level ℓ covering the feasible domain L in an n dimensional space has $(N + 1)^n$ grid points.

b. $\lim_{\ell \to \infty} \delta_{L_\ell} = 0$.

c. Consider x_q as a quantization of x onto a L_ℓ, then $A_a(x) \subseteq A_a(x_q)$, where $A_a(x)$ is the set of active constraints at x_0. This point is illustrated in Fig. 4.1.

Definition 23. *Consider x as a point in L, and S as a nonempty set of points in L such that $z \in S$ is the closest point in S from x. The pair (x, S) is called **activated** if and only if*

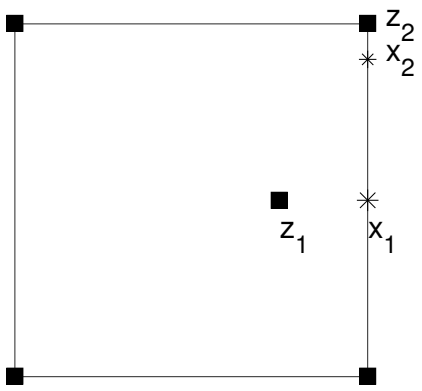


Figure 4.2: A
ctivated step illustration. The set S is shown by black squares. The point x_1 is not activated, since z_1 (the closest point in S to x_1) is not on the same boundary of L that x_1 lies. The point x_2 is activated, since z_2 (the closest point in S to x_2) is located on the same boundary of L that x_2 lies.

$A_a(x) \subseteq A_a(z)$, where $A_a(x)$ is the set of active constraints at x. (Note that the domain L has a total of $2n$ constraints.)

Remark 22. *If there are multiple points z which share the minimum distance from x in S, then the pair (x, S) is activated if, for all such z, $A_a(x) \subseteq A_a(z)$.*

Remark 23. *If x is an interior point in L, then the pair (x, S) is activated for any nonempty set S. However, if x is on the boundary of L, the pair (x, S) is activated or not based on the position of x and the points in S (see Fig. 4.2)*

Definition 24. *Consider S as a set of points in L which is partitioned into two subsets, $S = S_E \cup S_U$, as follows:*

- *The evaluated points are denoted by S_E, where the function values are available.*
- *The support points are denoted by S_U, where the function values are not available. Note that the support points will be helpful when developing the triangulation.*

Consider $p(x)$ as an interpolating function that passes through the points in S_E. The continuous and discrete search function are defined as follows:

$$\textit{Continuous Search function:} \quad s_c^k(x) = \begin{cases} \frac{p^k(x)-f_0}{e^k(x)}, & \textit{if } p^k(x) \geq f_0, \\ p^k(x) - f_0, & \textit{otherwise}, \end{cases} \tag{4.3}$$

$$\textit{Discrete Search function:} \quad s_d^k(x) = \begin{cases} \frac{p^k(x)-f_0}{Dis\{x,S_E\}}, & \textit{if } p^k(x) \geq f_0, \\ p^k(x) - f_0, & \textit{otherwise}, \end{cases} \tag{4.4}$$

where $e(x)$ is the uncertainty function constructed with all points in S, and $Dis\{x, S_E\} = \min_{z \in S_E} \|x - z\|$.

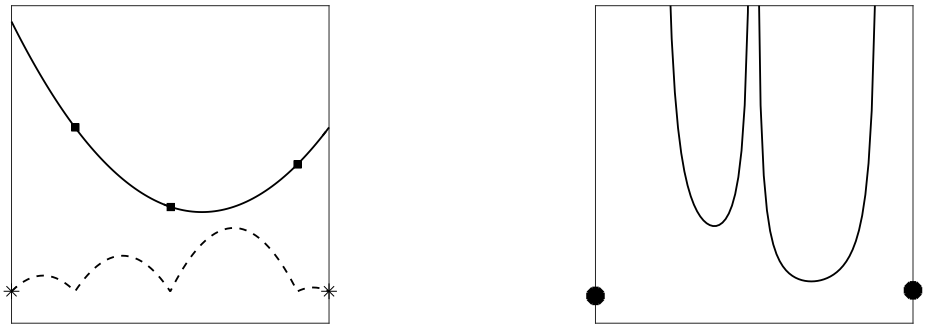
Remark 24. *Note that the continuous search function is similar to the search function defined in [67, 68]. An important difference to note, however, is that the interpolating function $p(x)$ and the uncertainty function $e(x)$ are defined based on two different set of points.*

Remark 25. *The value of the discrete search function is only defined at the points in S_U.*

Now we have all the tools necessary to present the modified optimization algorithm considered in this work. The following three key modifications to Δ-DOGS are performed to obtain Algorithm 4.1:

1. The datapoints in Algorithm 4.1 lie on the Cartesian grid, which is occasionally refined as the iteration proceeds.

2. At each iteration, two different sets of points exist, S_E and S_U. Function evaluations are available only for the points in S_E.



a $p(x)$, $e(x)$, S_E and S_U.

b $s_c(x)$ and $s_d(x)$.

Figure 4.3: Illustration of continuous and discrete search function. Figure (a) illustrates: (solid line) the interpolating function $p(x)$, (dashed line) the uncertainty function $e(x)$, (black square) evaluation points S_E, and (stars) support points S_U. Figure (b) illustrate: (sold line) the continuous search function $s_c^k(x)$ and (closed circles) the discrete search functions $s_d^k(x)$.

3. Two different search functions, $s_c(x)$ and $s_d(x)$, are considered at each iteration. One of them, $s_c(x)$ is minimized over the entire feasible domain L, and the other, $s_d(x)$, is minimized only over the points in S_U.

The complete algorithm is presented in Algorithm 4.1. It is observed that there are four possible cases at each iteration of this Algorithm:

a. In the first case, the pair (x_k, S) is not activated. This step is called an **inactivated step** which adds y_k to S_U^k, and no function evaluation is performed.

c. In the second case, the pair (x_k, S) is activated and $s_d^k(w_k) < s_d^k(x_k)$. This step is called an **activated replacing step** which removes z_k from S_U^k, and adds it to S_E^k, and calculates $f(w_k)$.

b. In the third case, the pair (x_k, S) is activated, $s_d^k(x_k) \leq s_d^k(w_k)$, and $y_k \notin S_E^k$. This step is called an **activated improving step** which adds a new point y_k is added to S_E^k, and calculates $f(x_k)$.

d. In the last case, the pair (x_k, S) is activated, $s_d^k(x_k) \leq s_d^k(w_k)$, and $y_k \notin S_E^k$. This step is a **mesh refinement step** which Note refinement L_l. Note that the evaluation point set and support points will not be changed at this step.

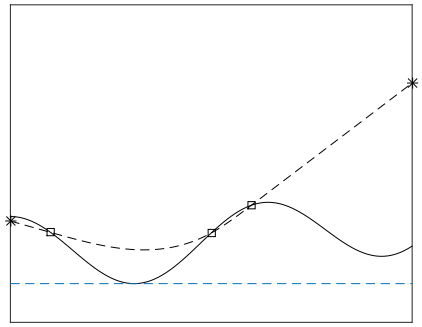
Remark 26. *The activated improving and replacing steps of Algorithm 4.1 are illustrated in Fig. 4.4. Note that, in 1D, all steps are activated.*

Algorithm 4.1 The modified Delaunay-Based optimization algorithm

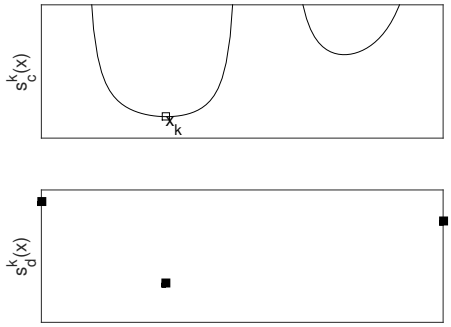
1: Set $k = 0$ and initialize ℓ. Take the initial set of support points S_U^0 as all 2^n vertices of the feasible domain L. Choose at least $n+1$ points on the initial grid which are affinely independent, put them in evaluation points set S_E^0, and calculate $f(x)$ at each of these points.
2: Calculate (or, for $k > 0$, update) an appropriate interpolating function $p^k(x)$ through all points in S_E^k.
3: Calculate (or, for $k > 0$, update) a Delaunay triangulation Δ^k over all of the points in $S^k = S_U^k \cup S_E^k$.
4: Find x_k as the minimizer of the continuos search function $s_c^k(x)$ (see Definition 24) in L, and y_k as its quantization on the grid L_ℓ. Whenever y_k or z_k is not unique, the algorithm selects one of the available options.
5: Find w_k as the minimizer of the discrete search function $s_d^k(x)$ (see Definition 24) in S_U^k.
6: **If** the pair (x_k, S^k) is not activated (see Definition 23), **then**, $S_U^{k+1} = S_U^k \cup \{y_k\}$, increment k, and repeat from 2.
7: **Elseif** $s_d^k(w_k) \leq s_d^k(x_k)$, **then** calculate $f(w_k)$. $S_E^{k+1} = S_E^k \cup \{w_k\}$, $S_U^{k+1} = S_U^k - \{w_k\}$, and increment k. **If** $f(w_k) > f_0$, **then** repeat from 2; **otherwise**, stop the algorithm.
8: **Elseif** $y_k \notin S_E^k$, **then** calculate $f(y_k)$, $S_E^{k+1} = S_E^k \cup \{y_k\}$, increment k. **If** $f(y_k) > f_0$, **then** repeat from 2; **otherwise**, stop the algorithm.
9: **Else** increment both ℓ and k, and repeat from 2.
10: **Endif**

4.2.1 Restriction of the decrease of the interpolating function

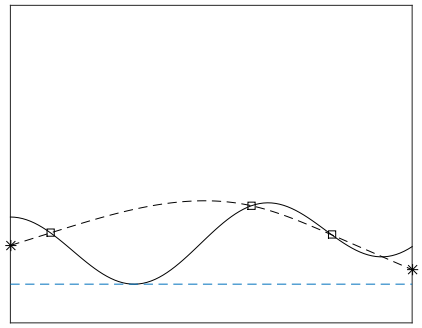
In this subsection, we will describe a small modification of Algorithm 4.1 which improves its convergence. At each step, the functions $s_c^k(x)$ and $s_d^k(x)$ have to be minimized. Note that if $s_c^k(x_k) < 0$, or $s_d^k(x_d^k) < 0$, then there is a point in L for which $p(x) < f_0$. Since x_k is taken as the minimizer of $s_c^k(x)$, then $p^k(x_k) < f_0$.



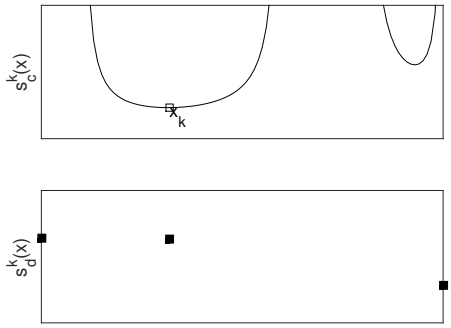
a $p^k(x)$ and $f(x)$ for improving step.



b $s_k(x)$ and $s_d^k(x)$ for improving step.



c $p^k(x)$ and $f(x)$ for replacing step.



d $s_k(x)$ and $s_d^k(x)$ for replacing step.

Figure 4.4: Illustration of improving (first two) and replacing step(second row) of Algorithm 4.1. Left figures: (dashed line) the interpolating function $p^k(x)$, (solid line) the objective function $f(x)$, (open squares) evaluated points S_E^k (starts) support points S_U^k by dashed line, solid line, square and star markers respectively. Right figures: (solid line) the continuous search function $s_c^k(x)$, (closed squares) the discrete search function $s_d^k(x)$, (open square) the global minimizer of the continuous search function x_k.

Definition 25. *Those steps of Algorithm 4.1 for which $p^k(x_k) \leq f_0$, which is equivalent to* $\min\{s_c^k(x_k), s_d^k(x_d^k)\} < 0$*, are called* ***extreme decreasing*** *steps.*

At each extreme decreasing step of Algorithm 4.1, if $p^k(x)$ is substantially less than f_0, then the value of the interpolation may not be reliable. This situation can happen only when x_k is far from the available datapoints. In this case, we will find a point in which $p^k(x) = f_0$ and which is close to the existing datapoints, and evaluate the function there instead. This approach is akin to the trust region approach, is more promising, as it is based on a more reliable value for the interplant.

To accomplish this, consider x_m as a point in S_E^k which has the minimum objective function value. By construction, $p^k(x_m) > f_0$ (otherwise, the algorithm is already terminated). Since step k is considered to be extreme decreasing, $p^k(x_k) < f_0$. Thus, there is a point $\bar{x}$ (not necessary unique) on the segment between x_m and x_k such that $p^k(\bar{x}) = f_0$. Finding $\bar{x}$ (see Fig. 4.5) is a simple one-dimensional root finding problem for the computationally inexpensive function $p^k(x) - f_0$. We use the false position method to find $\bar{x}$. Then, the point x_k is replaced by $\bar{x}$.

In summary, for each step of Algorithm 4.1, there are two possible situations:

a. Step k is not extreme decreasing, and thus,

$$s_c^k(x) > 0 \; \forall x \in L, \text{ and } \quad s_d^k(x) > 0 \; \forall x \in S_U^k.$$

b. Step k is extreme decreasing, and after the above modification, $p(x_k) = f_0$. In this case, we simply remove y_k from S_U^k, if it is already there, and add y_k to S_E^k (and calculate $f(y_k)$) if it is not already there. If y_k is already in S_E^k, we refine the mesh.

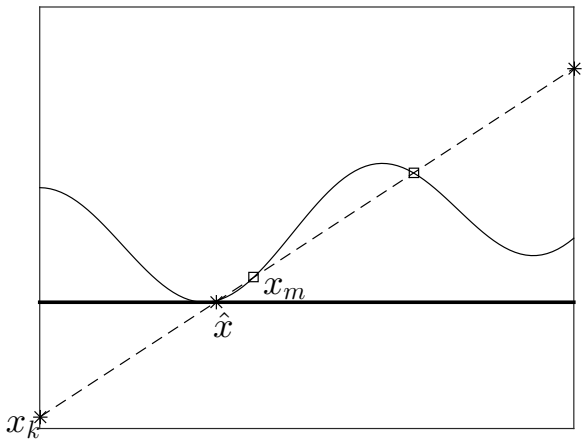


Figure 4.5: Illustration of a extreme decreasing step of Algorithm 4.1: (dashed line) the interpolating function $p^k(x)$, (solid line) the objective function $f(x)$, (open squares) evaluated points S_E^k (starts) support points S_U^k by dashed line, solid line, square and star markers respectively. The solid horizontal line indicates the target value of f_0.

4.3 Analysis of the Algorithm

In this section, we analyze the convergence properties of Algorithm 4.1. If the algorithm terminates after finite number of steps k, then a point x_k is found for which the function value is less or equal to the target value f_0; otherwise, all computed values of the objective function are greater than the target value. In this section, we will show, in the latter case, that a limit point of the datapoints that are obtained in the evaluation set S_E includes a feasible point whose objective function is equal to the target value. Therefore, for this analysis, we will assume that Algorithm 4.1 proceeds for infinite number of iterations.

Before analyzing the convergence property of Algorithm 4.1, we will show that Algorithm 4.1 includes an infinite number of mesh refinements. To show this, a preliminary lemma is first established.

Lemma 18. *Consider k as an inactivated step of Algorithm 4.1; then,* $y_k \notin S^k$.

Proof. We establish this lemma by contradiction. Assume that $y_k \in S^k$, and step k is inactivated; therefore, there is a point $z_k \in S^k$ that has minimum distance from x_k, and

$A_a(x_k) \nsubseteq A_x(z_k)$. Since all point in S^k are on the grid L_ℓ of step k, and y_k is a quantizer of x_k on this grid, $\|x_k - y_k\| \leq \|x_k - z_k\|$. On the other hand, since $y_k \in S^k$ and z_k is the closest point to x_k in S^k, then $\|x_k - y_k\| \geq \|x_k - z_k\|$. This leads to $\|x_k - z_k\| = \|x_k - y_k\|$. As a result; the point z_k is also a quantizer of x_k which is in contradiction with $A_a(x_k) \nsubseteq A_a(z_k)$. □

Theorem 9. *There are an infinite number of mesh refinement steps if Algorithm 4.1 proceeds without terminating.*

Proof. This theorem is also estabshied by contradiction. Assume that there are a finite number of mesh refinement steps as Algorithm 4.1 proceeds, then all datapoints must lie on a grid with some level ℓ. At each step of Algorithm 4.1, if it is activated improving, then $|S_E^k|$ and $|S^k|$ are both incremented by one. If it is activated replacing, then $|S_E^k|$ is incremented by one and $|S^k|$ is fixed. if it is inactivated, then $|S_E^k|$ is fixed and $|S_E^k|$ is incremented by one. Therefore, at each step of the algorithm which is not mesh refinement, we will increment the value of $|S^k| + |S_E^k|$ by at least one. Since the number of points on the grid of level ℓ is finite, we must have only finite number of iterations which are not mesh refinements, which is in contradiction with the fact that there are infinite number of steps for Algorithm 4.1. □

We now analyze the convergence of Algorithm 4.1. To do this, the following conditions are imposed for the objective and interpolating functions.

Assumption 7. *The interpolating functions $p^k(x)$, objective function $f(x)$, and $p^k(x) - f(x)$ are Lipschitz with the same Lipschitz constant $\hat{L}$.*

Assumption 8. *A constant $\hat{K} > 0$ exists in which*

$$\nabla^2\{f(x) - p^k(x)\} + 2\,\hat{K}\,I > 0, \quad \forall x \in L \text{ and } k > 0, \tag{4.5}$$

$$\nabla^2\{p^k(x)\} - 2\,\hat{K}\,I < 0, \quad \forall x \in L \text{ and } k > 0, \tag{4.6}$$

$$\nabla^2\{f(x)\} - 2\,\hat{K}\,I < 0, \ \forall x \in L. \tag{4.7}$$

Before analyzing the convergence properties of Algorithm 4.1.

Lemma 19. *Consider $G(x)$ as a twice differentiable function such that $\nabla^2 G(x) - 2\,K_1 I \leq 0$, and $x^* \in L$ as a local minimizer of $G(x)$ in L. Then, for each $x \in L$ such that $A_a(x^*) \subseteq A_a(x)$, we have:*

$$G(x) - G(x^*) \leq K_1 \|x - x^*\|^2. \tag{4.8}$$

Proof. Define function $G_1(x) = G(x) - K_1\,\|x - x^*\|^2$. By constrction, $G_1(x)$ is concave; therefore,

$$G_1(x) \leq G_1(x^*) + \nabla G_1(x^*)^T (x - x^*),$$

$$G_1(x^*) = G(x^*), \quad \nabla G_1(x^*) = \nabla G(x^*),$$

$$G(x) \leq G(x^*) + \nabla G(x^*)^T (x - x^*) + K_1\,\|x - x^*\|^2.$$

Since the feasible domain is a bounded domain, the constrained qualification holds; therefore, x^* is a KKT point. Therefore, using $A_a(x^*) \subseteq A_a(x)$ leads to $\nabla G(x^*)^T (x - x^*) = 0$, which verifies (4.8).

□

Lemma 20. *Consider k as a step of Algorithm 4.1 which is activated and extreme decreas-*

ing. Then

$$p^k(z_k) - f_0 \leq 2\{K + \hat{K}\}\|x_k - z_k\|^2, \tag{4.9}$$

where $K = s_c^k(x_k) > 0$.

Proof. Since x_k is a global minimizer of $s_c^k(x) = \frac{p^k(x)-f_0}{e^k(x)}$, and $s_c^k(x) \geq 0$ for all $x \in L$, then x_k is a global minimizer of $T^k(x) = p^k(x_k) - K\, e_k(x)$ too, and $T^k(x_k) = f_0$ (see §5 in [67] for discussion of why). Consider $\Delta_i^k \in \Delta^k$ as the simplex which includes x_k. By construction, $e^k(x_k) = e_i^k(x_k)$. Define $T_i^k(x) = p^k(x) - K\, e_i^k(x)$, then $T_i^k(x)$ is a twice differentiable function in L, and

$$\nabla^2 T_i^k(x) = \nabla^2\{p^k(x)\} + 2\,KI, \quad \nabla^2 T_i^k(x) - 2\{\hat{K} + K\}I \leq 0.$$

By Lemma (2), $e_i^k(z_k) \leq e^k(z_k)$, which leads to $T^k(x) \leq T_i^k(x)$ for all points $x \in L$, x_k is a global minimizer of $T^k(x)$, and $T^k(x_k) = T_i^k(x_k)$. Therefore, x_k is a global minimizer of $T_i^k(x)$ as well.

Since step k is activated, $A_a(x_k) \subseteq A_a(z_k)$; thus, using Lemma 19, we have:

$$T_i^k(z_k) - T_i^k(x_k) \leq 2\{K + \hat{K}\}\|z_k - x_k\|^2.$$

$$T^k(z_k) \leq T_i^k(z_k), \quad T^k(x_k) = T_i^k(x_k),$$

$$T^k(z_k) - T^k(x_k) \leq 2\{K + \hat{K}\}\|z_k - x_k\|^2,$$

$$T^k(z_k) - f_0 \leq 2\{K + \hat{K}\}\|z_k - x_k\|^2.$$

Since $z_k \in S_E^k$, $e^k(z_k) = 0$ and $p^k(z_k)$, which leads to $T^k(z_k) = p^k(z_k)$ which shows (4.9). □

Lemma 21. *Consider* x^* *as a global minimizer of* $f(x)$ *in L. Then, for each step of Algo-*

rithm 4.1 which is not extreme decreasing, we have:

$$\min\{\frac{s_c^k(x^*)}{\hat{K}}, \min_{z\in S_U^k}\{\frac{s_d^k(z)}{\hat{L}}\}\} \leq 2. \tag{4.10}$$

Proof. Consider Δ_i^k as a simplex in Δ^k which includes x^* whose vertices are $\{V_1^k, V_2^k, \ldots, V_{n+1}^k\}$. Define $L^k(x)$ as the unique linear function in Δ_i^k such that $L(V_i^k) = 2\,f(V_i^k) - p^k(V_i^k)$, and define $G^k(x) = p^k(x) + L^k(x) - 2\,\hat{K}e^k(x) - 2\,f(x)$. Then, for each vertex $V_i^k(x)$ of Δ_i^k,

$$G^k(V_i) = p^k(V_i) + L^k(V_i^k) - 2\hat{K}e^k(V_i) - 2\,f(V_i^k) = 0.$$

Moreover, since $\nabla^2 L_k(x) = 0$, and $\nabla^2 e^k(x) = -2\,I$ inside the simplex Δ_i^k, then according to Assumption 10, $\nabla^2 G^k(x) \geq 0$. Thus, $G^k(x)$ is convex in Δ_i^k, and its maximum is located at one of its vertices; therefore,

$$G^k(x^*) \leq 0, \quad p^k(x^*) + L^k(x^*) - 2\,f(x^*) - 2\,\hat{K}e^k(x^*) \leq 0.$$

Since f_0 is assumed to be achievable, $f_0 \geq f(x^*)$, and thus

$$p^k(x^*) + L^k(x^*) - 2\,f_0 - 2\,\hat{K}e^k(x^*) \leq 0.$$

Since $x^* \in \Delta_i^k$ and $L^k(x)$ is linear, it follows that

$$\min_{1\le j\le n+1} L^k(V_j^k) \le L^k(x^*),$$

$$\min_{1\le j\le n+1}\{2\,f(V_j^k) - p^k(V_j^k)\} = \min_{1\le j\le n+1} L^k(V_j^k),$$

$$\min_{z\in S^k}\{2\,f(z) - p^k(z)\} \le \min_{1\le j\le n+1}\{2\,f(V_j^k) - p^k(V_j^k)\},$$

$$p^k(x^*) - f_0 - 2\,\hat{K}e^k(x^*) + \min_{z\in S^k}\{2\,f(z) - p^k(z) - f_0\} \le 0. \tag{4.11}$$

Define $\hat{z}$ as the closest point to z in S_E^k. By construction, $p^k(\hat{z}) - f(\hat{z}) = 0$. Furthermore, by Assumption 9, the function $p^k(x) - f(x)$ is Lipschitz with constant $\hat{L}$, and thus

$$p^k(z) - f(z) \le \hat{L}\|z - \hat{z}\| = \hat{L}\,\mathrm{Dis}(z, S_E^k),$$

$$p^k(z) - 2\,\hat{L}\,\mathrm{Dis}(z, S_E^k) \le 2\,f(z) - p^k(z). \tag{4.12}$$

Using (4.11) and (4.12) leads to:

$$p^k(x^*) - f_0 - 2\,\hat{K}e^k(x^*) + \min_{z\in S^k}\{p^k(z) - f_0 - 2\,\hat{L}\mathrm{Dis}(z, S_E^k)\} \le 0. \tag{4.13}$$

Since step k is not a mesh refinement, $p^k(x) - f_0 > 0$ for all $x \in L$. Thus, $\frac{1}{s_d^k(x)}$ and $\frac{1}{s_c^k(x)}$ are well defined functions everywhere in L, and equation (4.13) can be rewritten as:

$$(p^k(x^*) - f_0)(1 - \frac{2\,\hat{K}}{s_d^k(x^*)}) + \min_{z\in S^k}\{(p^k(z) - f_0)(1 - \frac{2\,\hat{L}}{s_d^k(z)})\} \le 0 \tag{4.14}$$

Since $p^k(x^*) - f_0 > 0$ and $p^k(z) - f_0 > 0 \quad \forall z \in S^k$, (4.10) is verified. □

Lemma 22. *Consider k as a mesh refinement step of Algorithm* (4.1) *which is not extreme*

decreasing. Then

$$\min_{z \in S_E^k} f(z) - f_0 \leq \max\{3\,\hat{L}\,\delta_k, 6\,\hat{K}\,\delta_k^2\}, \tag{4.15}$$

where δ_k is the maximum discretization error of the Cartesian grid L_ℓ at this step.

Proof. Since step k is mesh refinement, by construction, $s_d^k(x_k) \leq s_d^k(w_k)$. Additionally, x_k and w_k are the global minimizer of $s_c^k(x)$ and $s_d^k(x)$ in L and S_U^k respectively. Thus, by using (4.10) in Lemma 21, we have:

$$\min\{\frac{s_c^k(x_k)}{\hat{K}}, \frac{s_d^k(x_k)}{\hat{L}}\} \leq 2. \tag{4.16}$$

There are two possible cases: In the first case, $s_c^k(x_k) \leq 2\hat{K}$; thus, using Lemma 20, we have:

$$p^k(y_k) - f_0 \leq 2\,[2\hat{K} + \hat{K}]\|x_k - y_k\|^2 = 6\,\hat{K}\|y_k - x_k\|^2.$$

Since $y_k \in S_E^k$, $f(y_k) = p^k(y_k)$. Furthermore, $\|x_k - y_k\| \leq \delta_k$; thus, (4.15) is verified in this case. In the second case, $s_d^k(x_k) \leq 2\hat{L}$. Since $y_k \in S_E^k$, and all points in S_E^k are on the grid L_ℓ, it follows that $\text{Dis}(x_k, S_E^k) = \|x_k - y_k\| = \delta_k$, and thus

$$p^k(x_k) - f_0 \leq 2\,\hat{L}\|x_k - y_k\|, \quad p^k(y_k) - p^k(x_k) \leq \hat{L}\|x_k - y_k\|,$$
$$f(y_k) - f_0 \leq 3\,\hat{L}\|x_k - y_k\| \leq 3\,\hat{L}\delta_k.$$

Thus, (4.15) is shown for both cases. □

Remark 27. *If step k of Algorithm 4.1 is a mesh refinement and extreme decreasing, then $p^k(x_k) = f_0$. Additionally, $p^k(x)$ is Lipschitz with constant $\hat{L}$; therefore,*

$$p^k(y_k) - f_0 \leq \hat{L}\|x_k - y_k\| \leq \hat{L}\delta_k. \tag{4.17}$$

Moreover, $y_k \in S_E^k$*, then*

$$f(y_k) - f_0 \leq \hat{L}\|x_k - y_k\| \leq \hat{L}\delta_k. \tag{4.18}$$

Theorem 10. *If Algorithm 4.1 is not terminated at any step, then the set* $S^\infty = \lim_{k\to\infty} S^k$ *has a limit point, denoted* $v \in L$*, such that* $f(v) = f_0$*.*

Proof. According to Theorem 9, there is an infinite number of mesh refinement steps during the execution of Algorithm 4.1, denoted here $\{k_1, k_2, \dots\}$. Define $v_i = argmin_{z\in S^{k_i}} f(z)$. According to Lemma 22 and Remark 27, we have:

$$f(v_i) - f_0 \leq \max\{3\,\hat{L}\,\delta_{k_i}, 6\,\hat{K}\,\delta_{k_i}^2\}, \tag{4.19}$$

Since Algorithm 4.1 is not terminated at any step, $f(v_i) - f_0 \geq 0$. Additionally, $\lim_{i\to\infty} \delta_{k_i} = 0$, which leads to $\lim_{i\to\infty} f(v_i) = f_0$. □

4.4 Results

In this section, we will check the performance of Algorithm 4.1 and compare it with the original Δ-DOGS, Algorithm 2 in [67], as summarized in Δ-DOGS. The test function which is considered is the n-dimensional Styblinski Tang function, which is a benchmarking test problem for global optimization:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^4 - 16\,x_i^2 + 5\,x_i}{2} - 39.16616\,n, \quad \text{where} \quad L = \{x| -5 \leq x_i \leq 5\}. \tag{4.20}$$

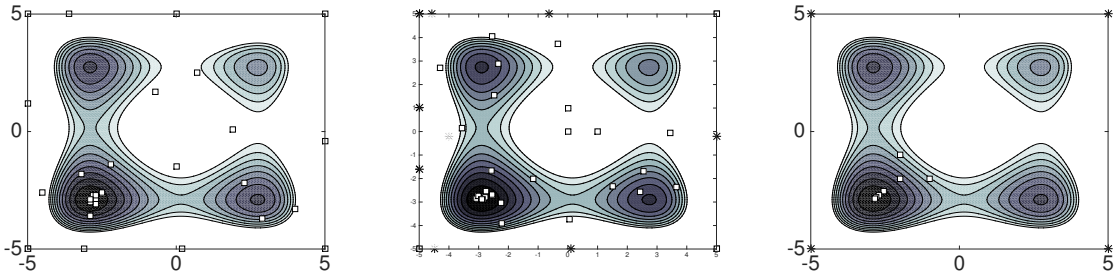
An initial grid level of $\ell_0 = 3$ is considered, and the algorithm continuous until the grid level of $\ell = 8$ is terminated. Note that Δ-DOGS is terminated when $\text{Dis}(x_k, S^k) \leq 0.05$, which leads to a comparable order of accuracy for both methods (i.e. the maximum quantization

error of level 8 is close to 0.05). The initial datapoints in S_E^0 are constructed by $n+1$ points as follows:

$$S_E^0 = \left\{ x_0, x_0 + \frac{b_i - a_i}{2^{\ell_0}} e_i, \forall i \in \{1, 2, \ldots, n\} \right\}. \tag{4.21}$$

For each i, e_i is one of the main coordinate directions, and x_0 is an initial point on the grid of level ℓ_0. In this section, we have considered two different choices of x_0 for the initialization of Algorithm 4.1.

The position of the datapoints that are used during the optimization process are illustrated in Fig. 4.6 for the n=2 dimensional implementation. It is observed that, for a bad initial point for Algorithm 4.1, the convergence to the global minimum is achieved with similar number of function evaluation as with Δ-DOGS. For good initial point, Algorithm 4.1 converges much faster. There is a noticeable differences in their result reported which gives Algorithm 4.1 a significant advantage for higher-dimensional problem (see Fig. 4.7 and Table 4.1). Specially, much less accumulation of datapoints on the boundary of the feasible domain is observed. This is due to the fact that the boundary points that are needed for Δ-DOGS are simply needed to regularize the triangulation. Algorithm 4.1 avoids this need by dividing S^k into the evaluated points S_E^k and support points S_U^k, and thus explores the interior of feasible domain more extensively which instead explores the interior of the feasible domain more extensively. Therefore, a significant improvement of Algorithm 4.1 over Δ-DOGZ in the speed of convergence compare is seen in Table 4.1), with this improvement is especially pronounced as n is increased.

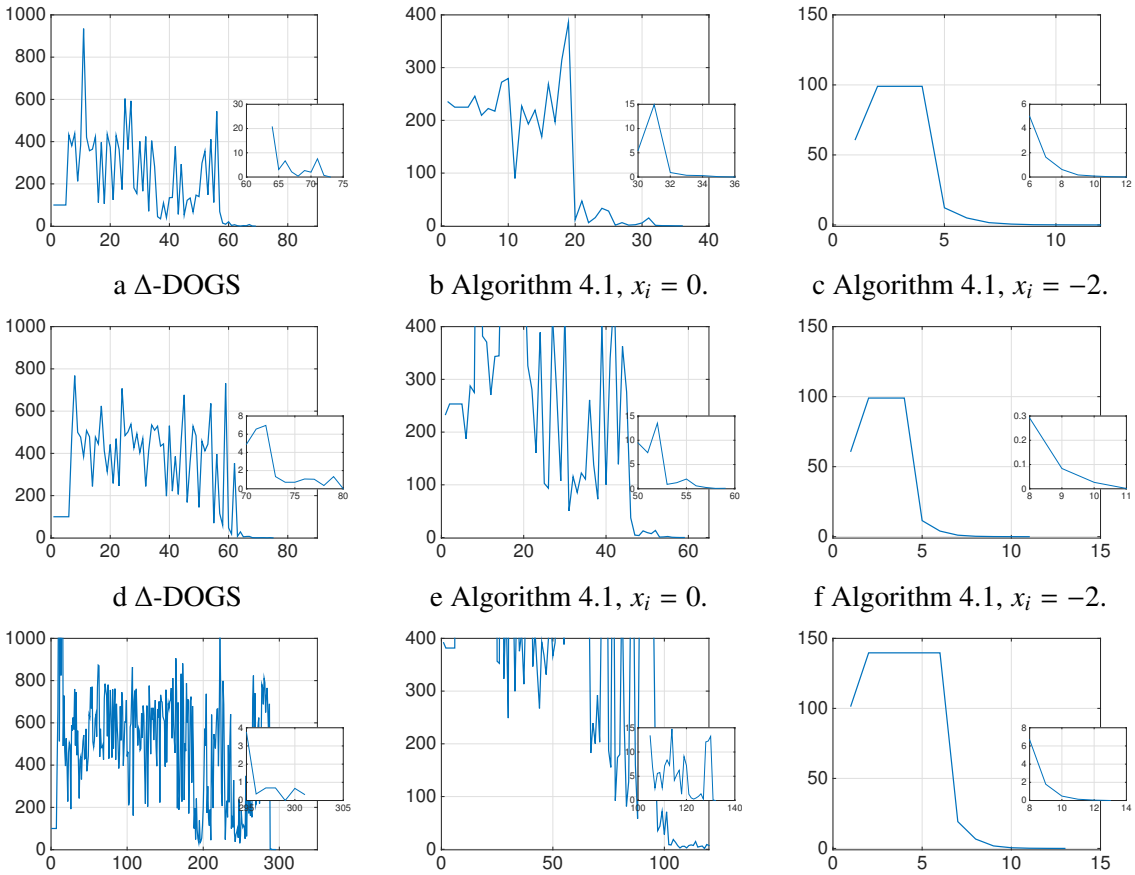


a Δ-DOGS
number of Fun. evals: 29

b Algorithm 4.1, $x_i = 0$.
number of Fun. evals: 31

c Algorithm 4.1, $x_i = -2$.
number of Fun. evals: 11

Figure 4.6: Implementation of Δ-DOGS and Algorithm 4.1 (with two different initial points) on problem (4.20) in 2D: (open square) evaluated points, (stars) support points.



a Δ-DOGS

b Algorithm 4.1, $x_i = 0$.

c Algorithm 4.1, $x_i = -2$.

d Δ-DOGS

e Algorithm 4.1, $x_i = 0$.

f Algorithm 4.1, $x_i = -2$.

g Δ-DOGS

h Algorithm 4.1, $x_i = 0$.

i Algorithm 4.1, $x_i = -2$.

Figure 4.7: The convergence history of Δ-DOGS and Algorithm 4.1 on optimization problem (4.20) (first row) 3, (second row) 4, and (third row) 5 dimensional problems.

Table 4.1: The summary of implementation of Algorithm 4.1 and Δ-DOGS on problems (4.20) in n = 2, 3, 4 and 5 dimensions.

Dimension	Algorithm	initial point	fn. evals	# of support points
2	Δ-DOGS	N/A	29	N/A
	Algorithm 4.1	$x_i = 0$	31	6
		$x_i = -2$	11	8
3	Δ-DOGS	N/A	50	N/A
	Algorithm 4.1	$x_i = 0$	36	32
		$x_i = -2$	11	8
4	Δ-DOGS	N/A	76	N/A
	Algorithm 4.1	$x_i = 0$	59	69
		$x_i = -2$	12	16
5	Δ-DOGS	N/A	295	N/A
	Algorithm 4.1	$x_i = 0$	132	112
		$x_i = -2$	13	32

4.5 Conclusions

In this chapter, we have modified the original Delaunay-based derivative-free optimization algorithm Δ-DOGS, proposed in [67], in order to accumulate fewer points on the boundary feasibility, and instead exploring the interior of the feasible domain more extensively. The Algorithm 4.1 has three main modifications as compared with the original algorithm:

- Two different sets of points are considered during the optimization procession: evaluation points and support points. The latter set helps to regulate the triangulation developed.
- Since the uncertainty function is zero at some points which are not in the evaluation set, another metric for the search function at these points.
- The datapoints that are used in the Algorithm 4.1 lie on a Cartesian grid.

Similar to the original algorithm, and any other derivative-free optimization algorithm, there is a curse of dimensionality, and optimization in only moderate dimensional problems (i.e., $n \lesssim 10$) is expected to be numerically tractable. A key bottleneck of the present code as the dimension of the problem is increased is the overhead associated with the enumeration of the triangulation. One of the limitations of the algorithm presented here is its restriction to bound constrained domains. Note that the original Delaunay-based optimization algorithm developed in [67] can handle any linearly-constrained domains. Another potential weakness is that the Cartesian grid present used is not the best option for the discretization (see [13]). In future work, the algorithm developed here will be modified to deal with general linearly-constrained domains, and different lattices will be considered. In later studies, more benchmarking test problems and application-based optimization problems will also be considered.

Acknowledgment

This chapter is submitted to Journal of Global optimization as the following article: "P. Beyhaghi, T.R. Bewley, ‘Implementation of Cartesian grids to accelerate Delaunay-based derivative-free optimization." The dissertation author was the primary investigator and author of this paper.

Chapter 5

A derivative-free optimization for efficiently minimizing the infinite time-averaged statistics

5.1 Introduction

In this chapter, the Delaunay-based derivative-free optimization algorithm developed in previous chapters is modified to minimize an objective function, $f(x) : \mathbb{R}^n \to \mathbb{R}$, obtained by taking the infinite-time average of a discrete-time ergodic process $g(x, k)$ for $k = 1, 2, 3, \ldots$ such that

$$f(x) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} g(x, k), \tag{5.1a}$$

where, for $k \gtrsim \bar{k}$, $g(x, k)$ is assumed to be statistically stationary. The feasible domain in which the optimal design parameter vector $x \in \mathbb{R}^n$ is sought is a bound constrained domain

$$L = \{x | a \leq x \leq b\} \quad \text{where } a < b \in \mathbb{R}^n. \tag{5.1b}$$

Note that calculating the true value of $f(x)$ for any x is impossible; $f(x)$ can only be approximated as the average of $g(x,k)$ over some *finite* number of samples N. The truth function $f(x)$ is typically a smooth function of x, though it is often nonconvex; computable approximations of $f(x)$, however, are generally *nonsmooth* in x, as the truncation error (associated with the fact any approximation of $f(x)$ must be computed with finite N) is effectively decorrelated from one approximation of $f(x)$ to the next.

Minimizing (5.1a) within the feasible domain (5.1b) is the subject of interest in host of practical applications, such as the optimization of stiffness and shape parameters and feedback control gains in mechanical systems and manufacturing processes involving turbulent flows, the setting of airline ticket prices, etc. Remarkably, however, there are very few optimization algorithms today ([69] and [70] being notable exceptions) that specifically address problems of this form, which instead are typically handled using classical derivative-free optimization approaches which, quite inefficiently, approximate each function evaluation with the same amount of sampling. Though many such classical derivative-free optimization approaches effectively keep function evaluations far apart in parameter space until convergence is approached, thereby mitigating somewhat the effect of uncertainty in the function evaluations, at least for a while, they are not specifically designed to adjust the amount of sampling associated with each individual function evaluation, making function evalutations more accurate (and, thus, more expensive), as required, as convergence is approached. The algorithm presented here, in contrast, automatically adjusts the sampling associated with each function evaluation as convergence is approached. Three existing classes of algorithms that might be considered to optimize problems of the form given in (5.1) are discussed further below.

The first is based on a convex production planning model [71], using the dynamic

programming principle to find an optimal solution. This modeling has been analyzed in both the discrete [72] and continuous [73, 74] settings. This approach is appropriate for control problems in which an adaptive controller is implemented with a large number of design parameters.

The second is based on an adjoint formulation (see, e.g., [75, 76, 77, 78]), which calculates the gradient of the function of interest at each iteration based on an adjoint computation. This method is very powerful for many problems in which essentially exact function evaluations are available, and scales exceptionally well to problems with many design parameters. However, for problems of the form (5.1), such as the optimization of airfoil shapes for turbulent flows, classical adjoint formulations are not suitable. This issue is addressed in a clever way in [79] and [80], where a novel method to alleviate it is presented. Nevertheless, this approach still only assures local convergence, and is highly sensitive to the accuracy of the mathematical model which is used for adjoint-based computations of the gradient.

The third is the class of derivative-free methods, which are indeed the most promising class of approaches for problems of the present form. These methods are also implemented for shape optimization in airfoil design [66], as well as in online optimization [69]. With such methods, only values of the function evaluations themselves are used, and neither a derivative nor its estimate is needed. The best methods of this class strive to keep function evaluations far apart in parameter space until convergence is approached, thereby mitigating somewhat the effect of uncertainty in the function evaluations. This class of methods generally handles feasibility boundaries quite well, and may be used to globally minimize the function of interest. However, this class of method scales quite poorly with dimension of the problem. The surrogate management framework [12] and Bayesian algo-

rithms [81, 11, 7, 70] are amongst the best derivative-free methods available today, and are implemented for minimizing a problem of the form in (5.1) in [66, 82, 83, 84].

For problems of the form (5.1), [69] and [70] are perhaps the two most closely related chapters to the present in the literature. In [69], a clever Lipschitzian optimization algorithm is presented which uses measurements with differing amounts of accuracy; this approach builds specifically upon accurate knowledge of the Lipchitz norm of the truth function. Proof of convergence of the algorithm proposed is provided in [69]. In [70], a promising Kriging-based algorithm of the surrogate management framework family is proposed, in a manner which increases the sampling of new measurements as convergence is approached. However, this method does not selectively refine existing measurements, which is a key contributor to the efficiency of the algorithm developed herein. Also, rigorous proof of convergence of the algorithm proposed in [70] is unavailable.

In this chapter, a highly efficient and provably convergent new optimization approach is developed for problems of the form given in (5.1). The structure of the remainder of the chapter is as follows: Section 5.2 lays out all of the new elements that compose the new optimization approach, as well as the new algorithm itself, dubbed α-DOGS. Section 5.4 analyzes the convergence properties of the new algorithm, and establishes conditions which are sufficient to guaranty its convergence to the global minimum. Section 7.4 applies the new algorithm to a selection of model problems in order to illustrate its behavior. Some conclusions are presented in Section 5.6.

5.2 Description of Algorithm:

This section presents the essential elements of the new optimization algorithm, dubbed α-DOGS, which is designed to efficiently minimize a function $f(x)$ given by (5.1a)

within the feasible domain L defined by (5.1b). We begin by introducing some fundamental concepts.

Definition 26. *Take S as a set of points x_i, for $i = 1, \ldots, M$, at which the function $f(x)$ in* (5.1a) *has been approximated; drawing a parallel to the nomenclature commonly used in estimation theory, we refer to any such approximation of $f(x)$, developed with a finite number of samples N_i, as a measurement, denoted y_i:*

$$y_i = y(x_i, N_i) = \frac{1}{N_i} \sum_{k=1}^{N_i} g(x_i, k). \tag{5.2}$$

Any such measurement y_i has a finite uncertainty associated with it, which may made small by increasing N_i.

Remark 28. *For many problems, there is an initial transient such that, for $k < \bar{k}$, the assumption of stationarity of $g(x, t_k)$ is not valid. For such problems, the initial transient in the data can be detected using the approach developed in [85] and set aside, and the signal considered as stationary therafter. In such problems, to increase the speed of convergence of the statistics, the finite sum used for averaging the samples in* (5.2) *is modified to begin at $\bar{k}$ instead of beginning at 1.*

Since, for $k > \bar{k}$, $g(x, k)$ is statistically stationary, each measurement y_i is an *unbiased estimate* of the corresponding value of $f(x_i)$. We assume that a model for the *standard deviation* quantifying the uncertainty of this measurement, denoted $\sigma_i = \sigma(x_i, N_i)$, is also available. Since $g(x, t)$ is a stationary ergodic process, for any point $x_i \in L$,

$$\lim_{N_i \to \infty} y(x_i, N_i) = f(x_i), \quad \lim_{N_i \to \infty} \sigma(x_i, N_i) = 0. \tag{5.3}$$

Remark 29. *If a stationary ergodic process $g(x,k)$ at some point $x_i \in L$ is independent and identically distributed (IID), then $\sigma(x_i, N_i) = \sigma(x_i, 1)/\sqrt{N_i}$; otherwise, estimates of $\sigma(x_i, N_i)$ can be developed using standard uncertainty quantification (UQ) procedures, such as those developed in [86, 87, 88, 89, 90]. The discrete-time process $g(x,k)$ may often be obtained by sampling a continuous-time process $g(x,t)$ at timesteps $t_k = kh$ for some appropriate sample interval h. For sufficiently large h, the samples of this continuous-time process $g(x,k)$ are often essentially IID; however, with the appropriate UQ procedures in place, significantly smaller sample intervals h will lead to a given degree of convergence in a substantially shorten period of time t, albeit with increased storage.*

Definition 27. *Define S as a set of measurements y_i, for $i = 1, 2, \ldots, M$, at corresponding points x_i and with standard deviation σ_i. We will call a regression $p(x)$ for this set of measurements a strict regression if, for some constant β,*

$$|p(x_i) - y_i| \leq \beta\, \sigma_i, \quad \forall\ 1 \leq i \leq M. \tag{5.4}$$

Based on the concepts defined above, Algorithm 5.1 presents our algorithm to efficiently and globally minimize a function of the form (5.1a) within a feasible domain defined by (5.1b). At each iteration k of this algorithm, S^k denotes the set of M points x_i, for $i = 1, 2, \ldots, M$, at which measurements have so far been made; for each point $x_i \in S^k$, $y_i = y(x_i, N_i)$ denotes the measured value, $\sigma_i = \sigma(x_i, N_i)$ denotes the uncertainty of this estimate, and N_i quantifies the sampling performed thus far at point x_i. Note that the values of M, N_i, y_i, ℓ, α, and K are all updated from time to time as the iterations proceed, and are thus annotated with a k superscript at various points in the analysis of §5.4 to remove ambiguity. Akin to Algorithm 2.2, at iteration k, $p^k(x)$ is assumed to be a strict regression

Algorithm 5.1 The new optimization algorithm, dubbed α-DOGS, for minimizing the function $f(x)$ in (5.1a) within the feasible domain L defined in (5.1b).

1: Set $k = 0$ and initialize the algorithm parameters α, K, γ, β, ℓ, N^0, and N^δ as discussed in §5.2. Take the initial set of M sampled points, S^0, as the 2^n vertices of the feasible domain $L = \{x | a \le x \le b\}$ together with any user-supplied points of interest quantized onto the grid L_ℓ. Take $N_i = N^0$ for $i = 1, \ldots, M$, and compute an initial measurement $y_i = y(x_i, N^0)$ and corresponding uncertainty $\sigma_i = \sigma(x_i, N^0)$ for each point $x_i \in S^0$.

2: Calculate a strict regression $p^k(x)$ for all M available measurements.

3: Calculate (or, for $k > 0$, update) a Delaunay triangulation Δ^k over all of the points in S^k.

4: Determine x_j as the minimizer (and, j as the corresponding index), over all $x_i \in S^k$, of the discrete search function $s_d^k(x_i)$, defined as follows:

$$s_d^k(x_i) = \min\{p^k(x_i), 2\,y_i - p^k(x_i)\} - \alpha\,\sigma_i^k \quad \text{for } i = 1, \ldots, M. \tag{5.5}$$

5: Find $x \in L$ that minimizes the continuous search function $s_c^k(x)$, defined as follows:

$$s_c^k(x) = p^k(x) - K\,e^k(x) \quad \text{for } x \in L. \tag{5.6}$$

Denote z_ℓ as the quantization of z onto the grid L_ℓ.

6: If $s_c^k(z) > s_d^k(x_j)$ and $N_j < \gamma\,2^\ell$ where N_j is the current number of samples taken at x_j, then take N^δ additional samples at x_j, update $N_j \leftarrow N_j + N^\delta$, update the measurement $y_j = y(x_j, N_j)$ and uncertainty $\sigma_j = \sigma(x_j, N_j)$, increment k, and repeat from 2.

7: Otherwise, if $z_\ell \notin S^k$, then set $x_{M+1} = z_\ell$ and $S^{k+1} = S^k \cup \{x_{M+1}\}$, take $N_{M+1} = N^0$, compute the measurement $y_{M+1} = y(x_{M+1}, N^0)$ and uncertainty $\sigma_{M+1} = \sigma(x_{M+1}, N^0)$, increment M and k, and repeat from 2.

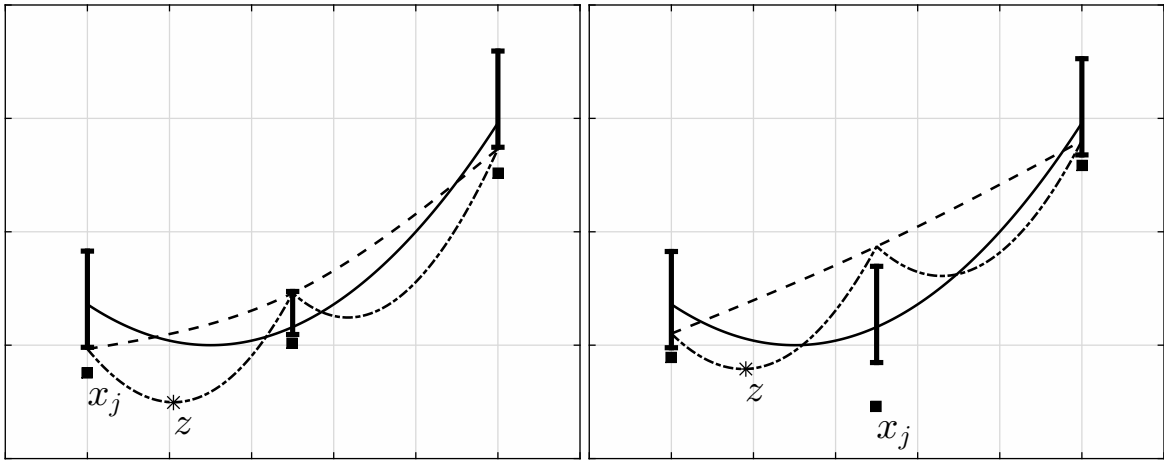
8: Otherwise, increment both ℓ and k, adjust the algorithm parameters such that $\alpha \leftarrow \alpha + \alpha^\delta$ and $K \leftarrow 2\,K$, and repeat from 2.

(for some value of β) of the current set of measurements y_i, and ℓ is the current grid level.

At each iteration of Algorithm 5.1, there are three possible situations, corresponding to three of the numbered iterations of this algorithm:

(6) The sampling of an existing measurement is increased. This is called a *supplemental sampling* iteration.

(7) A new point is identified, and an initial measurement at this point is added to the dataset. This is called an *identifying sampling* iteration.



a An identifying sampling iteration b A supplemental sampling iteration.

Figure 5.1: Representation of one iteration in Algorithm 5.1 in different situations: (solid line) truth function $f(x)$, (dashed line) the regression $p^k(x)$, (dash-dot line) continuous search function $s_c^k(x)$, (closed squares) $s_d^k(x)$, and (asterix) z. In figure (a), $s_c^k(z) < s_d^k(x_j)$; it is thus an identifying sampling iteration. In figure (b), $s_c^k(z) > s_d^k(x_j)$; it is thus an supplemental sampling iteration.

(8) The mesh coordinating the problem is refined and the algorithm parameters α and K adjusted. This is called a *grid refinement* iteration.

Figure 5.1 illustrates supplemental sampling and identifying sampling iterations of Algorithm 5.1.

Algorithm 5.1 depends upon a handful of algorithm parameters, the selection of which affects its rate of convergence, explored in §5.5, though not its proof of convergence, established in §5.4. The remainder of this section discusses heuristic strategies to tune these algorithm parameters, noting that this tuning is an application-specific problem, and alternative strategies (based on experiment or intuition) might lead to more rapid convergence for certain problems.

The first task encountered during the setup of the optimization problem is the definition of the design parameters. Note that the feasible domain considered during the

optimization process is characterized by simple upper and lower bounds for each design parameter; normalizing all design parameters to lie between 0 and 1 is often helpful.

The second challenge is to scale the function $f(x)$ itself, such that the range of the normalized function $f(x)$ over the feasible domain L is about unity. If an estimate of the actual range of $f(x)$ is not available a priori, we may estimate it at any given iteration using the available measurements. Following this approach, at any iteration k with available measurements $\{y_1, y_2, \ldots, y_M\}$, all measurements y_i, as well as the corresponding uncertainty of these measurements σ_i, may be scaled by a factor r_s wherever used in iteration k of Algorithm 5.1 where, for that iteration, r_s is computed such that

$$r = \frac{1}{\max_{1\leq i\leq M}\{y_i\} - \min_{1\leq i\leq M}\{y_i\}}, \quad r_s = r_l + R(r - r_l) - R(r - r_u),$$

where $R(x)$ is the ramp function. So defined, r_s is a "saturated" version of the factor r, constrained to lie in the range $r_l \leq r_s \leq r_u$. Note that scaling the y_i and σ_i does not interfere with the proof of the convergence of the algorithm, provided in §5.4, but can improve its performance. In the numerical simulations performed in §5.5, we take $r_l = 10^{-3}$ and $r_u = 10^3$.

For problems which are IID with no initial transient, $N^0 = N^\delta = 1$ is a reasonable starting point; increasing N^0 and N^δ ultimately reduces the number of iterations of the algorithm (and, thus, the number of Delaunay triangulations) required for convergence, but generally increases slightly the total amount of sampling performed. Suggested values of other algorithm parameters, which work well in the numerical simulations reported in §5.5 but the values of which do not affect the proof of convergence provided in §5.4, include $\alpha^0 = \alpha^\delta = 0.5$, $K^0 = 0.5$, $\ell^0 = 3$, $\beta = 4$, and $\gamma = 100$.

5.3 Polyharmonic spline regression

The algorithm described in this chapter depends upon a smooth regression $p^k(x)$ (see Assumption 9). The best technique for computing the regression is problem dependent. A key advantage of our Delaunay-based approach in the present work is that it facilitates the use of *any* suitable regression technique, subject to it satisfying the "strict" regression property given in Definition 27. Since our numerical tests all implement the polyharmonic spline regression technique, the derivation of this regression technique is briefly explained in this appendix; additional details may be found in [31].

The polyharmonic spline regression $p(x)$ of a function $f(x)$ in $\mathbb{R}^n$ is defined as a weighted sum of a set of radial basis functions $\varphi(r)$ built around the location of each measurement point, plus a linear function of x:

$$p(x) = \sum_{i=1}^{N} w_i \, \varphi(r) + v^T \begin{bmatrix} 1 \\ x \end{bmatrix}, \tag{5.7}$$

$$\text{where} \quad \varphi(r) = r^3 \quad \text{and} \quad r = \|x - x_i\|.$$

The weights w_i and v_i represent N and $n + 1$ unknowns. Assume that $\{y(x_1), y(x_2), \ldots, y(x_n)\}$ is the set of measurements, with standard deviations $\{\sigma_1, \sigma_2, \ldots, \sigma_2\}$. The w_i and v_i coefficients are computed by minimizing the following objective function, which expresses is a tradeoff between the fit to the observed data and the smoothness of the regressor:

$$L_p(x) = \sum_{i=1}^{N} \Big[\frac{(p(x_i) - y(x_i))}{\sigma_i} \Big]^2 + \lambda \int_B |\nabla^m p(x)|, \tag{5.8}$$

where B is a large box domain containing all of the x_i, and $\nabla^m p(x)$ is the vector including all m derivatives of $p(x)$ (see [91]). It is shown in [91] that the first-order optimality condition

for the objective function (5.8) is as follows:

$$p(x_i) - y(x_i) + \rho\,\sigma_i^2 w_i = 0, \quad \forall 1 \le i \le N, \tag{5.9}$$

where ρ is a parameter proportional to λ. In summary, the coefficient of the regression can be derived by solving:

$$\begin{bmatrix} F & V^T \\ V & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} f(x_i) \\ 0 \end{bmatrix}, \tag{5.10}$$

$$F_{ij} = \varphi(\|x_i - x_j\|) + \rho\delta_{i,j}\,\sigma_i^2, \qquad V = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \end{bmatrix},$$

where $\delta_{i,j}$ is the Kronecker delta.

The problem which is left to solve when computing the regression is to find an appropriate value of $\rho \in [0, \infty)$. Solving (5.10) for any value of ρ gives a unique regression, denoted $p(x, \rho)$. The parameter ρ is then obtained by a predictive mean-square error criteria developed in §4.4 in [31], which is given by imposing the following condition:

$$T(\rho) = \sum_{i=1}^{N} [\frac{p(x_i, \rho) - y(x_i)}{\sigma_i}]^2 = 1. \tag{5.11}$$

For $\rho \to \infty$, $w_i \to 0$, and the solution of (5.10) is a weighted mean-square linear regression, which is obtained by solving (5.11). If $T(\infty) \le 1$, we take this linear regression as the best current regression for the available data. Otherwise, we have $T(\infty) > 1$ and (by construction) $T(0) = 0$; thus, (5.11) has a solution with finite $\rho > 0$ which gives the desired regression.

If $T(\infty) > 1$, we thus seek a ρ for which for $T(\rho) = 1$. Following [31], using (5.10),

(5.11) simplifies to:

$$T(\rho) = \rho \sum_{i=1}^{N} w_{i,\rho}\, \sigma_i^2 = 1, \tag{5.12}$$

where $w_{i,\rho}$ is the w_i which is obtained by solving (5.10). Define Dw and Dv as the vectors whose i-th elements are the derivatives of w_i and v_i with respect to ρ, then

$$T'(\rho) = \sum_{i=1}^{N} Dw_i \sigma_i^2 + \rho \sum_{i=1}^{N} Dw_{i,\rho} \sigma_i^2,$$

$$\begin{bmatrix} F & V^T \\ V & 0 \end{bmatrix} \begin{bmatrix} Dw \\ Dv \end{bmatrix} + \begin{bmatrix} \rho\Sigma_2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where Σ_2 is a diagonal matrix whose i-the diagonal element is $\rho\, \sigma_i^2$. Therefore, the analytic expression for the derivative of $T(\rho)$ is available. Thus, (5.11) can be solved quickly using Newton's method.

The regression process presented here, imposing (5.12) as suggested by [31], is designed to obtain a regression which is reasonably smooth. However, there is no guaranty that this particular regression satisfies the strictness property required in the present work (see Definition 27). Note, however, that by imposing $\rho = 0$, the regression is made strict for arbitrary small β. Thus, to satisfy strictness for a given finite β, the value of ρ must sometimes be decreased from that which satisfies (5.12), as necessary.

5.4 Analysis of α-DOGS

We now analyze the convergence of Algorithm 5.1. We first present some preliminary definitions.

Definition 28. *The point $\eta^k \in S$ is called the candidate point at iteration k if*

$$\eta^k = argmin_{z \in S^k}\{y_k(z) + \alpha^k \sigma_k(z)\}. \tag{5.13}$$

Define $f(x^*)$ as the global minimum of $f(x)$ in L, then the regret is defined as

$$r_k = f(v_k) - f(x^*). \tag{5.14}$$

The definition of the regret given above is common in the optimization literature (see, e.g., [92, 69, 83]). We show in this section that, under the following assumptions, the regret of the optimization process governed by Algorithm 5.1 will converge to zero:

Assumption 9. *A constant $\hat{K}$ exist such that, for all $k > 0$ and $x \in L$,*

$$-2\,\hat{K} \leq \nabla^2 p^k(x) \leq 2\,\hat{K}, \quad -2\,\hat{K} \leq \nabla^2 f(x) \leq 2\,\hat{K}.$$

Assumption 10. *There is a real positive function $E(x) : \mathbb{R}^+ \to \mathbb{R}^+$, which has the following properties:*

a. $E(x)$ is continuous and monotonically increasing.

b. $E(x)$ is bounded such that

$$E(x) \leq Q, \quad \sup_{x \in \mathbb{R}^+} E(x) = Q. \tag{5.15}$$

c. For all $x \in L$ and $N \in \mathbb{N}$, we have:

$$\left|\frac{1}{N}\sum_{k=1}^{N} g(x,k) - f(x)\right| = |y(x,N) - f(x)| \leq E(\sigma(x,N)). \tag{5.16}$$

d. $\lim_{x\to 0^+} E(x) \to 0$.

Assumption 11. *There are real numbers* $\alpha > 0$ *and* $\theta \in (0\ 1]$*, such that*

$$\sigma(x, N) \leq \alpha N^{-\theta} \quad \forall x \in \mathbb{R},\ N \in \mathbb{N}. \tag{5.17}$$

Note that, if the stationary process $g(x, k)$ has a short-range dependence, like ARMA processes, the parameter $\theta = 0.5$. However, for different θ, this general model can also handle stationary processes with long-range dependence, like Fractional ARMA processes. Moreover, at any given point $x \in L$, $\sigma(x, N)$ is a monotonically nonincreasing function of N. This condition means that, by increasing the averaging interval, the uncertainty of the estimate must not increase.

Remark 30. *Assumption 2 is a stronger condition than ergodicity of* $g(x, k)$*. Recall that ergodicity of* $g(x, k)$ *is equivalent to the convergence of* $y(x, N)$ *to* $f(x)$ *as* $N \to \infty$*, which is the straightforward outcome of Assumptions 10 and 11. In Assumption 10, the convergence of the sample mean is assumed to be bounded by a function of* $\sigma(x, N)$ *of the form specified.*

Lemma 23. *For any point* $x \in L$ *and real positive number* $0 < \varepsilon < Q$*,*

$$|y(x, N) - f(x)| - \frac{Q}{E^{-1}(\varepsilon)}\,\sigma(x, N) \leq \varepsilon, \tag{5.18}$$

Proof. If $\sigma(x, N) \leq E^{-1}(\varepsilon)$ then, since $E(x)$ is an increasing function, by (5.16), we have:

$$E(\sigma(x, N)) \leq E(E^{-1}(\varepsilon)) = \varepsilon \quad \Rightarrow \quad |y(x, N) - f(x)| \leq \varepsilon.$$

Otherwise, $\sigma(x,N) > E^{-1}(\varepsilon)$; thus, again by (5.16), we have

$$\frac{Q}{E^{-1}(\varepsilon)}\,\sigma(x,N) \geq Q \quad \Rightarrow \quad |y(x,N) - f(x)| - Q \leq E(\sigma_N) - Q \leq 0,$$

Thus, (5.18) is verified for both cases. □

Lemma 24. *During the execution of Algorithm 5.1, there are an infinite number of mesh refinement iterations.*

Proof. This lemma is shown by contradiction. If Algorithm 5.1 has a finite number of mesh refinement iterations, then there is an integer number $\bar{\ell}$ such that the mesh $L_{\bar{\ell}}$ contains all datapoints obtained by the algorithm. Since the number of datapoints on this mesh is finite, only a finite number of points must be considered, which leads to having a finite number of identifying sampling iterations.

Since the number of identifying sampling and mesh refinement iterations are finite, there must be an infinite number of supplemental sampling iterations. At each supplemental sampling iteration, the averaging length of the estimate at an existing datapoint is incremented by $N^{\delta} \geq 1$. Since only a finite number of points is considered, a datapoint exists for which the estimate is improved for an infinite number of supplemental sampling iterations. As a result, there is an supplemental sampling iteration, such that $N_j > \gamma 2^{\bar{\ell}}$, which is in contradiction with the assumption of having finite number of mesh refinement iterations. □

Lemma 25. *Consider z, x_j, and x^* as global minimizers of $s_c^k(x)$, $s_d^k(x)$, and $f(x)$, respectively. Note that $s_d^k(x)$ is only defined for the points in S^k, but $s_c^k(x)$ and $f(x)$ are defined*

over the feasible domain L. Define M_k as:

$$M_k = \min\{s_c^k(z) - f(x^*), s_d^k(x_j) - f(x^*)\}. \tag{5.19}$$

Then,

$$\limsup_{k\to\infty} M_k \leq 0. \tag{5.20}$$

Proof. By Lemma 24, there are infinite number of mesh refinement iterations during the execution of Algorithm 5.1. Thus,

$$\lim_{k\to\infty} K^k = \infty, \quad \lim_{k\to\infty} \alpha^k = \infty. \tag{5.21}$$

As a result, for any $0 < \varepsilon < Q$, there is a k_ε such that, if $k > k_\varepsilon$, then

$$K^k \geq 3\,\hat{K} \quad \text{and} \quad \alpha^k \geq \frac{2\,Q}{E^{-1}(\varepsilon)}. \tag{5.22}$$

Consider $\Delta_{x^*}^k$ as a simplex in Δ^k, a Delaunay triangulation for S^k, that contains x^*. Define $M(x) : \Delta_{x^*}^k \to \mathbb{R}$ as the unique linear function in $\Delta_{x^*}^k$ such that

$$M(V_j^k) = 2\,f(V_j^k) - p^k(V_j^k),$$

where V_j^k are the vertices of $\Delta_{x^*}^k$. Define $G(x) : \Delta_{x^*}^k \to \mathbb{R}$ as follows:

$$G(x) = s_c^k(x) + M(x) - 2\,f(x) = p^k(x) + M(x) - 2\,f(x) - K^k\,e^k(x).$$

By construction, $G(V_j^k) = 0$. Moreover:

$$\nabla^2 G(x) = \nabla^2\{p^k(x) - 2\,f(x)\} + 2\,K^k\,I.$$

Using Assumption 9 and (5.22), $G(x)$ is strictly convex in simplex $\Delta_{x^*}^k$. Since $G(x) = 0$ at the vertices of $\Delta_{x^*}^k$, then $G(x^*) \leq 0$.

Moreover, since $M(x)$ is a linear function, then

$$\min_{x \in S^k}[2\,f(x) - p^k(x)] \leq \min_{1 \leq j \leq n+1}[2\,f(V_j^k) - p^k(V_j^k)] \leq M(x^*),$$

$$s_d^k(x) \leq [2\,f(x) - p^k(x)] + 2\,(y_k(x) - f(x)) - \alpha^k \sigma(x). \tag{5.23}$$

Using (5.18) in Lemma 23 and (5.22) leads to:

$$2\,y_k(x) - 2\,f(x) - \alpha^k \sigma(x) \leq 2\,\varepsilon. \tag{5.24}$$

Combining (5.23) and (5.24) leads to:

$$s_d^k(x) \leq [2\,f(x) - p^k(x)] + 2\,\varepsilon.$$

Since x_j is the minimizer of the $s_d^k(x)$,

$$s_d^k(x_j) \leq \min_{x \in S^k}[2\,f(x) - p^k(x)] + 2\,\varepsilon \leq M(x^*) + 2\,\varepsilon.$$

Furthermore, z is the global minimizer of $s_c^k(x)$ and $G(x^*) \leq 0$; therefore,

$$s_c^k(z) \leq s_c^k(x^*) \leq 2\, f(x^*) - M(x^*),$$
$$s_c^k(z) + s_d^k(x_j) \leq 2\, f(x^*) + 2\, \varepsilon. \tag{5.25}$$

Thus, for any $\varepsilon > 0$ and $k > \hat{k}_\varepsilon$, (5.25) is satisfied; therefore, (5.20) is verified. □

Lemma 26. *If* $\{k_1, k_2, \dots\}$ *are the mesh refinement iterations of Algorithm 5.1, then*

$$\limsup_{i\to\infty} \left\{ y(\eta^{k_i}, N_{\eta^{k_i}}^{k_i}) - f(x^*) + \alpha^{k_i} \sigma(\eta^{k_i}, N_{\eta^{k_i}}^{k_i}) \right\} \leq 0, \quad \textit{and} \tag{5.26a}$$

$$\lim_{i\to\infty} \sigma(\eta^{k_i}, N_{\eta^{k_i}}^{k_i}) = 0, \tag{5.26b}$$

where η^{k_i} *is the candidate point at iteration* k_i *and* x^* *is a global minimizer of* $f(x)$ *in* L.

Proof. Consider z as a global minimizer of $s_c^{k_i}(x)$ in L, and z_ℓ as its quantization on L_ℓ. Since iteration k_i is a mesh refinement, $z_\ell \in S^{k_i}$. Consider $\Delta_j^{k_i}$ as a simplex in the Delaunay triangulation Δ^{k_i} which contains z. According to the uncertainty function $e^{k_i}(x)$,

$$e^{k_i}(z_\ell) \geq e_j^{k_i}(z_\ell), \quad e^{k_i}(z) = e_j^{k_i}(z),$$
$$s_c^{k_i}(z_\ell) - s_c^{k_i}(z) = p^{k_i}(z_\ell) - p^{k_i}(z) + K^{k_i}(e^{k_i}(z) - e^{k_i}(z_\ell)),$$
$$s_c^{k_i}(z_\ell) - s_c^{k_i}(z) \leq p^{k_i}(z_\ell) - p^{k_i}(z) + K^{k_i}(e_j^{k_i}(z) - e_j^{k_i}(z_\ell)).$$

Using the property of the Cartesian grid quantizer and its quantizer, $A_a(z) \subseteq A_a(z_\ell)$. According to Assumption 9 and the construction of the uncertainty function introduced, $\nabla^2\{p^{k_i}(x) - K^{k_i} e_j^{k_i}(x)\} - \{\hat{K} + 2\, K^{k_i}\} I \leq 0$; thus, by Lemma 19 and the fact (see Lemma 5 in [93] for

proof) that z globally minimizes $p^{k_i}(x) - K^{k_i} e_j^{k_i}(x)$,

$$s_c^{k_i}(z_\ell) - s_c^{k_i}(z) \leq \{\hat{K} + 2\,K^{k_i}\}\|z_\ell - z\|^2. \tag{5.27}$$

Define δ_{k_i} as the maximum quantization error at iteration k_i, then $\|z_\ell - z\| \leq \delta_{k_i}$. On the other hand, $z_\ell \in S^{k_i}$, which leads to $s_c^{k_i}(z_\ell) = p^{k_i}(z_\ell)$, and

$$p^{k_i}(z_\ell) \leq s_c^{k_i}(z) + \{\hat{K} + 2\,K^{k_i}\}\delta_{k_i}^2. \tag{5.28}$$

At each mesh refinement iteration of Algorithm 5.1, there are two possibilities. In the first case, $s_c^{k_i}(z) \leq s_d^{k_i}(x_j)$; since x_j is a minimizer of $s_d^{k_i}(x)$, then

$$\begin{aligned} p^{k_i}(z_\ell) &\leq s_d^{k_i}(z_\ell) + \{\hat{K} + 2\,K^{k_i}\}\delta_{k_i}^2, \\ s_d^{k_i}(z_\ell) &\leq p^{k_i}(z_\ell) - \alpha^{k_i}\sigma(z_\ell, N_{z_\ell}^{k_i}), \\ \sigma(z_\ell, N_{z_\ell}^{k_i}) &\leq \frac{\{\hat{K} + 2\,K^{k_i}\}}{\alpha^{k_i}}\delta_{k_i}^2. \end{aligned} \tag{5.29}$$

Using (5.19) (see Lemma 25) and (5.28) leads to

$$p^{k_i}(z_\ell) - f(x^*) \leq M_{k_i} + \{\hat{K} + 2\,K^{k_i}\}\delta_{k_i}^2. \tag{5.30}$$

Since the regression is strict,

$$y(z_\ell, N_{z_\ell}^{k_i}) - p^{k_i}(z_\ell) \leq \beta\,\sigma(z_\ell, N_{z_\ell}^{k_i}). \tag{5.31}$$

Using (5.29), (5.30), and (5.31) leads to

$$y(z_\ell, N_{z_\ell}^{k_i}) - f(x^*) \leq M_{k_i} + \{\hat{K} + 2\,K^{k_i}\}\delta_{k_i}^2 + \beta\Big[\frac{\{\hat{K} + 2\,K^{k_i}\}}{\alpha^{k_i}}\delta_{k_i}^2\Big]. \tag{5.32}$$

In the second case, $s_c^{k_i}(z) > s_d^{k_i}(x_j)$, then by the construction of M_{k_i} (see (5.19)),

$$s_d^{k_i}(x_j) - f(x^*) = M_{k_i}. \tag{5.33}$$

Moreover, since iteration k_i is mesh refinement, then the sampling $N_j \geq \gamma 2^\ell$. Thus, using Assumption 11,

$$\sigma(x_j, N_{x_j}^{k_i}) \leq \alpha\,\gamma^{-\theta} 2^{-\theta\ell}. \tag{5.34}$$

Furthermore, the regression $p^{k_i}(x)$ is strict which leads to:

$$y(x_j, N_{x_j}^{k_i}) - s_d^{k_i}(x_j) \leq (\beta + \alpha^{k_i})\,\sigma(x_j, N_{x_j}^{k_i}) \tag{5.35}$$

Using (5.33), (5.34), and (5.35) leads to

$$y(x_j, N_{x_j}^{k_i}) - f(x^*) \leq M_{k_i} + (\beta + \alpha^{k_i})\alpha\gamma^{-\theta} 2^{-\theta\ell}. \tag{5.36}$$

Note that η^{k_i} is the candidate point at iteration k_i. Thus, using (5.29), (5.32), (5.34), and (5.36), and the construction of candidate point (see Definition 28),

$$\begin{aligned} y(\eta^{k_i}, N_{\eta^{k_i}}^{k_i}) - f(x^*) + \alpha^{k_i}\sigma(\eta^{k_i}, N_{\eta^{k_i}}^{k_i}) \leq M_{k_i} + \\ \max\Big\{(\beta + \alpha^{k_i})\alpha\gamma^{-\theta}2^{-\theta\ell}, (\hat{K} + 2\,K^{k_i})\delta_{k_i}^2 + \beta\Big[\frac{(\hat{K} + 2\,K^{k_i})}{\alpha^{k_i}}\delta_{k_i}^2\Big]\Big\} \\ +\alpha^{k_i}\max\Big\{\alpha\gamma^{-\theta}2^{-\theta\ell}, \frac{(\hat{K} + 2\,K^{k_i})}{\alpha^{k_i}}\delta_{k_i}^2\Big\}. \end{aligned} \tag{5.37}$$

On the other hand,

$$\delta_{k_i} = \frac{\|b-a\|}{2^{\ell_0+i}}, \quad \alpha^{k_i} = \alpha^0 + i\,\alpha^\delta, \quad K^{k_i} = K_0\,2^i, \quad \ell^{k_i} = \ell^0 + i. \tag{5.38}$$

By substituting (5.38) in (5.37) and using (5.20) (see Lemma 25), (5.26a) is verified. Furthermore, using Assumption 10, we have

$$|y(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) - f(\eta^{k_i})| \leq E(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) \leq Q. \tag{5.39}$$

Thus, using (5.26a), $f(\eta^{k_i}) - f(x^*) > 0$, and (5.39) leads to

$$\limsup_{i\to\infty} \Big\{ -Q + \alpha^{k_i}\sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) \Big\} \leq 0.$$

Since $\sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) \geq 0$ and $\lim_{i\to\infty} \alpha^{k_i} = \infty$, (5.26b) is verified.

□

Theorem 11. *Consider η^k as the candidate point at iteration k of Algorithm 5.1, then*

$$\lim_{k\to\infty} f(\eta^k) = f(x^*). \tag{5.40}$$

Proof. At any iteration $k > k_1$, take $k_i < k$ as the most recent mesh refinement iteration of Algorithm 5.1. Then $\eta^{k_i} \in S^k$, and

$$y(\eta^k, N^k_{\eta^k}) + \alpha^k\sigma(\eta^k, N^k_{\eta^k}) \leq y(\eta^{k_i}, N^k_{\eta^{k_i}}) + \alpha^k\sigma(\eta^{k_i}, N^k_{\eta^{k_i}}). \tag{5.41}$$

Using Assumption 10 leads to:

$$|y(\eta^{k_i}, N^k_{\eta^{k_i}}) - y(\eta^{k_i}, N^{k_i}_{\eta^{k_i}})| \leq E(\sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}})) + E(\sigma(\eta^{k_i}, N^k_{\eta^{k_i}})),$$

$$y(\eta^k, N^k_{\eta^k}) + \alpha^k \sigma(\eta^k, N^k_{\eta^k}) \leq y(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) + \alpha^k \sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) +$$

$$E(\sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}})) + E(\sigma(\eta^{k_i}, N^k_{\eta^{k_i}})).$$

By construction, since the sampling at η^{k_i} at iteration k is greater than or equal to its sampling at iteration k_i, $\sigma(\eta^{k_i}, N^k_{\eta^{k_i}}) \leq \sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}})$. Since the function $E(x)$ is nondecreasing,

$$y(\eta^k, N^k_{\eta^k}) + \alpha^k \sigma(\eta^k, N^k_{\eta^k}) \leq y(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) + \alpha^k \sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}}) + 2\, E(\sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}})).$$

Using (5.26) in Lemma 26, Assumption 10, and $\alpha^k = \alpha^{k_i} + \alpha^\delta$, leads to:

$$\limsup_{k\to\infty} y(\eta^k, N^k_{\eta^k}) - f(x^*) + \alpha^k \sigma(\eta^k, N^k_{\eta^k}) \leq \limsup_{k\to\infty} \{\alpha^\delta \sigma(\eta^{k_i}, N^{k_i}_{\eta^{k_i}})\} = 0. \tag{5.42}$$

Similar to the proof of (5.26b), it is thus again easy to show

$$\lim_{i\to\infty} \sigma(\eta^k, N^k_{\eta^k}) = 0.$$

On the other hand, based on Assumption 10, and optimality of $f(x^*)$

$$f(\eta^k) + \alpha^k \sigma(\eta^k, N^k_{\eta^k}) - f(x^*) - E(\sigma(\eta^k, N^k_{\eta^k})) \leq y(\eta^k, N^k_{\eta^k}) + \alpha^k \sigma(\eta^k, N^k_{\eta^k}) - f(x^*),$$

$$\lim_{k\to\infty} E(\sigma(\eta^k, N^k_{\eta^k})) = 0,$$

$$\limsup_{k\to\infty} f(\eta^k) - f(x^*) \leq 0.$$

Since $f(\eta^k) - f(x^*) \geq 0$, (5.40) is verified. □

5.5 Results

We now illustrate the performance of Algorithm 5.1 on some representative examples. The function $g(x, k)$ in (5.1a) is assumed to be a discrete-time statistically stationary random ergotic process. In this section, we further assume that $g(x, k)$ is IID in the index k, and that the variation of $g(x, k)$ from the truth function $f(x)$ is homogeneous in x. In particular, we take $\sigma(x_i, 1) = 0.3$, and

$$g(x, k) = f(x) + v_k \quad \text{where } v_k = \mathcal{N}(0, 0.09).$$

In this section, two different test functions for $f(x)$ are considered within the simple feasible domain $L = \{x | 0 \leq x_i \leq 1 \ \forall i\}$, the *shifted parabolic function*

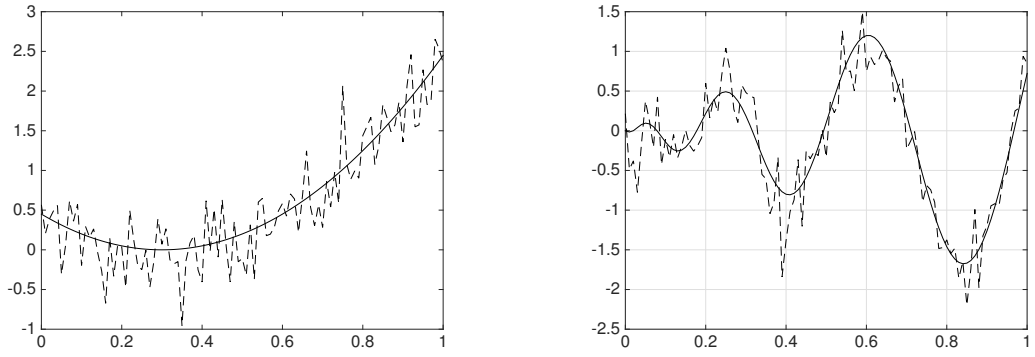
$$f(x) = \sum_{i=1}^{n} (x_i - 0.3)^2, \tag{5.43}$$

with a global minimizer in L of $x_i^* = 0.3$ and a corresponding global minimum of $f(x^*) = 0$, and the *scaled Schwefel fuction*

$$f(x) = 1.6759\, n - \sum_{i=1}^{n} \frac{x_i}{2} \sin(500\, |x_i|), \tag{5.44}$$

with a global minimizer in L of $x_i^* = 0.8419$ and a corresponding global minimum of $f(x^*) = -1.6759\, n$. We will consider these two functions in $n = 1$, 2, and 3 dimensions.

One-dimensional representations of these functions are illustrated in Fig 5.2: for the shifted parabolic function (5.43), the truth function (unknown to the optimization al-

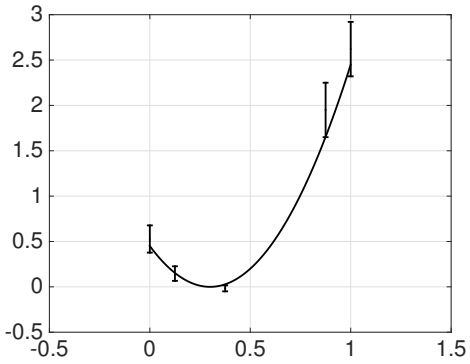


a Shifted parabolic function (5.43). b Scaled Schwefel fuction (5.44).

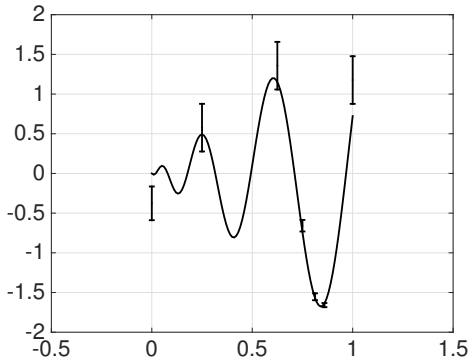
Figure 5.2: Illustration of test problems (5.43) and (5.44). (solid line) truth function $f(x)$, and (dashed line) a set of measurements y_i computed with a single sample at each measurement, $N_i = 1$.

gorithm) is a simple parabola, whereas for the scaled Schwefel fuction (5.44), the truth function is a smooth nonconvex function with four local minima. Note that the perturbations present in several measurements of these functions, computed with finite N_i, result in a complicated, nonsmooth, nonconvex behavior. This chapter shows how to efficiently minimize such functions based only on such noisy measurements, automatically refining the measurements (by increasing the sampling) as convergence is approached.

The optimizations are initialized with measurements of sample length $N^0 = 1$ at the vertices of L. Figure 5.3 illustrates the application of Algorithm 5.1 after $k = 100$ iterations in the 1D case, taking $N^0 = N^\delta = 1$ additional sample (at either a new measurement point, or at an existing measurement point) at each iteration of the algorithm. In Figure 5.3a, the sampling N_i after $k = 100$ iterations (plus the 2 initial sample points, for a total of 102 samples) at the $M = 5$ measured points y_i indicated, enumerated from left to right, is $\{4, 14, 82, 1, 1\}$; in Figure 5.3b, the sampling N_i after 100 iterations at the 7 measured points indicated is $\{2, 1, 1, 9, 23, 65, 1\}$. Both results clearly show that the algorithm focuses the bulk of its sampling in the immediate vicinity of the minimum, where the accuracy of



a Shifted parabolic function (5.43).



b Scaled Schwefel function (5.44).

Figure 5.3: Illustration of Algorithm 5.1 on model problems (5.43) and (5.44) in 1D after 100 iterations, taking $N^0 = N^\delta = 1$: (solid line) the truth function $f(x)$, and (error bars) the 66 percent confidence intervals of the measurements.

the measurements is especially important, while avoiding unnecessary sampling far from the minimum, where the accuracy of the measurements is of reduced importance. It is also seen that more exploration is performed for the scaled Schwefel function than for the shifted parabolic function, as a result of its more complex underlying trend.

Since the function evaluation process in these tests has a stochastic component, Algorithm 5.1 was next applied an ensemble of three separate tests, for both model problems discussed above, in each of three different cases with increasingly higher dimension (that is, $n = 1$, $n = 2$, and $n = 3$). The convergence histories of these simulations are illustrated in Figures 5.4 and 5.5.

To better quantify the performance of the algorithm proposed, we now introduce the following concept.

Definition 29. *Assume that the stationary process $g(x, k)$ is IID, and that the nominal variance $\sigma(x_i, 1) = \sigma_0$ for all points $x_i \in L$. As mentioned in Remark 29, denoting N_i as the total number of samples taken at point x_i, the uncertainty of the corresponding measurement y_i given by* (5.2) *is $\sigma_i = \sigma(x_i, N_i) = \sigma_0/\sqrt{N_i}$. If we assume that all of sampling of the*

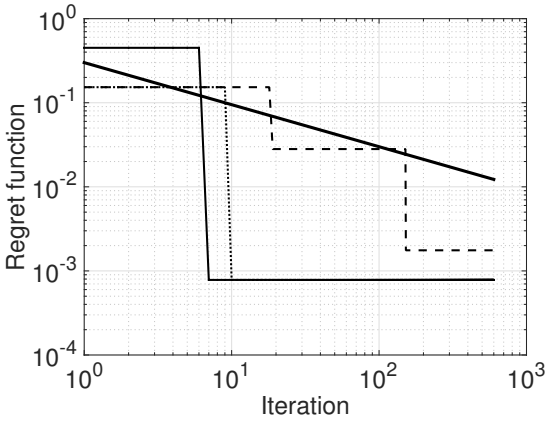
algorithm is performed at a single point, the uncertainty of this single measurement after k samples would thus be $\sigma_0/\sqrt{k}$*, which we refer to as the reference error. This function is indicated in Figures 5.4 and 5.5 by a solid line of slope* $-1/2$ *in log-log coordinates.*

It is remarkable to note that, in all 18 of the optimizations reported in Figures 5.4 and 5.5, in which we have again taken 1 new sample at each iteration, the regret function of Algorithm 5.1 is eventually diminished to a value substantially smaller than the reference error. That is, the value of the regret at the end of these optimizations is actually substantially *less* than the uncertainty of a single measurement, assuming that all of the sampling is done at a single point.

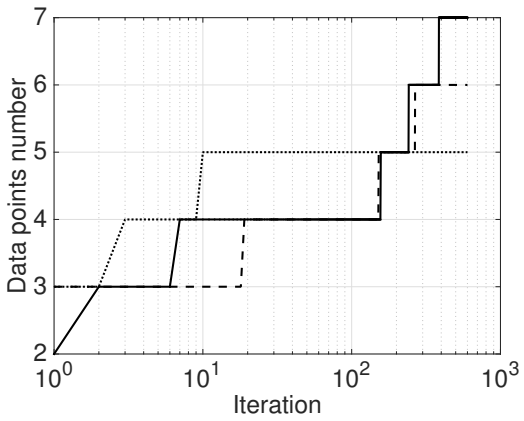
Figures 5.4 and 5.5 also report the number of datapoints which are considered by the optimization algorithm as the iterations proceed. This number is important in optimization problems for which the function evaluations are obtained from simulations which have an (expensive) initial transient, which must be set aside before sampling the statistic of interest, as discussed further in Remark 28. It is observed, as in the 1D case illustrated in Figure 5.3, that the number of datapoints that are considered for the shifted parabolic function is less than that for the scaled Schwefel function. Further, the regret function converges faster to the general proximity of the global solution, in about 10 to 50 iterations, for the shifted parabolic function. This result is reasonable, since the underlying function in the shifted parabolic case is simpler.

5.6 Conclusions

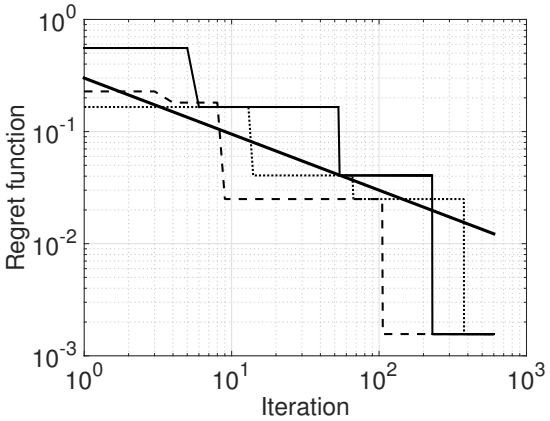
This chapter presents a new optimization algorithm, dubbed α-DOGS, for the minimization of functions given by the infinite-time average of stationary ergodic processes



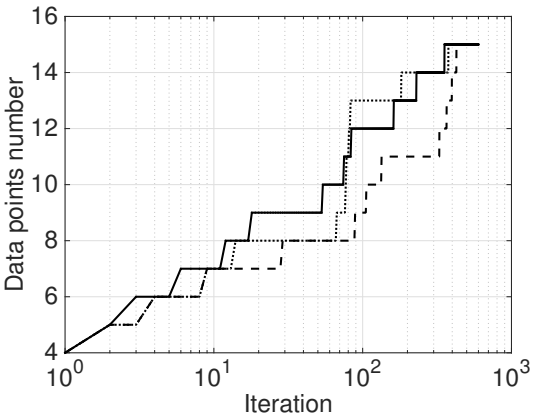
a The regret function in 1D.



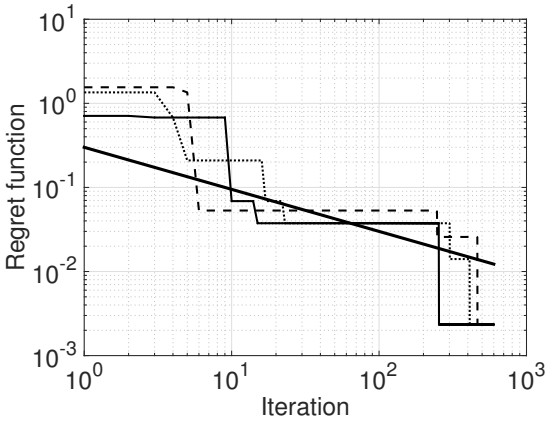
b Total number of datapoints in 1D.



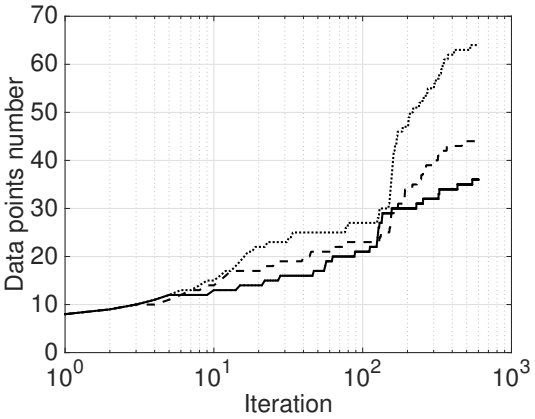
c The regret function in 2D.



d Total number of datapoints in 2D.

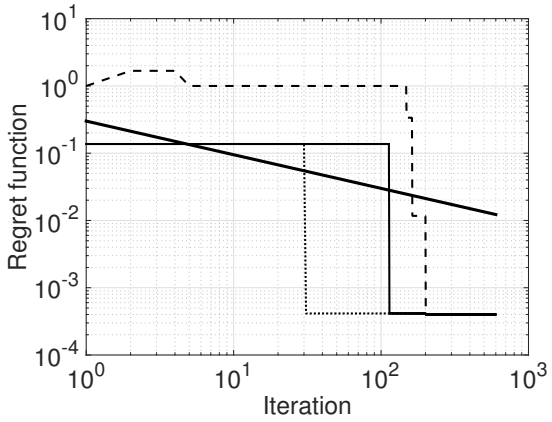


e The regret function in 3D.

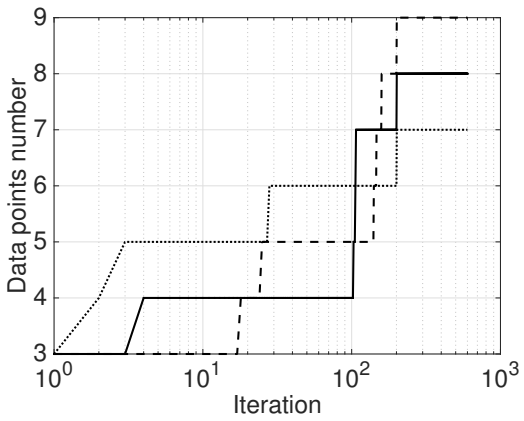


f Total number of datapoints in 3D.

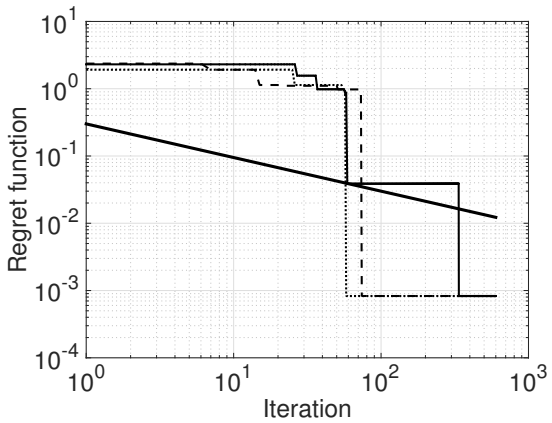
Figure 5.4: Implementation of Algorithm 5.1 on parabolic test problem (5.43).



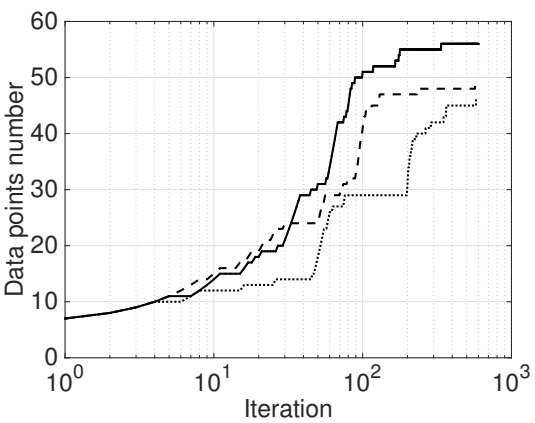
a The regret function in 1D.



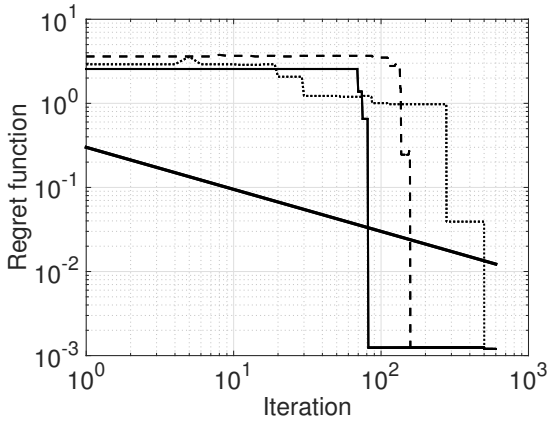
b Total number of datapoints in 1D.



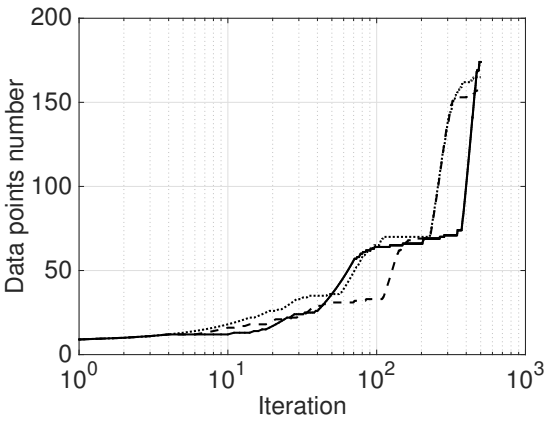
c The regret function in 2D.



d Total number of datapoints in 2D.



e The regret function in 3D.



f Total number of datapoints in 3D.

Figure 5.5: Implementation of Algorithm 5.1 on Schwefel test problem (5.44).

in the computational or experimental setting. Two search functions are considered at each iteration. The first is a continuous search functions, $s_c^k(x)$, defined over the entire feasible space $x \in L$, combining a strict regression $p^k(x)$ of the available datapoints together with a uncertainty function characterizing the distance of any given point in the feasible domain from the nearest measurements, and built on the framework of a Delaunay triangulation over all available measurements at that iteration. The second is a discrete search function, $s_d^k(x_i)$, defined over the available measurements $x_i \in S^k$. A comparison between the minima of these two search functions is made in order to decide between further sampling (and, therefore, refining) an existing measurement, or sampling at a new point in parameter space. The convergence of the algorithm is established in problems for which

a. The underlying truth (infinite-time averaged) function, as well as the regressions computed at each iteration k, are twice differentiable.

b. The stationary process $g(x, k)$ upon which the truth function $f(x)$ is generated, in (5.1a), is ergodic, and the convergence of the averaging process to the underlying truth function is bounded by a monotonic function of a computable uncertainty function (see Assumption 10).

c. The uncertainty of the time averaging process decays exponentially to zero (see Assumption 11); this is true for almost all stationary models of random processes.

The α-DOGS algorithm performs and refines measurements with different amounts of sampling in different locations in the feasible region of parameter space as necessary. By so doing, the total cost of the optimization process is substantially reduced as compared with using existing derivative-free optimization strategies, with the same amount of sampling at different locations in parameter space. Computational experiments demon-

strate that the algorithm developed ultimately devotes most of its sampling time to points in parameter space near to the global minimum. Further, these computational experiments indicate that the regret function (see Definition 28) eventually diminishes to a value that is actually substantially less than the uncertainty of a single measurement, assuming that all of the sampling is done at a single point.

In future work, the α-DOGS algorithm will be applied to additional benchmark and application-based optimization problems, including shape optimization for airfoils and hydrofoils. For problems in which the function is determined computationally (from, e.g., numerical simulations of turbulent flows), the extension of the present framework to, as convergence is approached, simultaneously (a) refine the computational grid, and (b) increase the measurement sampling, is also under development.

Acknowledgment

This chapter is submitted to SIAM Journal on optimization as the following article: P. Beyhaghi, T.R. Bewley, “A derivative-free optimization for efficiently minimizing the infinite time-averaged statistics." The dissertation author was the primary investigator and author of this paper.

Chapter 6

A multiscale, asymptotically unbiased approach to uncertainty quantification in the numerical approximation of infinite time-averaged statistics

6.1 Introduction

Statistical characterizations are essential in many scientific and engineering problems. For example, statistical measures such as turbulent kinetic energy (TKE), skin friction drag, pressure drop, and velocity correlation lengths are of fundamental importance in characterizing turbulent flowfield fluctuations in a time-averaged sense or, if the system is not statistically stationary, in an ensemble-averaged sense (without loss of generality, the present chapter focuses on time averaging in the statistically stationary setting). In practice, only a finite number of samples are available in order to approximate such statistics;

it is thus important to quantify the expected deviation between the quantity measured, obtained with a finite number of samples, and the infinite-time-averaged statistic of interest. This quantity is often referred to as *time-averaging error*, but is sometimes referred to in the turbulence literature as *random error* [90] or *sampling error* [94]; for simplicity, the remainder of this chapter simply calls it *averaging error*.

Estimation of the averaging error plays a key role in determining necessary run times in the large-eddy simulation (LES) and direct numerical simulation (DNS) of turbulent flows of engineering interest; such simulations are often extremely computationally expensive. The importance of determining the averaging error is especially pronounced in optimization problems, such as shape optimization [95], using derivative-free optimization approaches, as such optimization codes perform and compare many different simulations or experiments, at a variety of different sets of feasible values of the design parameters, in search of the optimum point in parameter space. In chapter 5, a derivative-free optimization algorithm has been developed specifically for the efficient minimization of time-averaged statistics of the type considered in this chapter. Accurate uncertainty quantification is of fundamental importance in the effectiveness of such an approach, which adjusts the amount of sampling associated with each individual function evaluation, making function evaluations more accurate (and, thus, more expensive), as required, as convergence is approached.

As mentioned in the abstract, if the measured samples are i.i.d. (such as thermocouple measurements, for which the noise is often well modelled as white), the standard deviation, ε_N, of a finite-sample approximation of an infinite-time-average statistic (that is, the "averaging error") is given by

$$\varepsilon_N = \sigma \sqrt{1/N}, \tag{6.1a}$$

where σ is the standard deviation of a single sample, and N is the number of samples taken. However, in most problems of interest, the measured samples are not i.i.d.; in such problems, convergence is even slower.

In a numerical simulation of a continuous-time chaotic system that is assumed to behave in a stationary ergodic manner, such as a turbulent flow, with sampling times of $t_k = k\,h$ for some sampling interval h, it is actually the total simulation time T, not the total number of samples taken N, that best represents the computational expense of a given measurement. There are four main approaches available in the literature for estimating the averaging error in such problems. The first, developed in [96], imposes the following informative model for the standard deviation of the average over a simulation time T, assuming essentially continuous sampling:

$$\hat{\varepsilon}(T) = \sigma \sqrt{2\,\tau_f / T}, \tag{6.1b}$$

where σ is the standard deviation (from the infinite-time average) of a single sample, $\hat{\varepsilon}(T)$ is a model of the averaging error after time T, and τ_f is a modeling parameter, referred to as the *integral time scale*, which is introduced to model the largest decorrelation timescale of the samples of the system. The integral time scale τ_f is studied extensively in [97, 98]. This model is found to be effective in practice only if the simulation time divided by the integral time scale, T/τ_f, is relatively large, and is thus of specifically limited utility for UQ; regardless, it is a very revealing starting point, as discussed further in the following paragraph.

In the discrete sampling setting considered in this chapter, we take h as the sampling interval, and thus $N = T/h$ as the number of samples taken over a simulation of length T. If h is taken as so large that the samples are effectively decorrelated, (6.1a) holds;

note that this relation may in fact be recovered by redefining $\tau_f = h/2$ in the discrete realization of the model given in (6.1b). As h is made smaller for a given T and τ_f, more samples are taken, but they begin to become correlated (and, thus, do not each provide independent information). The relation given by (6.1b) sets an approximate lower bound on the averaging error over a simulation of length T, assuming continuous sampling. Thus, comparing (6.1b) and (6.1a), taking $c/N = 2\,\tau_f/T$ where c is some $\sim O(10)$ constant, and noting that $N = T/h$, reveals that

$$h = 2\,\tau_f/c \tag{6.2}$$

is a reasonable value for the sampling interval h in a given problem; sampling substantially more frequently than this will not substantially reduce $\varepsilon(T)$, whereas sampling substantially less frequently than this provides less information, thus increasing $\varepsilon(T)$. Note further that, if h is taken as unnecessarily small (and, thus, N as unnecessarily large), then the overhead associated with storing these samples may become significant, and the errors related to the finite precision of the arithmetic used may corrupt the computation of the average. This effectively motivates one to pick an appropriate intermediate sampling interval h according to (6.2), for which samples are indeed correlated with each other. A well-designed UQ method, such as that designed in the present chapter, is thus required to estimate the uncertainty of the approximation of the averaged statistic of interest determined from these samples.

A second approach for estimating the averaging error in such problems is to model the autocorrelation function with a simple exponential decay such that

$$\hat{\rho}(k) = \exp(-\alpha_f\, k), \tag{6.3}$$

where α_f is a fitting parameter, then using this model to estimate the averaging error. With this approach, the parameter α_f is determined from the available data via empirical modeling of the autocorrelation function [99]. Unfortunately, the data upon which this empirical model of the autocorrelation function is built often has spurious oscillations, which can lead to inaccuracies in the estimation of the averaging error. Filtering methods have been shown in [100, 101, 102, 90] to alleviate this problem somewhat, though special care is required in its implementation.

A third approach for estimating the averaging error is known as batch mean methods (see, e.g., [103, 104, 105, 106]). With such methods, the N available samples are divided, in an ad hoc fasion, into p non-overlapping blocks of length $n = N/p$, and the averaged values for these smaller blocks computed to generate another random process. This new process is closer to i.i.d.; the overall averaging error can then be estimated from the nominal deviations of these block averages from their overall average divided by the square root of p. A central challenge with this approach is the determination of the block length n that works best for a given sample size N.

In the fourth approach for estimating the averaging error, a statistical model of the random process is imposed, where the parameters of this model are determined via a maximum likelihood formulation [107]. The statistical model which is typically considered in this setting is an autoregressive moving average (ARMA) process (see, e.g., [108] and [94]). A significant challenge with this approach is the presence of systematic error (a.k.a. "bias") in the uncertainty quantification which does not diminish to zero as the simulation time is increased, as quantified further in §6.4 below.

In this chapter, we present a new method for quantifying the expected squared averaging error of a finite-time-average approximation of an infinite-time-average statistic

of a stationary ergodic process. The method developed (in §6.2) is *multiscale*, meaning that it is based on an autocorrelation model that is tuned to the data to fit the statistic of interest at a range of different timescales. The method developed is proven in §6.3 to be *asymptotically unbiased* (see Definition 30), meaning that the expected squared averaging error asymptotically converges like $Q/\sqrt{N}$ for the same value of Q as the actual system, if it is modelled as a random process with the correct (that is, infinite-time-averaged) mean, variance, and autocorrelation. The maximum likelihood formulation of [107], which is a leading competing UQ strategy, is shown to not satisfy this valuable property. An automated procedure to identify the initial transient in a dataset is also reviewed. A primary application that motivates this work is turbulence research, though many other applications are also envisioned.

The structure of the remainder of the chapter is as follows: Section 6.1.1 reviews a framework to automatically identify the initial transient of a dataset. Once this portion of the dataset is set aside, the remainder of the dataset is modeled as a realization of a stationary ergodic process. Section 6.2 presents our new method to calculate the averaging error for the stationary part of the dataset. Section 6.3 analyzes the salient properties of the new method. Section 6.4 implements the method developed on synthetic data derived from an autoregressive (AR) model, on data derived from the Kuramoto-Sivashinsky (KS) equation, and on data derived for a low-Reynolds number turbulent channel flow DNS at $Re_\tau = 180$.

6.1.1 Identification of the initial transient

In this section, we review three automated procedures to identify approximately the initial transient of a dataset. As stated previously, this is an important first step in

developing an asymptotically unbiased quantification of uncertainty of the average in the applications of interest.

The first approach identifies the smallest transient time such that, by its removal, a second-order stationarity condition is satisfied by the remainder of the dataset. The second-order stationarity condition may be tested in two different ways:

a. the Priestley-Subba-Rao test [109], which is based on a time-varying Fourier spectrum analysis, and

b. the Wavelet Spectrum test [110], which is based on a time-varying wavelet-based analysis.

These two tests are designed to validate or invalidate the stationarity of a given random process, rather than establishing the "degree" of stationarity of a dataset, which is perhaps more appropriate for the problems of interest here.

The second approach determines the initial transient based on von Neumann's randomness test [111, 112, 113, 114], which uses a batch means approach which divides, in an ad hoc fashion, the N available samples into p non-overlapping blocks of length $n = N/p$, then analyzes the distribution of the averages of each block. Such a heuristic procedure, which is somewhat computationally expensive, often leads to acceptable results. Indeed, in problems for which there is a specific time t_1 after which the state of the system is exactly statistically stationary[1], the method developed in [85] is shown to identify t_1 correctly in the limit that the simulation time T goes to infinity.

The third approach, which is implemented in the present work and is computationally quite inexpensive, was originally introduced in [115, 116, 117] for numerical simula-

[1]Note that this is not precisely the case in the problems of interest here, in which a continuous-time chaotic system exponentially approaches an attractor.

tions in systems engineering and finance. This approach is well-suited for calculating an unbiased estimate of the infinite time-averaged value of a statistic, as it is specifically designed to find an estimate of the average with minimum uncertainty. Take $\{X_1, X_2, \ldots, X_N\}$ as a dataset modelled as a realization of a random process with N samples; the initial transient of this dataset is estimated via this approach by solving the following optimization problem:

$$\hat{k} = \underset{1 \leq k \leq \frac{N}{2}}{\operatorname{argmin}} \frac{1}{(N-k-1)^2} \sum_{i=k+1}^{N} (X_i - \bar{X}_{k,N})^2 \text{ where } \bar{X}_{k,N} = \frac{1}{N-k} \sum_{i=k+1}^{N} X_i. \tag{6.4}$$

That is, this approach selects the number of initial samples $\hat{k}$ to set aside in order to minimize an estimate of $\sigma^2/(N-\hat{k})$, which is (within a multiplicative constant) an estimate of the squared uncertainty, $\varepsilon^2_{N-\hat{k}}$, of the averaged value of the remainder of the dataset. This optimization problem can be solved in $O(N^2)$ flops using a brute force method, or with $O(N)$ flops using a more advanced optimization algorithm (see, e.g., [118, 119]).

6.2 Estimation of the averaging error

In this section, we present a new multiscale technique to estimate the averaging error of a dataset modelled as a stationary ergodic random process x_i. Define an additional random variable, referred to as the *sample mean* y_s, such that

$$y_s = \frac{1}{s} \sum_{i=1}^{s} x_i,$$

and identify the *mean* μ, *variance* σ^2, and *autocorrelation function* $\rho(k)$ such that

$$\mu = \mathrm{E}[x_i] = \mathrm{E}[y_s], \quad \sigma^2 = \mathrm{E}[(x_i - \mu)^2], \quad \rho(k) = \frac{\mathrm{E}[(x_i - \mu)(x_{i+k} - \mu)]}{\sigma^2}. \tag{6.5}$$

The *expected squared averaging error*, defined as the expected value of the square of the deviation of the sample mean y_s from the true mean μ, is given (see §1 of [107]) by

$$\varepsilon_s^2 = \mathrm{E}[(y_s - \mu)^2] = \frac{\sigma^2}{s}\Big[1 + 2\sum_{k=1}^{s-1}(1 - \frac{k}{s})\,\rho(k)\Big]. \tag{6.6}$$

Note that $\varepsilon_s^2 \to 0$ as $s \to \infty$, as the process is ergodic, and that (6.6) reduces to (6.1a) in the limit that the samples are i.i.d. It follows immediately from (6.6) that

$$\mathrm{E}[y_s^2] = \mu^2 + \varepsilon_s^2 = \mu^2 + \frac{\sigma^2}{s}\Big[1 + 2\sum_{k=1}^{s-1}(1 - \frac{k}{s})\,\rho(k)\Big]. \tag{6.7}$$

Now consider a sequence of N statistically stationary random variables, x_1 to x_N. An unbiased estimate of μ can be developed from this sequence leveraging the following definition of the *shifted sample means*

$$m_{\ell,\ell+s} = \frac{1}{s}\sum_{i=\ell+1}^{\ell+s} x_i \quad \text{for} \quad \ell = 0, 1, \ldots, N - s, \tag{6.8a}$$

each of which is considered as a random variable with a distribution identical to that of y_s. Note that the shifted sample means are not independent. In practice, in the spirit of a batch means method, we will consider only those shifted sample means $m_{\ell,\ell+s}$ corresponding to non-overlapping blocks such that $\ell \in L_s = \{0, s, 2\,s, \ldots, (p_s - 1)\,s\}$ where $p_s = \lfloor \frac{N}{s} \rfloor$. Define also the *mean-squared shifted sample mean*, $\bar{m}_s^2$, as the mean-squared value of $m_{\ell,\ell+s}$ for

these p_s nonoverlapping blocks,

$$\bar{m}_s^2 = \frac{1}{p_s} \sum_{\ell \in L_s} m_{\ell,\ell+s}^2; \tag{6.8b}$$

the random variable $\bar{m}_s^2$ has the same expected value as y_s^2, but reduced variance.

We will denote[2] $\{X_1, X_2, \ldots, X_N\}$ as a dataset modelled as a realization, of length N, of the random process x_i described above. Corresponding realization values of the sample mean, the shifted sample means, and the mean-squared shifted sample mean are denoted by Y_s, $M_{\ell,\ell+s}$, and $\bar{M}_s^2$, respectively. The value of $\bar{M}_s^2$ computed from this dataset provides an estimate of the expected mean of y_s^2 that is accurate for values of s that are small enough that there are several blocks to average over; we thus only consider in this work the mean-squared shifted sample mean $\bar{m}_s^2$, and its realization value $\bar{M}_s^2$, for $s \leq q_N$ where $q_N = \lfloor \sqrt{N} \rfloor$.

We now define a model quantity $\hat{\varepsilon}_s^2$, in an analogous form as the expected squared averaging error ε_s^2 in (6.6), such that

$$\hat{\varepsilon}_s^2 = \frac{\hat{\sigma}^2}{s}\Big[1 + 2 \sum_{k=1}^{s-1} (1 - \frac{k}{s})\, \hat{\rho}(k; \hat{\theta})\Big], \tag{6.9}$$

where $\hat{\sigma}$ is an model (i.e., an estimate) of the variance σ, and $\hat{\rho}(k; \hat{\theta})$ is a model[3] of the autocorrelation function $\rho(k)$, with its adjustable model parameters assembled into the vector $\hat{\theta}$:

$$\hat{\rho}(k; \hat{\theta}) = \sum_{i=1}^{m} \hat{A}_i\, \hat{\tau}_i^k \quad \text{where} \quad \hat{\theta} = [\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_m, \hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_m], \tag{6.10}$$

[2] That is, random variables in this work are indicated by lowercase letters, and corresponding realizations of these random variables are indicated by uppercase letters.

[3] Models of the autocorrelation function of various statistics of interest, in a number of chaotic systems of interest, have been studied broadly (e.g., autocorrelations of some statistics in turbulent flows are discussed in [97, 120]). The autocorrelation model given in (6.10) is typical in such studies.

where the feasible domain Ω for the $\hat{\theta}$ parameters is the linearly constrained domain

$$0 \leq \hat{\tau}_i < 1, \quad \sum_{i=1}^{m} \hat{A}_i = 1. \tag{6.11}$$

We now denote, by $\{\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N\}$, optimized values of $\{\hat{\theta}, \hat{\sigma}, \hat{\mu}\}$ based on the sequence $\{x_1, \ldots, x_N\}$ of length N. These optimized values are determined by solving the following optimization problem:

$$\{\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N\} = \operatorname{argmin} f(\hat{\theta}, \hat{\sigma}, \hat{\mu}) = \sum_{s=1}^{q_N} \left[g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})\right]^2 \quad \text{where} \tag{6.12a}$$

$$g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu}) = \hat{\mu}^2 + \frac{\hat{\sigma}^2}{s}\Big[1 + 2 \sum_{k=1}^{s-1} (1 - \frac{k}{s})\, \hat{\rho}(k; \hat{\theta})\Big] - \bar{m}_s^2, \tag{6.12b}$$

where $\bar{m}_s$ is derived from the sequence $\{x_1, \ldots, x_N\}$ via (6.8), while imposing that the last term in the sum in (6.12a) vanishes, $g_{q_N}(\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N) = 0$, in addition to the feasibility of the parameters of the autocorrelation model, $\hat{\theta} \in \Omega$ [see (6.11)], as well as $\hat{\sigma} \geq 0$ and $\hat{\mu} \geq 0$. In other words, we seek to find the best model parameters, $\{\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N\}$, such that the expression for $\mathrm{E}[y_s^2]$ in (6.7), leveraging the model values $\hat{\mu}$ and $\hat{\sigma}$ and the model of the autocorrelation in (6.10), $\hat{\rho}(k; \hat{\theta})$, exactly matches the unbiased estimate $\bar{m}_s^2$ of y_s^2 at $s = q_N = \lfloor \sqrt{N} \rfloor$, while the sum of the squares of the mismatch of these quantities over all the batch lengths $1 \leq s < q_N$ is minimized. That is, the tuning of the available parameters in the model, $\{\hat{\theta}, \hat{\sigma}, \hat{\mu}\}$, is performed in such a way as to accurately match the model [given in (6.9)] of the expected squared averaging error [given by (6.6)], at a range of different timescales s, to the information available in the sequence $\{x_1, \ldots, x_N\}$; we thus refer to the approach developed as a *multiscale* fit.

In this work, for any given realization of the sequence, $\{X_1, \ldots, X_N\}$, the optimiza-

Algorithm 6.1 Estimation of the expected averaging error $\varepsilon_N^2 = \mathrm{E}[(y_N - \mu)^2]$ of a set of data $\{X_1, X_2, \ldots, X_N\}$ modelled as a realization of a stationary random process $\{x_1, x_2, \ldots, x_N\}$.

1: **for** each $s \in \{1, 2, \ldots, \lfloor \sqrt{N} \rfloor\}$ **do**
2: Calculate $M_{\ell,\ell+s}$ via (6.8a) for all $\ell \in L = \{0, s, 2\,s, \ldots, (\lfloor \frac{N}{s} \rfloor - 1)s\}$.
3: Compute the mean-squared value $\bar{M}_s^2$ of the $M_{\ell,\ell+s}$ for all $\ell \in L$ via (6.8b).
4: Solve the optimization problem (6.12) to find the optimized model parameters $\{\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N\}$.
5: Generate an estimate of the averaging error, $\hat{\varepsilon}_N^2$, with (6.9), using the autocorrelation model given by $\hat{\rho}(k; \hat{\theta})$ in (6.10), implementing the optimized parameter values $\{\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N\}$ determined in step 5.

tion problem defined by (6.12) is solved using SNOPT [121], which is an advanced Sequential Quadratic Programming (SQP) method. Though the application of such a solver to a problem of this form is entirely straightforward, the optimization problem given in (6.12) is nonconvex, and thus SNOPT might only find a local minimum. The resulting framework for estimating the averaging error is summarized in Algorithm 6.1.

Analytical expressions for the derivatives of $f(\hat{\theta}, \hat{\sigma}, \hat{\mu})$ and $g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})$ are useful in the optimization process. The derivatives of the these functions are given as follows:

$$\nabla f(\hat{\theta}, \hat{\sigma}, \hat{\mu}) = 2 \sum_{s=1} g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})\, \nabla g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu}),$$

$$\frac{\partial g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})}{\partial \hat{\mu}} = 2\,\hat{\mu}, \quad \frac{\partial g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})}{\partial \hat{\sigma}} = \frac{2\,\hat{\sigma}}{s}\Big[1 + 2 \sum_{k=1}^{s-1}(1 - \frac{k}{s}) \sum_{i=1}^{m} \hat{A}_i \hat{\tau}_i^k\Big],$$

$$\frac{\partial g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})}{\partial \hat{A}_i} = \frac{\hat{\sigma}^2}{s}\Big[2 \sum_{k=1}^{s-1}(1 - \frac{k}{s})\, \hat{\tau}_i^k\Big], \quad 1 \le i \le m,$$

$$\frac{\partial g_s(\hat{\theta}, \hat{\sigma}, \hat{\mu})}{\partial \hat{\tau}_i} = \frac{\hat{\sigma}^2}{s}\Big[2 \sum_{k=1}^{s-1} \hat{A}_i\, k\, (1 - \frac{k}{s})\, \hat{\tau}_i^{k-1}\Big], \quad 1 \le i \le m.$$

6.3 Analysis of the estimator

We now analyze various properties of the new estimation technique, presented in §6.2 and summarized in Algorithm 6.1, applied to a stationary random process $\{x_1, x_2, \dots\}$. The mean μ, variance σ^2, and autocorrelation $\rho(k)$ of the random process x_i considered are defined as in (6.5).

If the autocorrelation function $\rho(k)$ is summable, then it follows from (6.6) that the expected squared averaging error, ε_s^2, approaches zero like the reciprocal square root of s times a constant Q, that is,

$$\lim_{s\to\infty} s\,\varepsilon_s^2 = Q \quad \text{for finite } Q. \tag{6.13}$$

It is natural to seek a UQ method that satisfies the same property; this notion is made precise by the following definition and theorem.

Definition 30. *The random process a_s is called an* asymptotically unbiased *estimate of the expected squared averaging error ε_s if*

$$\lim_{s\to\infty} s\,(\mathrm{E}[a_s] - \varepsilon_s^2) = 0. \tag{6.14}$$

Theorem 12. *The random process $\hat{\varepsilon}_s^2$ in* (6.9)*, the N'th element of which is obtained by implementing Algorithm 6.1 on the first N elements of the random process x_i, provides an asymptotically unbiased estimate of ε_s^2 in* (6.6).

Proof. The model of the autocorrelation function (6.10), with parameters as optimized by Algorithm 6.1, is necessary summable (since the $\tau_i < 1$ for all i). Denote $\hat{\varepsilon}_{s,N}^2$ as the model, given by (6.9), of the expected squared averaging error over a sequence of length s, implementing the optimized parameters $\{\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N\}$ derived from a sequence of length N.

It follows that

$$\lim_{s\to\infty} s\,\hat{\varepsilon}^2_{s,N} = \hat{Q}_N, \quad \text{for finite } \hat{Q}_N. \tag{6.15}$$

By construction, the parameters $\hat{\theta}_N, \hat{\sigma}_N, \hat{\mu}_N$, $\hat{Q}_N$, and $\hat{\varepsilon}_{q_N,N}$ are random variables, as they are derived from the random process x_i. Moreover, these variables are obtained by solving the optimization problem (6.12); therefore, the equality constraint of the optimization problem, $g_{q_N}(\hat{\theta}, \hat{\sigma}, \hat{\mu}) = 0$ where $q_N = \lfloor \sqrt{N} \rfloor$, must be satisfied:

$$\hat{\mu}^2_N + \hat{\varepsilon}^2_{q_N,N} - \bar{m}^2_{q_N,N} = 0. \tag{6.16}$$

Since $\bar{m}^2_{q_N,N}$ is unbiased, $\mathrm{E}[\bar{m}^2_{q_N,N}] = \mu^2 + \varepsilon^2_{q_N}$. Taking the expected value of (6.16), it follows that

$$\mathrm{E}[\hat{\mu}^2_N] + \mathrm{E}[\hat{\varepsilon}^2_{q_N,N}] - \mu^2 - \varepsilon^2_{q_N} = 0. \tag{6.17}$$

Multiplying the above equation by q_N and rearranging gives

$$q_N(\mathrm{E}[\hat{\mu}^2_N] - \mu^2) + (\mathrm{E}[q_N\hat{\varepsilon}^2_{q_N,N}] - q_N\varepsilon^2_{q_N}) = 0. \tag{6.18}$$

Thus, taking the limit as $N \to \infty$ (and, therefore, $q_N \to \infty$) and substituting (6.13) and (6.15), it follows that

$$\lim_{N\to\infty} q_N(\mathrm{E}[\hat{\mu}^2_N] - \mu^2) + \mathrm{E}[\hat{Q}_N] - Q = 0.$$

Since Q and $\hat{Q}_N$ are bounded,

$$\lim_{N\to\infty} \mathrm{E}[\hat{\mu}_N^2] = \mu^2, \quad \lim_{N\to\infty} \mathrm{E}[\hat{Q}_N] = Q,$$

$$\lim_{N\to\infty} N(\mathrm{E}[\hat{\varepsilon}_N^2] - \varepsilon_N^2) = \lim_{N\to\infty} \mathrm{E}[\hat{Q}_N] - Q = 0. \qquad \square$$

We have thus established that the implementation of Algorithm 6.1 on any realization of a stationary random process x_i results in an unbiased estimate of the averaging error. This is a valuable property of the present method for estimating the averaging error, as it implies that the uncertainty quantification does not have any systematic error in the limit of large N. Note that certain other leading methods for uncertainty quantification, such as that developed in [107], based on a maximum likelihood formulation, do not share this valuable property.

6.4 Numerical simulations

We now apply the algorithm developed in section 6.2 for estimating the averaging error to three different datasets generated as follows:

1. A synthetic autoregressive process of order six, AR(6).

2. Statistics of the kinetic energy of the Kuramoto-Sivashinsky (KS) equation.

3. Statistics of the TKE from a DNS of a turbulent channel flow at $Re_\tau = 180$.

For the purpose of comparison, in all three cases, the averaging error is estimated using the maximum likelihood approach.

Also, for comparison, the expected squared averaging error ε_s^2, given by (6.6), is calculated based on accurate values of μ, σ^2, and $\rho(k)$. For the dataset generated by AR(6),

the true values of μ, σ^2, and $\rho(k)$ are available, so the true value of ε_s^2 is directly computable. To develop a "truth model" for the UQ of the other datasets, since analytical expressions for μ, σ^2, and $\rho(k)$ are not available in these cases, we simply apply the algorithm developed above for very large N (at least 30 times larger than the maximum value of N considered in the plots). This approach provides a very large number of samples to average over when approximating ε_s^2 numerically.

6.4.1 Autoregressive model

We first apply the new UQ method developed in §6.2 to a dataset generated by an autoregressive (AR) model of the general form

$$x_i = \sum_{k=1}^{n} \alpha_k \, x_{i-k} + \epsilon_i; \tag{6.19}$$

for the present study, we take $\epsilon_i = \mathcal{N}(0, \sigma_\epsilon^2)$. After an initial transient (related to the specified n initial values of x_i) has passed, this system is statistically stationary, with a mean of $\mu = 0$ and, defining the unnormalized autocorrelation function $\gamma(k) = \sigma^2 \rho(k) = \mathrm{E}\{(x_i - \mu)(x_{i+k} - \mu)\}$, the values of $\gamma(k)$ related (see [107]) by

$$\gamma(0) = \sum_{j=1}^{n} \alpha_j \gamma(j) + \sigma_\epsilon^2, \tag{6.20a}$$

$$\gamma(h) = \sum_{j=1}^{n} \alpha_j \gamma(h-j) \quad \text{for} \quad h > 0. \tag{6.20b}$$

Noting that $\gamma(k) = \gamma(-k)$, the first $n+1$ values of γ may be determined by solving the linear system of equations, known as the Yule-Walker equations, given by (6.20a) together with (6.20b) for $h = 1, \ldots, n$; additional values of $\gamma(k)$ are then given directly by (6.20b). Note

further that $\sigma^2 = \gamma(0)$, and $\rho(k) = \gamma(k)/\sigma^2$.

In the simulations reported here, we consider an AR(6) process (that is, we take $n = 6$) with $\epsilon_i = \mathcal{N}(0, \sigma_\epsilon^2)$ where $\sigma_\epsilon^2 = 0.1$, and

$$\alpha_1 = 3.1378,\ \alpha_2 = -3.9789,\ \alpha_3 = 2.6788,\ \alpha_4 = -1.0401,\ \alpha_5 = 0.2139,\ \alpha_6 = -0.0133.$$

The poles of the difference equation corresponding to this AR(6) model are given by

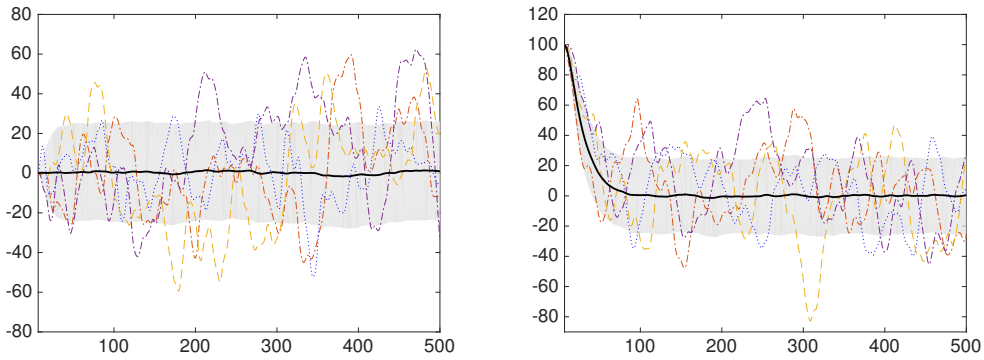
$$\{0.1,\ 0.95,\ 0.8,\ 0.7,\ 0.25 + \sqrt{3}/2,\ 0.25 - \sqrt{3}/2\}.$$

After the initial transient has passed, the system in (6.20) reveals that the standard deviation $\sigma = 24.97$, and that the autocorrelation function $\rho(k)$, for $k = 0, 1, 2, \ldots$, is:

$$\rho(k) = \{1,\ 0.9967,\ 0.9870,\ 0.9716,\ 0.9516,\ 0.9277,\ 0.9010,\ 0.8722,\ 0.8418, \ldots\}.$$

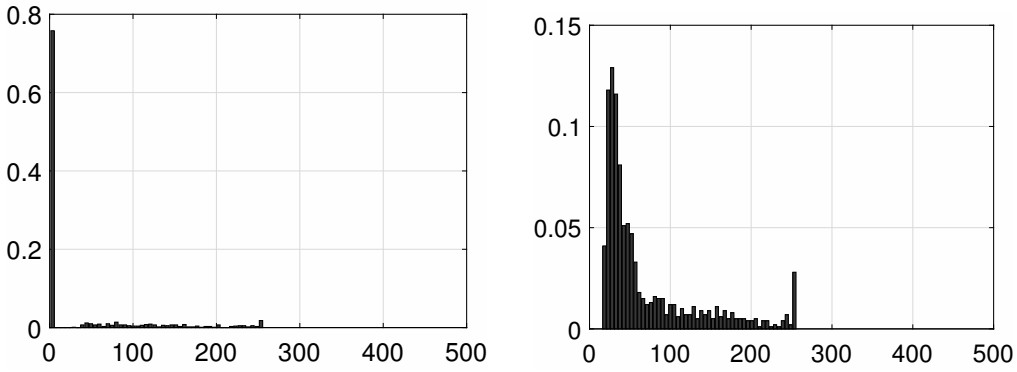
The typical behavior of the AR(6) model described above is illustrated in Figure 6.1.

Figure 6.2 illustrates the behavior of the transient time estimation method reviewed in §6.1.1 on the ensemble of 1000 simulations summarized in Figure 6.1. Initialization with $x_{-5} = \ldots = x_0 = 0$, as illustrated in Figure 6.2a, shows that, for about 75% of the ensemble members considered, the minimization problem given by (6.4) results simply in $\hat{k} = 1$. This is entirely to be expected, as the transient estimation method implemented in this work is *not* based on a second-order stationarity condition, but rather simply on the minimization of the average squared value of $(X_i - \bar{X})$ over the remaining samples. Thus, even though the AR(6) system is clearly not statistically stationary in the first few samples in this case, the particular transient estimation method implemented is insensitive to this



a Initialization: $x_{-5} = \ldots = x_0 = 0$. b Initialization: $x_{-5} = \ldots = x_0 = 100$.

Figure 6.1: Simulation of the AR(6) model described in §6.4.1, evolving away from different initial values of x_i as indicated. (dot-dashed lines) Representative realizations. (solid line) Ensemble mean and (gray region) ensemble mean +/− ensemble variance, computed over 1000 ensemble members.



a Initialization as in Figure 6.1a. b Initialization as in Figure 6.1b.

Figure 6.2: Histogram of the transient time estimates (see §6.1.1) for the AR(6) model described in §6.4.1, computed over 1000 ensemble members.

fact.

Initialization with $x_{-5} = \ldots = x_0 = 100$, as illustrated in Figure 6.2b, shows a much more typical behavior of the transient time estimation method selected for the problems of interest in this work. In this case, the minimization problem given by (6.4) results in an average value of $\hat{k} = 40$, which is very nearly the value one would select by eye given the (very significant) advantage of hindsight, as embodied by the ensemble average results depicted in Figure 6.1b. This is indeed remarkable, as each transient time calculation is based solely on an individual ensemble member, each of which has significant random fluctuations associated with it (see the dot-dashed curves of Figure 6.1b).

Figure 6.3 illustrates the performance of Algorithm 6.1 on an ensemble of 30 datasets obtained via simulation of the AR(6) model described above, taking $x_{-5} = \ldots = x_0 = 0$ and $\hat{k} = 0$, with realization lengths of $N = \{2^{7:14}\}$. For the purpose of comparison, we have also estimated the averaging error using the maximum likelihood approach applied to an AR(3) model; to facilitate a fair comparison, the same number of model parameters is used for both methods. We also compare with the expected squared averaging error given by (6.6), with the exact formulae for μ, σ, and ρ as determined by solution of (6.20).

It is clearly evident for moderate to large realization lengths, $N \gtrsim 2000$, that the performance of the uncertainty quantification method given by Algorithm 6.1 is remarkably better than that given by the MLE-based approach. In particular, it is distinctly evident that the estimates given by Algorithm 6.1 are asymptotically unbiased (see Definition 30), whereas the estimates generated by the MLE-based approach are not (as the underlying AR(3) model used in the MLE-based approach can not entirely capture the dynamics of the AR(6) system). For small realization lengths N, the performance of the estimators are similar.

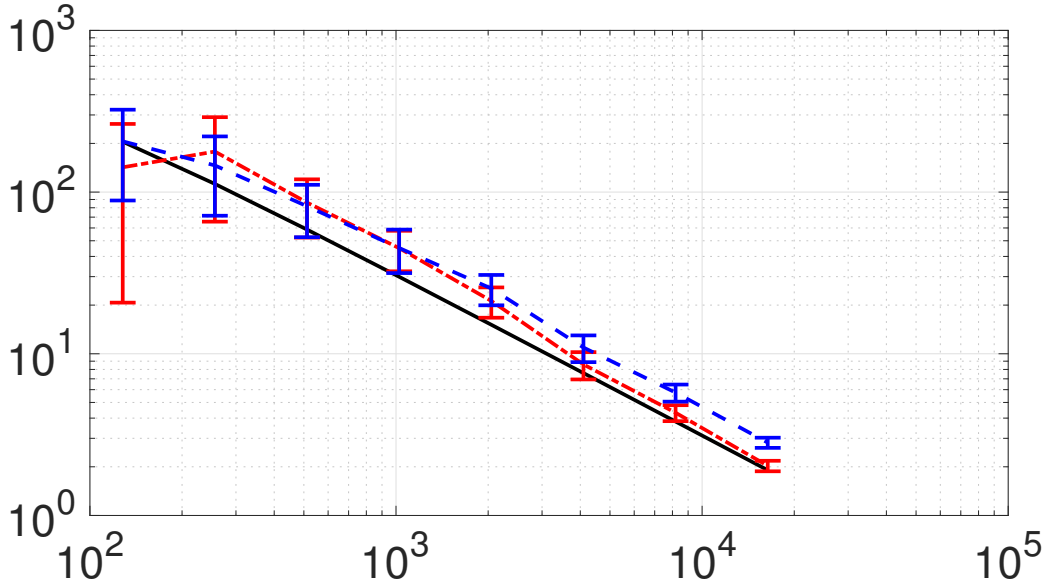


Figure 6.3: Implementation of UQ on AR(6) data. Implementation of Algorithm 6.1 and the MLE-based UQ approach on an ensemble of 30 simulations of the AR(6) model described in §6.4.1. (horizontal axis) Averaging length N, (vertical axis) averaging error ε_N. (red dotted-dashed line) Ensemble average and (red error bars) ensemble variance of the estimate of the averaging error given by Algorithm 6.1. (blue dashed line) Ensemble average and (blue error bars) ensemble variance of the estimate of the averaging error given by the MLE-based approach, using an AR(3) model. (sold black line) Actual averaging error, based on (6.6) with the true values of μ, σ^2, and $\rho(k)$.

6.4.2 Kuramoto-Sivashinsky equation

We next apply the new UQ method to a dataset derived from a simulation of the Kuromoto-Sivashinsky (KS) equation,

$$u_t + u_{xxxx} + u_{xx} + u\,u_x = 0 \quad \text{for} \quad 0 \le x < L, \tag{6.21}$$

with periodic boundary conditions $u(0) = u(L)$. The statistic we consider in this work is the spatially-averaged value of the energy, defined as

$$e(t) = \frac{1}{L}\int_0^L u^2(x,t)\,dx. \tag{6.22}$$

The KS PDE is simulated in this work using a dealiased pseudospectral method for computing spatial derivatives, and a low-storage Implicit-Explicit Runge-Kutta (IMEXRK) scheme [122, 123] for marching in time.

In the simulations reported here, we take $L = 200$, $N_x = 512$, $\Delta x = L/N_x \approx 0.391$, and $\Delta t = 0.2$. The initial field is taken as

$$u(x,0) = \sin(0.5\,\pi\ x) + \sin(0.85\,\pi\,x) + 0.2\,v, \quad v = \mathcal{N}(0,1). \tag{6.23}$$

After the initial transient has passed, the KS system defined above approaches a chaotic attractor, as indicated in Figure 6.4a.

The transient identification method reviewed in §6.1.1 is again implemented to detect and set aside the initial transient in the dataset. A typical estimate of the transient time is illustrated in Figure 6.4; after setting aside this initial transient, the remainder of the dataset appears to be approximately statistically stationary.

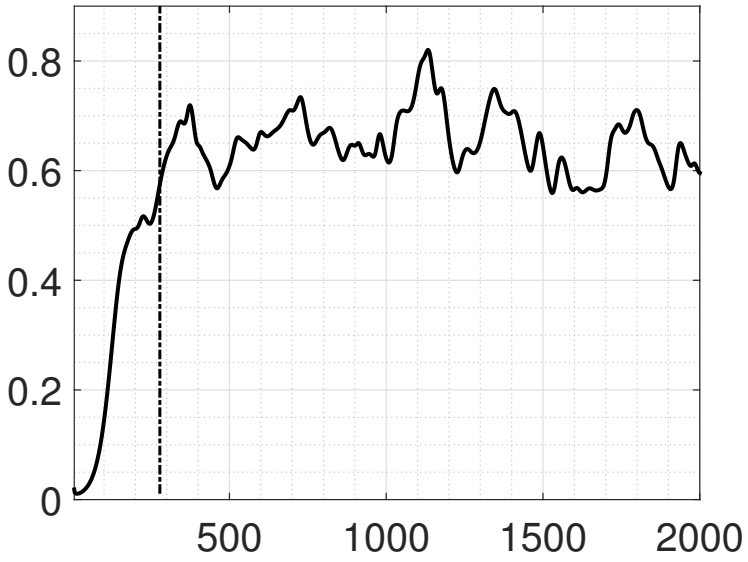


Figure 6.4: Evolution of (solid line) the kinetic energy in a simulation of the KS model described in §6.4.2 versus the number of timesteps taken, including (vertical dashed line) the identification of the initial transient.

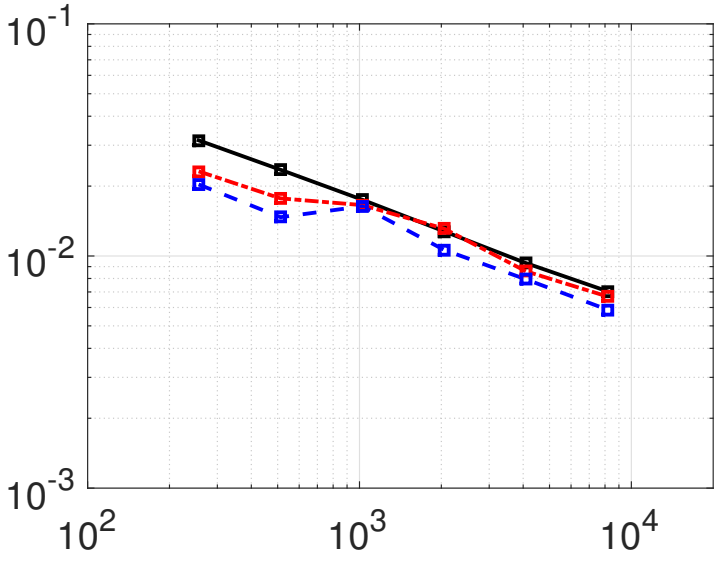


Figure 6.5: Implementation of UQ on KS dataset. Implementation of Algorithm 6.1 and the MLE-based UQ approach on a single simulation of the KS model described in §6.4.2. Horizontal axis is the number of samples taken, with one sample per timestep. Line types are identical to those in Figure 6.3, with the error bars removed because only a single simulation is shown. Note that accurate estimates, not exact values, are used for μ, σ^2, and $\rho(k)$ when computing the actual averaging error.

Sampling every timestep after the initial transient (see Figure 6.4) is set aside, the averaging error was computed with realization lengths of $N = \{2^8 : 2^{13}\} = \{256 : 8192\}$, incrementing by powers of two. Note that, since we take $\Delta T = 0.2$, this is equivalent to taking $T = 0.2 * N = \{50 : 3268\}$ time units of the original KS equation. Again, the averaging error is estimated using Algorithm 6.1 and the MLE-based approach, with an AR(3) model incorporated. We also compare with the expected squared averaging error given by (6.6), with the accurate values for μ, σ, and ρ as determined from a simulation 30 times longer than the the longest simulation reported here.

As observed in Figure 6.5, the performance of the UQ method developed here is significantly improved as compared with the MLE-based approach, especially as the number of samples is increased.

6.4.3 Navier-stokes equations

Finally, we apply the UQ method to a dataset generated by a DNS of a low Reynolds number incompressible 3D turbulent channel flow (see, e.g., [124, 125]). Periodic boundary conditions are applied in the streamwise direction, x, and the spanwise direction, z; homogeneous Dirichlet boundary conditions on the velocity are applied at the walls in the wall-normal direction, y.

Following [120], the incompressible Navier-Stokes equation is implemented in a 2-variable formulation of the wall-normal components of velocity and vorticity (other velocity, vorticity, and pressure components may be computed from these two components as needed). The simulation, which used the code developed in [126], used a dealiased pseudospectral method for computing spatial derivatives in the x and z directions, and the compact finite difference method [127] for computing spatial derivatives in the y direction.

The CN/RKW3 method [128] was used for time integration.

In the simulations reported here, we consider a spatial domain with $0 \leq x \leq 2\pi$, $0 \leq y \leq 2$, and $0 \leq z \leq 2\pi$, a grid of $N_x = 128$, $N_y = 64$, and $N_z = 128$, Reynolds number $Re_\tau = 180$, and timesteps of $\Delta t = 0.01$. The simulation is performed for $N = 10^5$ timesteps, and the statistic that is analyzed is turbulent kinetic energy (TKE).

As indicated in Figure 6.6, the transient identification process is completely analogous to that in the KS case.

Sampling every timestep after the initial transient (see Figure 6.6) is set aside, the averaging error was computed with realization lengths of $N = \{2^9 : 2^{12}\} = \{512 : 4096\}$, incrementing by powers of two. Note that, since we take $\Delta T = 0.01$, this is equivalent to taking $T = 0.01 * N = \{5.12 : 40.96\}$ time units of the original NS equation. Again, the averaging error is estimated using Algorithm 6.1 and the MLE-based approach, with an AR(18) model incorporated. We also compare with the expected squared averaging error given by (6.6), with the accurate values for μ, σ, and ρ as determined from a simulation much longer than the the longest simulation reported here.

Again, as observed in Figure 6.7, the performance of the UQ method developed here is seen to be significantly improved as compared with the MLE-based approach, especially as the number of samples is increased.

6.5 Conclusions

A new approach has been developed to quantify the uncertainty associated with finite-time-average approximations of infinite-time-average statistics of statistically stationary ergodic processes. For applications of this new UQ approach that are derived from continuous-time chaotic systems like turbulent flows, an adequate sampling inter-

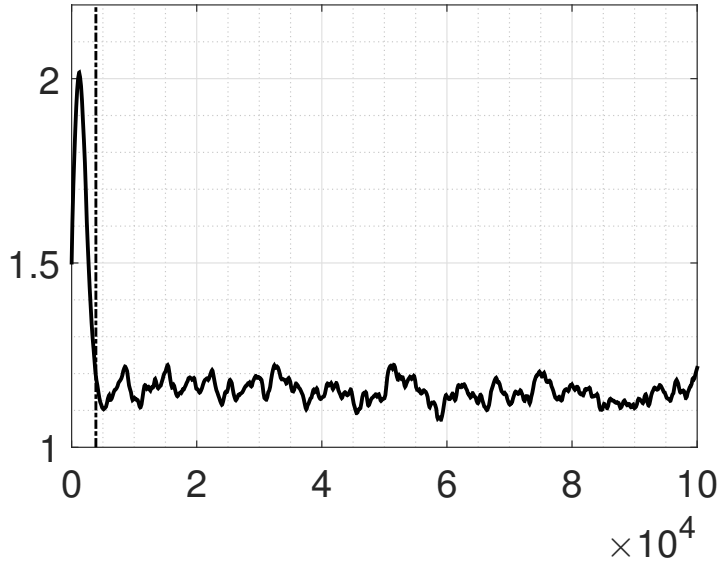


Figure 6.6: Evolution of (solid line) the turbulent kinetic energy in a simulation of the $Re_\tau = 180$ channel flow model described in §6.4.3 versus the number of timesteps taken, including (vertical dashed line) the identification of the initial transient.

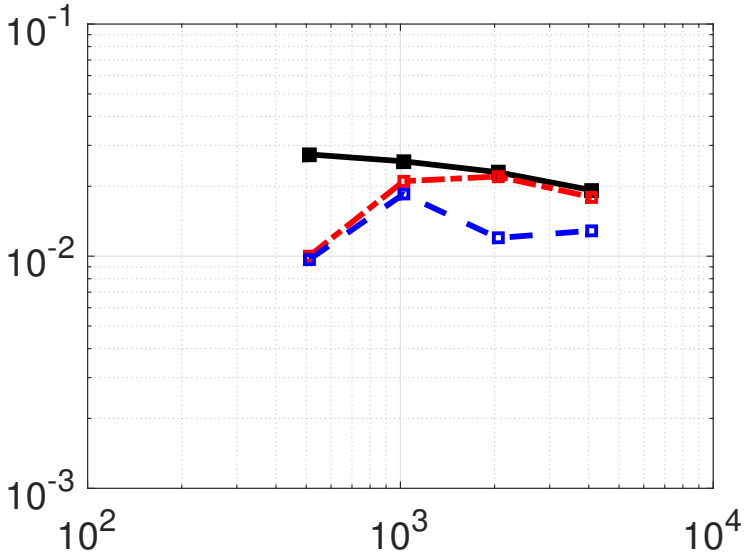


Figure 6.7: Implementation of UQ for the simulation of channel flow. Implementation of Algorithm 6.1 and the MLE-based UQ approach on a single simulation of the $Re_\tau = 180$ channel flow model described in §6.4.3. Horizontal axis is the number of samples taken, with one sample per timestep. Line types are identical to those in Figures 6.5 and 6.3.

val h is identified in (6.2), and an effective method for removing the initial transient from the dataset is reviewed in §6.1.1. Chapter 5 illustrates how an effective UQ approach of

this sort can be directly leveraged for maximally-efficient derivative-free optimization of infinite-time-averaged statistics of chaotic systems which depend upon a handful of adjustable parameters.

The new UQ method is presented in §6.2 and analyzed mathematically in §6.3. This analysis reveals that, for long simulations, the UQ so determined is asymptotically unbiased; this important property is not guaranteed by various competing UQ methods, such as the leading method developed in [107], which based on a maximum likelihood formulation.

The new UQ method is tested in §6.4 on datasets generated by an AR(6) process, by the Kuramoto-Sivashinsky equation, and by a low-Reynolds number turbulent channel flow DNS. Results are compared with both the leading UQ approach developed in [107], as well as the expected deviation of the sample mean y_s from the true mean μ, as quantified by (6.6), based on accurate values of the true mean μ, the variance σ^2, and the autocorrelation $\rho(k)$.

It is observed that the method developed here has a significant improvement from that provided by the approach in [107], especially as the realization length N is increased.

Acknowledgment

This chapter is in the preparation for submission to SIAM Journal of Uncertainty Quantification as the following article: P. Beyhaghi, S. Alimohammadi, T.R. Bewley, "A multiscale, asymptotically unbiased approach to uncertainty quantification in the numerical approximation of infinite time-averaged statistics." The dissertation author is the primary investigator and author of this paper.

Chapter 7

Simulation-based optimization of the hydrofoil of a flying catamaran

7.1 Introduction

Hydrofoils play an increasingly important role in the design of high-performance sailboats and catamarans. The 34th America's Cup (San Francisco, 2013) highlighted the importance of efficient hydrofoil design, and the Class Rule for the 35th America's Cup (Bermuda, 2017), to be held on 48-foot catamarans, even further emphasizes their importance: as hydrofoil design is now one of the few features of the sailboat design left open in the competition rules. Hydrofoils also play an increasingly important role on many sailboats outside of high-profile America's Cup races, including the Hydroptere (a large, fast trimaran), the International Moth class of small, fast sailing hydrofoils, and foil boards, which are now quite popular for high-speed kiteboarding.

Accurate hydrofoil performance assessment and design optimization is, in general, a time-consuming and computationally expensive undertaking. Challenges are present in both the physical and the numerical modeling: complex physics including boundary layers, free-surface effects, and cavitation generally require high-fidelity numerical codes and large computational resources to assure accurate results. Direct Numerical Simulations (DNS), Large Eddy Simulations (LES), and Reynolds-Averaged Navier-Stokes (RANS) simulations, however, are often unaffordable in the design phase, which often requires a significant number of design iterations. Approximate performance estimates derived from computationally inexpensive models, such as vortex-lattice methods, are generally sufficient for tuning the handful of adjustable parameters characterizing such designs. Numerical models of this sort are already well developed and used extensively for the design of rigid wings [129], and are applied here for the related problem of hydrofoil optimization.

In this chapter, we consider the application of our new derivative-free optimization algorithm dubbed Δ-DOGS(C) to the design of a 3D hydrofoil with seven adjustable parameters. The computationally inexpensive vortex-lattice model implemented in AVL (the Athena Vortex Lattice code; see [130]) is used to compute the lift and drag coefficients of the hydrofoil.

This chapter is organized as follows. In §7.2, we describe the AVL model, discuss its limitations, and presents a careful validation based on experimental results from the literature. Next, §7.3 describes the parametrization of the hydrofoil used in the present optimization, and the reasoning behind the particular choice of parameters used. The results of our optimization study are presented in §7.4, and conclusions are drawn in §7.5.

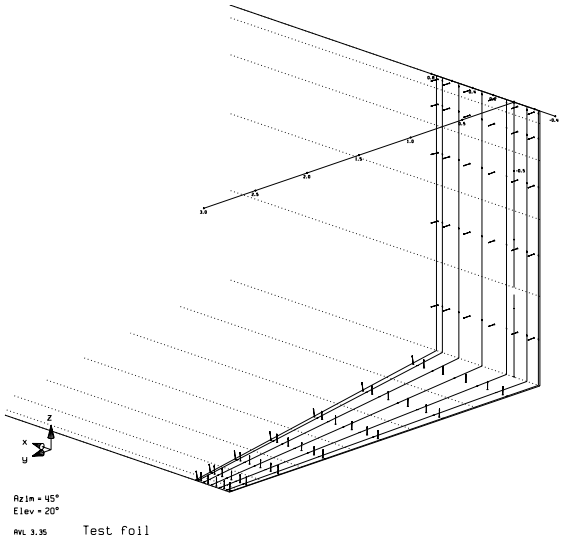


Figure 7.1: Illustration of the AVL model. AVL model: the foil is discretized by vortices along the spanwise direction and extending to infinity past the foil (dotted line).

7.2 Foil model and validation

The numerical model at the basis of our optimization is AVL [130]. AVL computes the inviscid lift and drag coefficients based on a vortex-lattice discretization of the foil, see Figure 7.1 and [131] for a more detailed description. The viscous drag is computed based on the local lift coefficient C_L from the foil sections' $C_D(C_L)$ curve, where C_D is the drag coefficient [132].

AVL implements a "free surface" boundary condition in the form of a constant-pressure, constant-height horizontal plane: this is known to be a good approximation of a real free surface in the limit of high Froude numbers [133, chapt. 6], correctly capturing the changes in the inviscid lift and drag. In contrast, this approximation is unable to capture other effects associated to the presence of a free surface, e.g. wave drag, whose relative importance grows at lower Froude numbers. It is therefore an informative exercise to evaluate the model results against experimental data: to this purpose we use the water-tank measurements of a rectangular hydrofoil having an aspect ratio of 10 and a NACA64_1-412

Table 7.1: Comparison between AVL and experimental results

		AVL[a]	Exp[b]	Err[c]	%Err[d]
depth = 0.84c	$dC_L/d\alpha$	0.071	0.071	0.000	0.0%
(Fn_h = 10.48)	$\alpha(C_L = 0)$	-3.28	-3.3	0.0	0.6%
	$C_D(C_L = 0.4)$	0.01378	0.016	-0.002	13.9%
	$C_D(C_L = 0.6)$	0.02476	0.028	-0.003	11.6%
depth = 3.84c	$dC_L/d\alpha$	0.0817	0.083	-0.001	1.6%
(Fn_h = 4.97)	$\alpha(C_L = 0)$	-3.15	-3.2	0.1	1.6%
	$C_D(C_L = 0.4)$	0.01167	0.014	-0.002	16.6%
	$C_D(C_L = 0.6)$	0.01981	0.022	-0.002	10.6%

Only significant digits are reported in each column.
[a] AVL result, viscous coefficients obtained from [132]
[b] Data from [134], values are for the highest speed tested in the deepest tank
[c] AVL − Exp
[d] $\left\| \frac{\text{AVL}-\text{Exp}}{\text{Exp}} \right\|$

foil section, as reported in [134].

Table 7.1 presents numerical results and experimental measurements for the $dC_L/d\alpha$ coefficient (α being the angle of attack in degrees), the angle of attack for zero lift $\alpha(C_L = 0)$, and for the total drag coefficients C_D at two lift conditions of $C_L = 0.4, 0.6$. Two depths of submergence $h = 0.84\,c$ and $h = 3.84\,c$, c being the hydrofoil chord, are tested. The submergence depth h is measured as the distance between the undisturbed free surface and the quarter-chord location of the foil. The two configurations correspond to experimental values of the depth-based Froude number $Fn_h = \frac{U}{\sqrt{g\,h}} = 10.48$ and 4.97 respectively.

AVL results for lift (first two rows for each depth entry) are obtained by a least-square fit of a straight line through $C_L(\alpha)$ data computed for the $-3.5 < \alpha < 6.0$ range;

all significant decimal digits are reported. Drag results are obtained by running the AVL model with the desired lift as a constraint; the viscous drag is obtained from wind-tunnel measurements reported in [132].

Experimental data are obtained from figure 10 and 11 of [134]: only results for the largest water tank and the highest speed measured, equivalent to a chord-based Froude number $Fn_c = \frac{U}{\sqrt{g\,c}} \approx 10$, are used. All significant digits, intended as the decimal digits that can be reliably obtained by digitalization of the figures in [134], are reported in the table.

Agreement between numerical and experimental data for the hydrofoil lift is extremely good, with an error always lower than 2% for both depths tested. Agreement for the drag is at first sight less accurate, with the numerical model consistently underestimating the experimental drag coefficient by $\Delta C_D \approx 0.002$, or $10 - 15\%$. Nonetheless, discrepancies in the drag coefficients have to be considered with care: the fact that the difference between numerical and experimental results, at 0.002, is independent of depth, suggests that such discrepancy is not related to the lack of wave drag modeling in AVL: wave drag is expected to strongly depend on depth. We suggest three alternative causes: (i) differences in the experimental Reynolds number between [134] ($Re = 1.5 \times 10^6$) and [132] ($Re = 3.0 \times 10^6$), (ii) differences in the free stream turbulence levels in the water-tank measurement of [134] and the wind tunnel measurement of [132], causing a different boundary layer transition location and/or (iii) the way drag measurements are obtained in [134] — i.e. by first measuring the total drag of the strut-hydrofoil combination and then subtracting the drag of the strut measured alone, thus neglecting effects associated to the struct-foil interaction.

Independently of the reason behind the discrepancies in the drag coefficient, we remark that the AVL model (i) correctly captures the *lift* properties of the foil at both tested

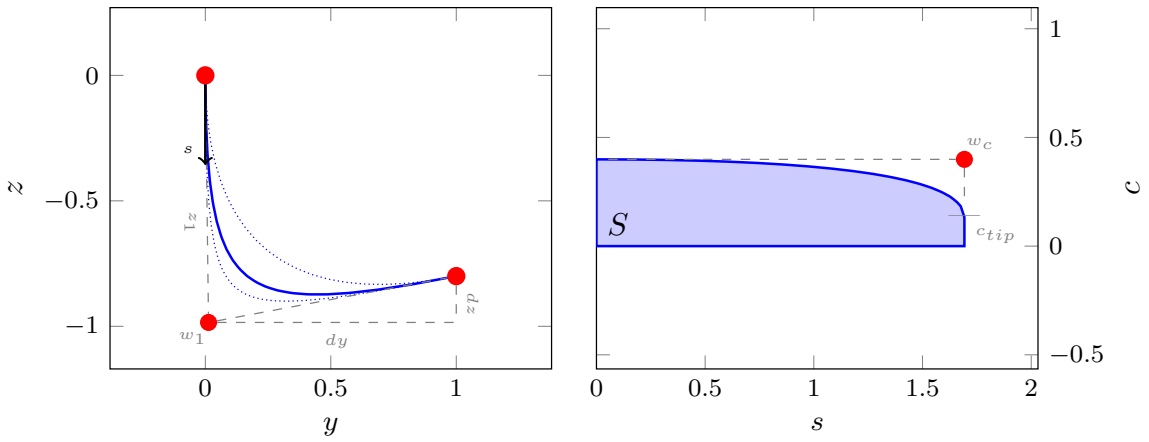


Figure 7.2: Parametrization of the foil. Left: front view in the $y-z$ plane; right: planform view. The shape of the foil is defined by seven parameters defining two rational Bezier curves: z_1, dy and dz, together with the weight w_1.

depths and (ii) correctly captures the *changes in drag* associated to changes in lift coefficient and depth. For the purpose of this work, we consider the AVL model to be validated. Further comparisons with experimental results, as well as with numerical results based on more accurate codes, will be pursued in the future.

7.3 Parametrization of the foil

Because of their computational cost, global optimization methods are limited in the number of optimization variables, or parameters, that can be optimized. The choice of the geometry parametrization is then a fundamental step of the optimization process.

The foil parametrization is visualized in Figure 7.2. The reference system used has z as the vertical coordinate, y as the horizontal crossflow coordinates and x as the horizontal stream-wise coordinate. A curvilinear coordinates s is defined along the quarter-chord of the foil, with origin at the free surface.

The main parameter is the planform area S. A minimum surface is required in order to have a physically achievable lift coefficient, representing a lower bound for the feasible

domain. The optimization process will then balance the contribution of the viscous drag — proportional to the foil surface — and of the inviscid drag — proportional to the square of the lift coefficients or, equivalently, to the inverse of the foil surface.

The other parameters govern the lift distribution by describing the foil shape in the $y - z$ plane and the chord distribution along the curvilinear s. Both the shape of the foil's quarter-chord and the chord distribution are represented using Bezier curves

$$\boldsymbol{x}(t) = \frac{\sum_{i=0}^{n} b_{i,n} \boldsymbol{P}_i w_i}{\sum_{i=0}^{n} b_{i,n} w_i} \qquad b_{i,n} = \binom{n}{i} t^i (1-t)^{n-i} \tag{7.1}$$

where $\boldsymbol{x} = (y, z)$, $\boldsymbol{P}_i = (y_i, z_i)$ are the control points marked in red in Figure 7.2 and w_i are the weights of the control points.

7.3.1 Parametrization of the solver

The two more important parameters governing the solver behavior are the estimate of a bound for the objective function value $y_0 = \max(L/D) = \max(C_L/C_D)$ and the stopping criteria δ_0, identifying a minimum distance in parameter space. Additionally, bounds for the parameters must be computed to identify the feasible domain.

The objective function bound, y_0, can be obtained using classical aerodynamic theory. The drag coefficient for a foil of aspect ratio AR and elliptic spanwise load is:

$$C_D = \frac{C_L^2}{\pi AR} + C_{Dv}(C_L) \tag{7.2}$$

where $C_{Dv}(C_L)$ is the drag coefficient for the two dimensional foil section and can be obtained from experimental data [132] or computationally inexpensive numerical models [135]. Figure 7.3 shows drag coefficients for the NACA64_1-412 foil section (left) and

efficiency curves for an aspect ratio 10 foil with elliptic load based on (7.2) (right). No free-surface effect is taken into account. Curves are shown for two different values of the boundary layer transition parameter $n_c = 4, 9$ as well as for a fully turbulent boundary layer. Both the maximum achievable efficiency and the corresponding optimal lift coefficient C_L strongly depend on the transition location of the boundary layer. In the rest of this work, we consider the case for $n_c = 4$, corresponding to a high level of free-stream turbulence. The corresponding estimated bound for the efficiency is $y_0 = 38$.

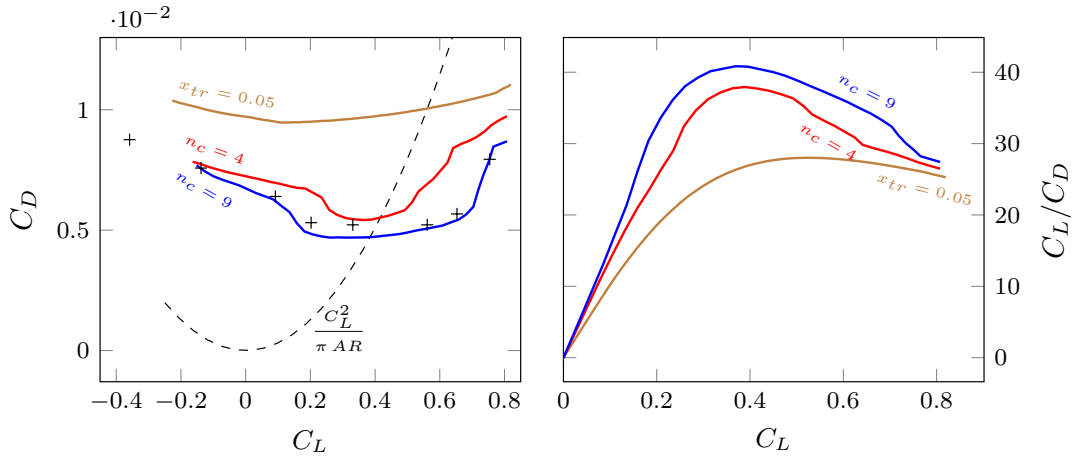


Figure 7.3: NACA64$_1$-412 wing section polar curves. Viscous drag coefficients are computed with XFoil [135]. Experimental drag coefficients from [132] are marked with +.

7.4 Optimization results

The optimization of the L/D ratio for the surface lifting foil described by the parametrization in Figure 7.2 is performed for a design vertical and horizontal lift $S\,C_z = 0.120$ and $S\,C_y = 0.066$, with constraints on the parameters as given in Table 7.2. The vortex lattice method implemented in AVL is used to compute lift and inviscid drag, with

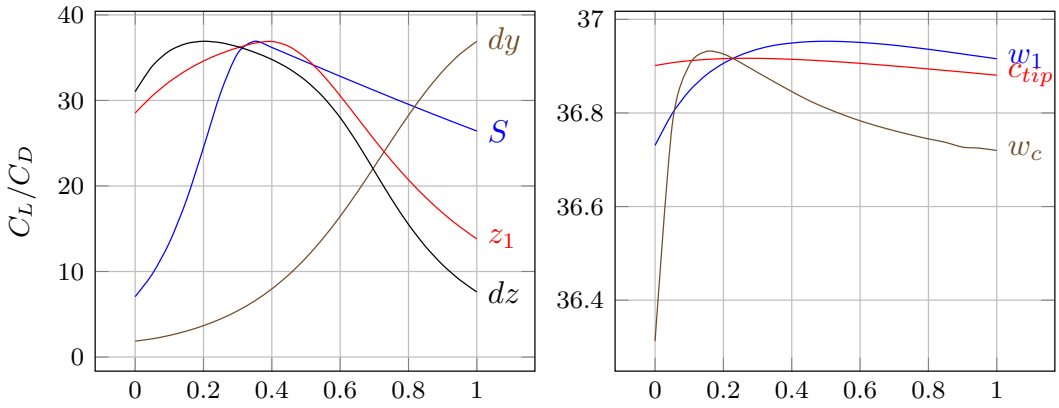


Figure 7.4: Variation of the efficiency as a function of the parameters. All parameters are made dimensionless and varied one by one between zero and one.

Table 7.2: Foil parametrization and bounds

$0.2 \leq S \leq 0.5$
$0.5 \leq z_1 \leq 1.5 \quad 0.5 \leq dy \leq 1.5 \quad -0.3 \leq dz \leq 0.3 \quad 4.3 \leq w_1 \leq 11$
$0.05 \leq c_{tip} \leq 0.5 \quad 1.5 \leq w_c \leq 11$

the free surface modeled as an horizontal constant pressure surface. The viscous drag coefficient is obtained by interpolation from experimental wind-tunnel data [132]. Validation of the model has been provided in §7.2.

The convergence history for the optimization process is shown in Figure 7.5. An efficiency larger than 32 is obtained after only 23 function evaluations, but approximately 160 function evaluations are needed to reach the maximum value 36.81.

Figure 7.6 shows the optimal geometry identified by the optimization algorithm, as well as the ensemble of the geometries tested. The optimal parameters are

It can be remarked that only dy takes its maximum allowed value. At the same

Table 7.3: Optimal parameters for two different bounds for dy

parameter	$dy \leq 1.50$	$dy \leq 2.00$
S	0.305	0.305
z_1	0.89	1.30
dy	1.50	2.00
dz	-0.29	-0.27
w_1	7.25	4.33
c_{tip}	0.21	0.43
w_c	2.58	3.60
$\frac{L}{D}$	36.81	47.60
α	3.78699	3.29558
β	0.02691	-0.07407

time, this does not correspond to the maximum spanwise length of the foil which would be obtained for a 90° angle between the shaft (the vertical part) and the tip (the horizontal part).

7.5 Conclusions

The recently published, $\Delta - DOGS$ global optimization algorithm has been applied to the design of a flying-catamaran hydrofoil, with the goal of maximizing the lift-drag ratio for a specified working condition. The vortex-lattice model implemented in AVL is used to compute the foil's lift and drag characteristics, after being validated with experimental data. While a first-guess, L-shaped, constant chord design with $z_1 = dy = 1.5$ and the same

surface area $S = 0.3050$ has efficiency $L/D \approx 15$, the optimized foil efficiency is above 35. This work shows how computationally inexpensive numerical models can be successfully coupled with efficient, recently developed global optimization algorithms, providing design guidance for the early stages of the design process.

We nonetheless remark that the model implemented in AVL is valid at chord based Froude numbers $Fn_c \approx 10$. Below that, unmodeled wave generation becomes important [133, section 6.8] while above that phenomena like cavitation or ventilation kicks in. As a comparison, the foil of an AC72 boat, having a $0.7m$ chord and sailing at 40 knots ($20m/s$) has $Fn_c = \frac{20}{\sqrt{9.81 \cdot 0.7}} = 7.63$. The use of more computationally intensive, high-fidelity numerical codes for optimization purposes will be investigated in future work.

Acknowledgment

This chapter is submitted to Journal of Ocean Engineering as the following article: G. Meneghello, P. Beyhaghi, T.R. Bewley, “Simulation-based optimization of the hydrofoil of a flying catamaran." The dissertation author was the secondary investigator and author of this paper.

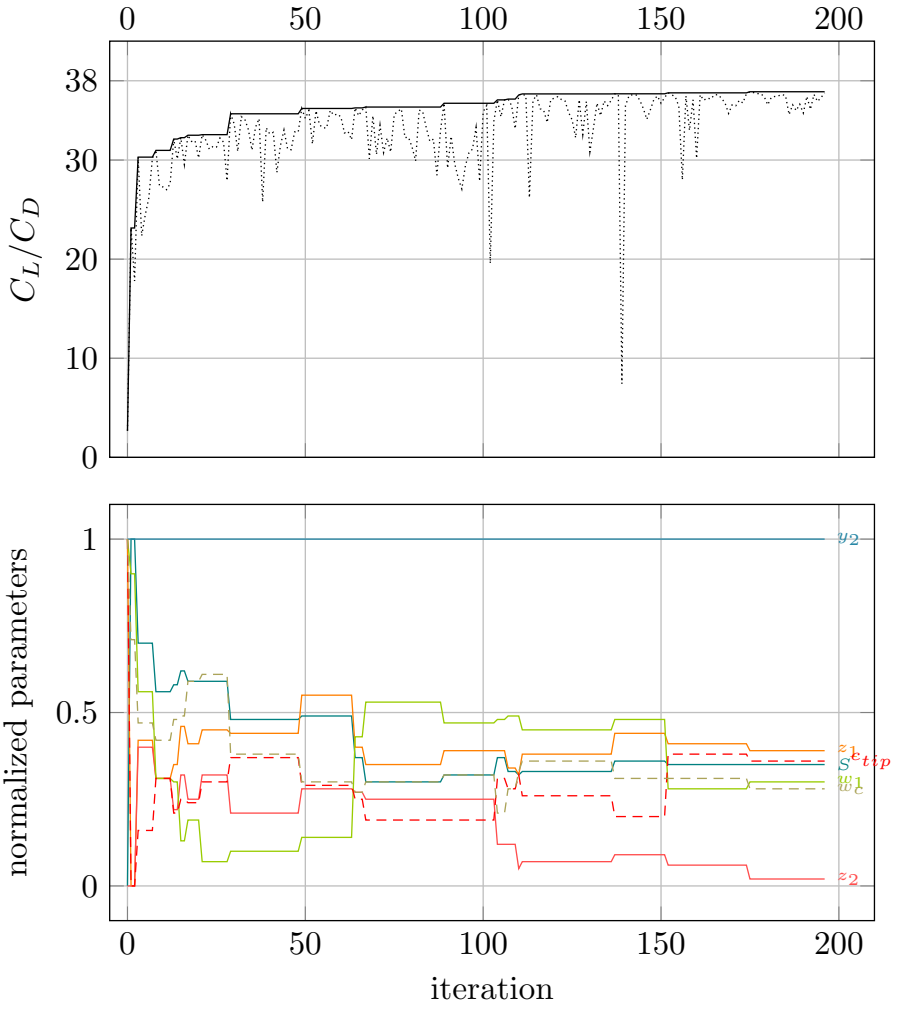


Figure 7.5: Convergence history.

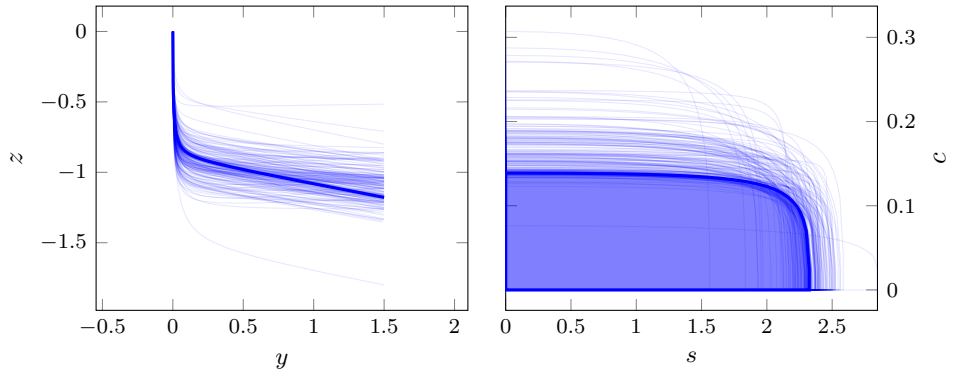


Figure 7.6: Optimized geometry (thick) and all tested geometries

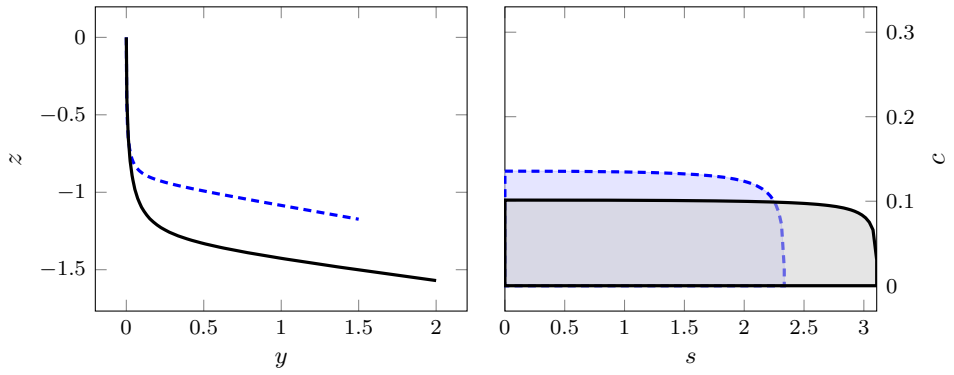


Figure 7.7: Optimized geometries with two different aspect ratios

Chapter 8

Conclusions and future work

In this thesis, we develop a new class of derivative-free optimization algorithms which minimize nonconvex and computationally expensive functions whose function evaluation process are not accurate. The methods presented here can be classified as the response surface methods which use a surrogate function to predict the behavior of the objective function in the regions which are not known. Unlike other response surface methods, our optimization algorithms can employ any interpolation strategy which is smooth. The main novelty in these methods is the introduction of an artificial uncertainty function build on the concept of the Delaunay triangulation. In this thesis, four optimization algorithms are introduced:

The first algorithm called Δ-DOGS is the basic algorithm that minimizes a general objective function within a linearly constrained domain. This algorithm is initialized by performing function evaluations at all vertices of the feasible domain. Afterward, at each step, a metric based on an interpolating strategy and the artificial uncertainty function based on the delay triangulation is minimized to estimated the position of the next function evaluation. We have proved the convergence to the global minimum for general twice

differentiable functions. This optimization algorithm has two main limitations. First, the number of simplifies in the construction of the Delaunay triangulation grows rapidly as the dimension of the problem increases. The second issue is the initialization process which needs function evaluations at all vertices of the feasible domain.

The second algorithm called Δ-DOGS(C) modifies the basic algorithm to deal with nonlinear (but convex) constrained domain. This approach uses less number of function evaluations at the initial step, and will use only $n + 1$ number of initial function evaluations in the n dimensional problem. Moreover, this algorithm performs less exploration in the region which are far from the solution compare to the basic Δ-DOGS.

The third algorithm called Δ-DOGS(Z), addresses an important issue of the basic algorithm which is the nonsmooth behavior of the uncertainty function close to the boundary of the feasibility. This issue exacerbates when the objective function has sharp variation close to the boundary of the search domain. This issue is addresses in the new algorithm by defining a new uncertainty function on the boundary of the feasibility. Moreover, this algorithm implements a Cartesian grid which regularizes the data points during the optimization algorithm which improves the convergence speed of the algorithm.

The fourth algorithm called α-DOGS is designed to solve those objective functions that are obtained by taking the time-averaged statistics of a stationary ergodic random process. The main novelty of this approach was to use different averaging length at different data point in the feasible domain during the optimization process. In this way, the total computational time of the optimization process is significantly reduced. One of the challenging subproblems in α-DOGS is to quantify the uncertainty of averaging process in the stationary ergodic processes. We have also developed a new approach to solving this subproblem which has better performance compare to the existing methods.

Finally, we have validated these optimization algorithms by implementing one of them on an application based optimization problem in ship design.

As future work, more application-based and benchmarking test problems will be studied. Moreover, these methods will be extended to solve those optimization problems which has unknown constraints. We will also study those optimization problems that has an inaccuracy in the input parameter of the function evaluation process. Note that these problems are different from the problems that are addressed in chapter 5, since the inaccuracy would be in the parameter space rather than the function evaluation processes.

Bibliography

[1] R. M. Lewis, V. Torczon, Pattern search methods for linearly constrained minimization, SIAM Journal on Optimization 10 (3) (2000) 917–941.

[2] W. Spendley, G. R. Hext, F. R. Himsworth, Sequential application of simplex designs in optimisation and evolutionary operation, Technometrics 4 (4) (1962) 441–461.

[3] H. Rosenbrock, An automatic method for finding the greatest or least value of a function, The Computer Journal 3 (3) (1960) 175–184.

[4] M. J. Powell, An efficient method for finding the minimum of a function of several variables without calculating derivatives, The computer journal 7 (2) (1964) 155–162.

[5] V. J. Torczon, Multi-directional search: a direct search algorithm for parallel machines, Ph.D. thesis, Citeseer (1989).

[6] V. Torczon, On the convergence of pattern search algorithms, SIAM Journal on optimization 7 (1) (1997) 1–25.

[7] M. Schonlau, W. J. Welch, D. R. Jones, A data-analytic approach to bayesian global optimization, in: Department of Statistics and Actuarial Science and The Institute for Improvement in Quality and Productivity, 1997 ASA conference, 1997.

[8] D. R. Jones, A taxonomy of global optimization methods based on response surfaces, Journal of global optimization 21 (4) (2001) 345–383.

[9] D. G. Krige, A statistical approach to some mine valuation and allied problems on the witwatersrand, Ph.D. thesis (1951).

[10] G. Matheron, Principles of geostatistics, Economic geology 58 (8) (1963) 1246–1266.

[11] C. E. Rasmussen, Gaussian processes for machine learning.

[12] A. J. Booker, J. Dennis Jr, P. D. Frank, D. B. Serafini, V. Torczon, M. W. Trosset, A rigorous framework for optimization of expensive functions by surrogates, Structural optimization 17 (1) (1999) 1–13.

[13] P. Belitz, T. Bewley, New horizons in sphere-packing theory, part ii: lattice-based derivative-free optimization via global surrogates, Journal of Global Optimization 56 (1) (2013) 61–91.

[14] H.-M. Gutmann, A radial basis function method for global optimization, Journal of Global Optimization 19 (3) (2001) 201–227.

[15] A. Alexandrov, Convex polyhedra, gosudarstv, Izdat. Tehn.-Teor. Lit., Moscow-Leningrad.

[16] M. L. Balinski, An algorithm for finding all vertices of convex polyhedral sets, Journal of the Society for Industrial and Applied Mathematics 9 (1) (1961) 72–88.

[17] M. Manas, J. Nedoma, Finding all vertices of a convex polyhedron, Numerische Mathematik 12 (3) (1968) 226–229.

[18] T. Matheiss, D. S. Rubin, A survey and comparison of methods for finding all vertices of convex polyhedral sets, Mathematics of operations research 5 (2) (1980) 167–185.

[19] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2004.

[20] H. Borouchaki, P. L. George, F. Hecht, P. Laug, E. Saltel, Delaunay mesh generation governed by metric specifications. part i. algorithms, Finite elements in analysis and design 25 (1) (1997) 61–83.

[21] R. A. Dwyer, A faster divide-and-conquer algorithm for constructing delaunay triangulations, Algorithmica 2 (1-4) (1987) 137–151.

[22] D. F. Watson, Computing the n-dimensional delaunay tessellation with application to voronoi polytopes, The computer journal 24 (2) (1981) 167–172.

[23] S. Hornus, J.-D. Boissonnat, An efficient implementation of delaunay triangulations in medium dimensions, Ph.D. thesis, INRIA (2008).

[24] J.-D. Boissonnat, O. Devillers, S. Hornus, Incremental construction of the delaunay triangulation and the delaunay graph in medium dimension, in: Proceedings of the twenty-fifth annual symposium on Computational geometry, ACM, 2009, pp. 208–216.

[25] P. McMullen, The maximum numbers of faces of a convex polytope, Mathematika 17 (02) (1970) 179–184.

[26] R. A. Dwyer, Higher-dimensional voronoi diagrams in linear expected time, Discrete & Computational Geometry 6 (3) (1991) 343–367.

[27] R. A. Dwyer, The expected number of k-faces of a voronoi diagram, Computers & Mathematics with Applications 26 (5) (1993) 13–19.

[28] P. E. Gill, W. Murray, Newton-type methods for unconstrained and linearly constrained optimization, Mathematical Programming 7 (1) (1974) 311–350.

[29] J. Nocedal, S. Wright, Numerical optimization, Springer Science & Business Media, 2006.

[30] K. L. Hoffman, A method for globally minimizing concave functions over convex sets, Mathematical Programming 20 (1) (1981) 22–32.

[31] G. Wahba, Spline models for observational data, Vol. 59, Siam, 1990.

[32] S. N. Lophaven, H. B. Nielsen, J. Søndergaard, Dace-a matlab kriging toolbox, version 2.0, Tech. rep. (2002).

[33] X.-Y. Li, Generating well-shaped d-dimensional delaunay meshes, Theoretical Computer Science 296 (1) (2003) 145–165.

[34] X.-Y. Li, S.-H. Teng, Generating well-shaped delaunay meshed in 3d, in: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2001, pp. 28–37.

[35] J. R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation, Computational geometry 22 (1) (2002) 21–74.

[36] X.-S. Yang, Appendix a: test problems in optimization, Engineering optimization (2010) 261–266.

[37] G. H. Hardy, Weierstrass's non-differentiable function, Transactions of the American Mathematical Society 17 (3) (1916) 301–325.

[38] R. Horst, P. M. Pardalos, Handbook of global optimization, Vol. 2, Springer Science & Business Media, 2013.

[39] D. R. Jones, C. D. Perttunen, B. E. Stuckman, Lipschitzian optimization without the lipschitz constant, Journal of Optimization Theory and Applications 79 (1) (1993) 157–181.

[40] R. Paulavicius, J. Zilinskas, Simplicial global optimization, Springer, 2014.

[41] B. O. Shubert, A sequential method seeking the global maximum of a function, SIAM Journal on Numerical Analysis 9 (3) (1972) 379–388.

[42] A. Zhigljavsky, A. Zilinskas, Stochastic global optimization, Vol. 9, Springer Science & Business Media, 2007.

[43] P. E. Gill, W. Murray, M. A. Saunders, M. H. Wright, Inertia-Controlling Methods for General Quadratic Programming, SIAM Review 33 (1) (1991) 1–36.

[44] P. E. Gill, W. Murray, M. A. Saunders, SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization, SIAM Journal on Optimization 12 (4) (2002) 979–1006.

[45] D. P. Bertsekas, On penalty and multiplier methods for constrained minimization, SIAM Journal on Control and Optimization 14 (2) (1976) 216–235.

[46] B. Kort, D. Bertsekas, A new penalty function method for constrained minimization, in: Decision and Control, 1972 and 11th Symposium on Adaptive Processes. Proceedings of the 1972 IEEE Conference on, IEEE, 1972, pp. 162–166.

[47] B. W. Kort, D. P. Bertsekas, Combined primal-dual and penalty methods for convex programming, SIAM Journal on Control and Optimization 14 (2) (1976) 268–294.

[48] V. Torczon, On the convergence of pattern search algorithms, SIAM Journal on optimization 7 (1) (1997) 1–25.

[49] R. M. Lewis, V. Torczon, Pattern search algorithms for bound constrained minimization, SIAM Journal on Optimization 9 (4) (1999) 1082–1099.

[50] R. M. Lewis, V. Torczon, A globally convergent augmented lagrangian pattern search algorithm for optimization with general constraints and simple bounds, SIAM Journal on Optimization 12 (4) (2002) 1075–1089.

[51] M. A. Abramson, C. Audet, J. E. Dennis, S. L. Digabel, OrthoMADS: A Deterministic MADS Instance with Orthogonal Directions (2009).

[52] C. Audet, J. E. Dennis, A Pattern Search Filter Method for Nonlinear Programming without Derivatives, SIAM Journal on Optimization 14 (4) (2004) 980–1010.

[53] C. Audet, J. E. Dennis, Mesh Adaptive Direct Search Algorithms for Constrained Optimization, SIAM Journal on Optimization 17 (1) (2006) 188–217.

[54] C. Audet, J. E. Dennis, A Progressive Barrier for Derivative-Free Nonlinear Programming, SIAM Journal on Optimization 20 (1) (2009) 445–472.

[55] P. Combettes, Hilbertian convex feasibility problem: Convergence of projection methods, Applied Mathematics and Optimization 35 (3) (1997) 311–330.

[56] F. Facchinei, J.-S. Pang, Finite-dimensional variational inequalities and complementarity problems, Springer Science & Business Media, 2007.

[57] O. L. Mangasarian, S. Fromovitz, The fritz john necessary optimality conditions in the presence of equality and inequality constraints, Journal of Mathematical Analysis and Applications 17 (1) (1967) 37–47.

[58] H. S. M. Coxeter, Regular polytopes, Courier Corporation, 1973.

[59] S. Susca, F. Bullo, S. Martínez, Gradient algorithms for polygonal approximation of convex contours, Automatica 45 (2) (2009) 510–516.

[60] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, M. H. Wright, On projected newton barrier methods for linear programming and an equivalence to Karmarkar's projective method, Mathematical Programming 36 (2) (1986) 183–209.

[61] P. B. Zwart, Global maximization of a convex function with linear inequality constraints, Operations Research 22 (3) (1974) 602–609.

[62] D. Bertsimas, J. N. Tsitsiklis, Introduction to Linear Optimization, Vol. 30, 1997.

[63] J. Heinonen, Lectures on Lipschitz analysis, Univ., 2005.

[64] H. L. Smith, P. Waltman, The theory of the chemostat: dynamics of microbial competition, Vol. 13, Cambridge university press, 1995.

[65] S. Le Digabel, Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm, ACM Transactions on Mathematical Software (TOMS) 37 (4) (2011) 44.

[66] A. L. Marsden, M. Wang, J. E. Dennis, Jr., P. Moin, Optimal Aeroacoustic Shape Design Using the Surrogate Management Framework, Optimization and Engineering 5 (2) (2004) 235–262.

[67] P. Beyhaghi, D. Cavaglieri, T. Bewley, Delaunay-based derivative-free optimization via global surrogates, part I: linear constraints, Journal of Global Optimization (2015) 1–52.

[68] P. Beyhaghi, T. Bewley, Delaunay-based derivative-free optimization via global surrogates, part II: Convex constraints, Journal of Global OptimizationUnder review.

[69] R. Kleinberg, A. Slivkins, E. Upfal, Multi-armed bandits in metric spaces, in: Proceedings of the fortieth annual ACM symposium on Theory of computing, ACM, 2008, pp. 681–690.

[70] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: Advances in neural information processing systems, 2012, pp. 2951–2959.

[71] F. Modigliani, F. E. Hohn, Production planning over time and the nature of the expectation and planning horizon, Econometrica, Journal of the Econometric Society (1955) 46–66.

[72] J. R. Hosking, Fractional differencing, Biometrika 68 (1) (1981) 165–176.

[73] S. P. Sethi, G. L. Thompson, What is Optimal Control Theory?, Springer, 2000.

[74] S. P. Sethi, Q. Zhang, H.-Q. Zhang, Average-cost control of stochastic manufacturing systems, Vol. 54, Springer Science & Business Media, 2005.

[75] R. M. Hicks, P. A. Henne, Wing design by numerical optimization, Journal of Aircraft 15 (7) (1978) 407–412.

[76] A. Jameson, S. Yoon, Lower-upper implicit schemes with multiple grids for the euler equations, AIAA journal 25 (7) (1987) 929–935.

[77] J. Reuther, A. Jameson, J. Farmer, L. Martinelli, D. Saunders, Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation, Vol. 96, NASA Ames Research Center, Research Institute for Advanced Computer Science, 1996.

[78] J. J. Reuther, A. Jameson, J. J. Alonso, M. J. Rimllnger, D. Saunders, Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 2, Journal of aircraft 36 (1) (1999) 61–74.

[79] Q. Wang, P. Moin, G. Iaccarino, Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation, SIAM Journal on Scientific Computing 31 (4) (2009) 2549–2567.

[80] Q. Wang, Forward and adjoint sensitivity computation of chaotic dynamical systems, Journal of Computational Physics 235 (2013) 1–13.

[81] V. Picheny, D. Ginsbourger, Y. Richet, G. Caplin, Quantile-based optimization of noisy computer experiments with tunable precision, Technometrics 55 (1) (2013) 2–13.

[82] A. L. Marsden, M. Wang, J. E. Dennis Jr, P. Moin, Suppression of vortex-shedding noise via derivative-free shape optimization, Physics of Fluids 16 (10) (2004) 83–86.

[83] N. Srinivas, A. Krause, S. M. Kakade, M. W. Seeger, Information-theoretic regret bounds for gaussian process optimization in the bandit setting, Information Theory, IEEE Transactions on 58 (5) (2012) 3250–3265.

[84] C. Talnikar, P. Blonigan, J. Bodart, Q. Wang, Parallel optimization for les, in: Proceedings of the Summer Program, 2014, p. 315.

[85] H. P. Awad, P. W. Glynn, On an initial transient deletion rule with rigorous theoretical support, in: Proceedings of the 38th conference on Winter simulation, Winter Simulation Conference, 2006, pp. 186–191.

[86] J. Beran, Statistics for long-memory processes, Vol. 61, CRC Press, 1994.

[87] J. Beran, Maximum likelihood estimation of the differencing parameter for invertible short and long memory autoregressive integrated moving average models, Journal of the Royal Statistical Society. Series B (Methodological) (1995) 659–672.

[88] D. Lenschow, J. Mann, L. Kristensen, How long is long enough when measuring fluxes and other turbulence statistics?, Journal of Atmospheric and Oceanic Technology 11 (3) (1994) 661–673.

[89] T. A. Oliver, N. Malaya, R. Ulerich, R. D. Moser, Estimating uncertainties in statistics computed from direct numerical simulation, Physics of Fluids (1994-present) 26 (3) (2014) 035101.

[90] S. T. Salesky, M. Chamecki, N. L. Dias, Estimating the random error in eddy-covariance based fluxes and other turbulence statistics: the filtering method, Boundary-layer meteorology 144 (1) (2012) 113–135.

[91] J. Duchon, Splines minimizing rotation-invariant semi-norms in sobolev spaces, in: Constructive theory of functions of several variables, Springer, 1977, pp. 85–100.

[92] S. Bubeck, R. Munos, G. Stoltz, C. Szepesvari, X-armed bandits, The Journal of Machine Learning Research 12 (2011) 1655–1695.

[93] P. Beyhaghi, D. Cavaglieri, T. Bewley, Delaunay-based derivative-free optimization via global surrogates, part i: linear constraints, Journal of Global Optimization 1–52.

[94] T. A. Oliver, N. Malaya, R. Ulerich, R. D. Moser, Estimating uncertainties in statistics computed from direct numerical simulation, Physics of Fluids (1994-present) 26 (3) (2014) 035101.

[95] A. L. Marsden, M. Wang, J. E. Dennis Jr, P. Moin, Optimal aeroacoustic shape design using the surrogate management framework, Optimization and Engineering 5 (2) (2004) 235–262.

[96] J. L. Lumley, H. A. Panofsky, The structure of atmospheric turbulence.

[97] J. Wyngaard, O. Coté, The budgets of turbulent kinetic energy and temperature variance in the atmospheric surface layer, Journal of the Atmospheric Sciences 28 (2) (1971) 190–201.

[98] K. Sreenivasan, A. Chambers, R. Antonia, Accuracy of moments of velocity and scalar fluctuations in the atmospheric surface layer, Boundary-Layer Meteorology 14 (3) (1978) 341–359.

[99] S. Alimohammadi, D. He, Multi-stage algorithm for uncertainty analysis of solar power forecasting, in: IEEE Power & Energy Society, General Meeting Conference, 2016.

[100] N. L. Dias, M. Chamecki, A. Kan, C. M. Okawa, A study of spectra, structure and correlation functions and their implications for the stationarity of surface-layer turbulence, Boundary-layer meteorology 110 (2) (2004) 165–189.

[101] A. M. Yaglom, Correlation theory of stationary and related random functions, Springer, 1987.

[102] R. Theunissen, A. Di Sante, M. Riethmuller, R. Van den Braembussche, Confidence estimation using dependent circular block bootstrapping: application to the statistical analysis of piv measurements, Experiments in Fluids 44 (4) (2008) 591–596.

[103] M. Bernardes, N. Dias, The alignment of the mean wind and stress vectors in the unstable surface layer, Boundary-layer meteorology 134 (1) (2010) 41–59.

[104] A. Gluhovsky, E. Agee, A definitive approach to turbulence statistical studies in planetary boundary layers, Journal of the atmospheric sciences 51 (12) (1994) 1682–1690.

[105] D. N. Politis, H. White, Automatic block-length selection for the dependent bootstrap, Econometric Reviews 23 (1) (2004) 53–70.

[106] A. Suarez-González, J. C. López-Ardao, C. Lopez-García, M. Rodríguez-Pérez, M. Fernández-Veiga, M. E. Sousa-Vieira, New simulation output analysis techniques: a batch means procedure for mean value estimation of processes exhibiting long range dependence, in: Proceedings of the 34th conference on Winter simulation: exploring new frontiers, Winter Simulation Conference, 2002, pp. 456–464.

[107] J. Beran, Statistics for long-memory processes, Vol. 61, CRC Press, 1994.

[108] J. R. Hosking, Fractional differencing, Biometrika 68 (1) (1981) 165–176.

[109] M. Priestley, T. S. Rao, A test for non-stationarity of time-series, Journal of the Royal Statistical Society. Series B (Methodological) (1969) 140–149.

[110] G. Nason, A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 75 (5) (2013) 879–904.

[111] E. K. Lada, N. M. Steiger, J. R. Wilson, Performance evaluation of recent procedures for steady-state simulation analysis, IIE Transactions 38 (9) (2006) 711–727.

[112] A. C. Mokashi, J. J. Tejada, S. Yousefi, A. Tafazzoli, T. Xu, J. R. Wilson, N. M. Steiger, Performance comparison of mser-5 and n-skart on the simulation start-up problem, in: Proceedings of the Winter Simulation Conference, Winter Simulation Conference, 2010, pp. 971–982.

[113] S. Robinson, New simulation output analysis techniques: a statistical process control approach for estimating the warm-up period, in: Proceedings of the 34th conference on Winter simulation: exploring new frontiers, Winter Simulation Conference, 2002, pp. 439–446.

[114] A. Tafazzoli, J. R. Wilson, N-skart: A nonsequential skewness-and autoregression-adjusted batch-means procedure for simulation analysis, in: Winter Simulation Conference, Winter Simulation Conference, 2009, pp. 652–662.

[115] K. P. White, An effective truncation heuristic for bias reduction in simulation output, Simulation 69 (6) (1997) 323–334.

[116] K. P. White Jr, M. J. Cobb, S. C. Spratt, A comparison of five steady-state truncation heuristics for simulation, in: Proceedings of the 32nd conference on Winter simulation, Society for Computer Simulation International, 2000, pp. 755–760.

[117] K. Hoad, S. Robinson, R. Davies, Automating warm-up length estimation, in: Proceedings of the 40th Conference on Winter Simulation, Winter Simulation Conference, 2008, pp. 532–540.

[118] P. Beyhaghi, D. Cavaglieri, T. Bewley, Delaunay-based derivative-free optimization via global surrogates, part i: linear constraints, Journal of Global Optimization (2015) 1–52.

[119] P. Beyhaghi, D. Cavaglieri, T. Bewley, Delaunay-based derivative-free optimization via global surrogates, part II: Convex constraints, Journal of Global Optimization (2016) 1–52.

[120] J. Kim, P. Moin, R. Moser, Turbulence statistics in fully developed channel flow at low reynolds number, Journal of fluid mechanics 177 (1987) 133–166.

[121] P. E. Gill, W. Murray, M. A. Saunders, Snopt: An sqp algorithm for large-scale constrained optimization, SIAM review 47 (1) (2005) 99–131.

[122] D. Cavaglieri, T. Bewley, Low-storage implicit/explicit runge–kutta schemes for the simulation of stiff high-dimensional ode systems, Journal of Computational Physics 286 (2015) 172–193.

[123] D. Cavaglieri, P. Beyhaghi, T. Bewley, Low-storage imex runge-kutta schemes for the simulation of navier-stokes systems, in: 21st AIAA computational fluid dynamics conference, San Diego, CA, 2013.

[124] P. Moin, K. Mahesh, Direct numerical simulation: a tool in turbulence research, Annual review of fluid mechanics 30 (1) (1998) 539–578.

[125] R. D. Moser, J. Kim, N. N. Mansour, Direct numerical simulation of turbulent channel flow up to re= 590, Phys. Fluids 11 (4) (1999) 943–945.

[126] P. Luchini, M. Quadrio, A low-cost parallel implementation of direct numerical simulation of wall turbulence, Journal of Computational Physics 211 (2) (2006) 551–571.

[127] W. Y. Kwok, R. D. Moser, J. Jiménez, A critical evaluation of the resolution properties of b-spline and compact finite difference methods, Journal of Computational Physics 174 (2) (2001) 510–551.

[128] J. Kim, Control of turbulent boundary layers, Physics of Fluids (1994-present) 15 (5) (2003) 1093–1105.

[129] K. Graf, A. Hoeve, S. Watin, Comparison of full 3d-rans simulations with 2d-rans/lifting line method calculations for the flow analysis of rigid wings for high performance multihulls, Ocean Engineering 90 (2014) 49–61.

[130] M. Drela, H. Youngren, AVL Athena Vortex Lattice, `http://web.mit.edu/drela/Public/web/avl/`.

[131] J. Katz, A. Plotkin, Low-speed aerodynamics, Vol. 13, Cambridge University Press, 2001.

[132] I. H. Abbott, A. E. Von Doenhoff, Theory of wing sections, including a summary of airfoil data, Courier Corporation, 1959.

[133] O. M. Faltinsen, Hydrodynamics of high-speed marine vehicles, Cambridge university press, 2005.

[134] K. L. Wadlin, C. L. Shuford, J. R. McGehee, A theoretical and experimental investigation of the lift and drag characteristics of hydrofoils at subcritical and supercritical speeds, Tech. rep., Langley Aeronautical Laboratory, Langley Field, Va (1955).

[135] M. Drela, M. B. Giles, Viscous-inviscid analysis of transonic and low reynolds number airfoils, AIAA journal 25 (10) (1987) 1347–1355.