12-2016

Adaptive sampling trust-region methods for derivative-based and derivative-free simulation optimization problems

Sara Shashaani
Purdue University

ADAPTIVE SAMPLING TRUST-REGION METHODS FOR DERIVATIVE-BASED AND DERIVATIVE-FREE SIMULATION OPTIMIZATION PROBLEMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Sara Shashaani

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By SARA SHASHAANI

Entitled

ADAPTIVE SAMPLING TRUST-REGION METHODS FOR DERIVATIVE-BASED NAD DERIVATIVE-FREE SIMULATION OPTIMIZATION PROBLEMS

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

RAGHU PASUPATHY
Chair

SUSAN HUNTER
Co-chair

MOHIT TAWARMALANI

HONG WAN

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): RAGHU PASUPATHY AND SUSAN HUNTER

Approved by: ABHIJIT DESHMUKH 9/28/2016

Head of the Departmental Graduate Program Date

To my father, mother and brother
for being the best family to have.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the support of many individuals. First, I would like to thank my co-advisers Raghu Pasupathy and Susan Hunter for their years of guidance and assistance. Especially Raghu's efforts in guiding me in the world of research were key for my growth and I wholeheartedly appreciate them. I would like to thank my other committee members Rohit Tawarmalani and Hong Wan for their invaluable insight into this work and raising instructive challenges.

I am grateful of the mentorship of Bruce Schmeiser, Roshanak Nateghi, Susan Prieto Welch and Nung Yip. They were ready to give me courage and strength when I was stuck and did not refuse to extend their help whenever possible.

I am thankful for the financial support provided to me as a graduate research and teaching assistant by the Industrial Engineering department at Purdue University and Virginia Tech, as well as the Ross fellowship granted to me by Purdue University.

I would like to thank my friends and companions, Armin Ashoury Rad, Narges Dorratoltaj, Behnaz Ghahestani, Nima Moghimian, Jeff Atkinson, Kalyani Nagaraj, Veronica Quitalo, Fernando Charro, Majid Arabgol, Pushpak Bandari, Oscar Rincon, Manuel Ramirez, Adil Can Dai, Noella Cacciotti, Lorenzo Langella, Zahra Abolhelm, Alireza Javadi, Olivia Smith, Jocelyn Dunn, Sebastian Saeidi, Golyad Akhtari, Lina Khechadurian, and Fazilat Nassiri for their unlimited support and constant encouragement. I thank my meditation guru and instructors for the life-long lessons that they taught me.

Last but not least I want to thank my family, Mohammadbagher Shashaani, Maryam Abolhelm and Ali Shashaani, for giving me the freedom to find my way, sacrificing so much for me to pursue my goals, and loving me unconditionally all along.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

SYMBOLS ix

ABBREVIATIONS xi

ABSTRACT xii

1 INTRODUCTION 1
- 1.1 Motivation 2
- 1.2 Key Complications 3
- 1.3 Related Work 5
  - 1.3.1 SA and SAA 6
  - 1.3.2 Adaptive Sampling 8
- 1.4 Methodological Overview 11
- 1.5 Notation and Convention 12
- 1.6 Organization of the Document 13

2 DETERMINISTIC TRUST-REGION OPTIMIZATION 14
- 2.1 The General DTRO Framework 14
  - 2.1.1 Definitions 15
  - 2.1.2 Algorithm Listing 15
  - 2.1.3 Convergence Results 18
- 2.2 DTRO — Derivarive-Free 21
  - 2.2.1 Definitions 22
  - 2.2.2 Algorithm Listing 27
  - 2.2.3 Convergence Results 30

3 ASTRO: ADAPTIVE SAMPLING TRUST-REGION OPTIMIZATION 36
- 3.1 Problem Statement and Notations 37
- 3.2 Useful Results 38
- 3.3 Related Work 39
- 3.4 Algorithm Listing 40
- 3.5 Convergence Results 43
- 3.6 Implementation and Numerical Experiments 63
  - 3.6.1 Numerical Results 66

Page

4 ASTRO-DF: ADAPTIVE SAMPLING TRUST-REGION OPTIMIZATION — DERIVATIVE-FREE . . . 71
4.1 Preliminaries . . . 71
4.1.1 Related Work . . . 71
4.1.2 Definitions . . . 73
4.2 Algorithm Listing . . . 74
4.3 Convergence Results . . . 78
4.4 Stochastic Interpolation Model Gradient Error Bounds . . . 80
4.5 Main Results . . . 87
5 IMPLEMENTATION HEURISTICS AND NUMERICAL EXPERIENCE WITH ASTRO-DF . . . 96
5.1 Key Implementation Heuristics . . . 96
5.1.1 Choosing Design Points for the Model Construction Step . . . 97
5.1.2 Choosing Algorithm Parameters . . . 100
5.1.3 Pre-processing . . . 102
5.1.4 Solving the TR Subproblem . . . 103
5.1.5 Updating the Next Iterate . . . 103
5.2 Numerical Experience and Discussion . . . 104
6 CONCLUDING REMARKS . . . 114
REFERENCES . . . 118
A PYTHON CODE FOR ASTRO AND ASTRO-DF . . . 123
B QUANTILE PLOTS OF ASTRO AND ASTRO-DF . . . 156
VITA . . . 174

LIST OF TABLES

Table		Page
3.1	Input parameters of the numerical experiment with ASTRO.	66
3.2	Selected problems and their global solutions from the CUTEst problem set with dimensions varying from 2 to 8.	67
3.3	The estimated mean and standard deviation of the true optimality gap at a (random) returned solution of ASTRO, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO on each problem.	69
3.4	The estimated mean and standard deviation of the true gradient norm at a (random) returned solution of ASTRO, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO on each problem.	70
5.1	The estimated mean and standard deviation of the true optimality gap at a (random) returned solution of ASTRO-DF, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO-DF on each problem.	107
5.2	The estimated mean and standard deviation of the true gradient norm at a (random) returned solution of ASTRO-DF, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO-DF on each problem.	108
5.3	The number of iterations, number of points visited, final true function gradient and final optimality gap with 25,000 simulation budget, on different levels of simulation noise.	111

LIST OF FIGURES

Figure		Page
2.1	Deterministic Trust-Region Illustration.	18
2.2	Iteration types in the deterministic DTRO-DF Algorithm. *sqm* stands for sufficient quality model. *stay* refers to $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ and *move* means $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$, while *expand* refers to $\Delta_{k+1} > \Delta_k$ and *contract* refers to $\Delta_{k+1} < \Delta_k$. When neither *expand* nor *contract*, it refers to the trust-region remaining unchanged.	31
3.1	Illustration of the $\hat{\rho}_k$ possibilities. If $\eta_2 = \eta_3$ then the successful and non-contracting iterations overlap.	42
3.2	An example of the set-up used in the proof of Theorem 8.	59
5.1	The one standard deviation interval from the mean of the optimality gap for the Rosenbrock function with $\sigma = 1$ at different levels of simulation budget. Reduction in the first 1500 simulation calls is from $7,398,689$ to 2.37 by average. After 13000 simulation calls the mean stays unchanged at 0.16.	109
5.2	The one standard deviation interval from the mean of the optimality gap (on the left) and true function gradient norm (on the right) for the Rosenbrock function with $\sigma = 1$ at different levels of simulation budget. The variability in the function gradient is evidently more than the variability in the optimality gap.	110
5.3	The $\log(\lvert f(\boldsymbol{X}_{k_{max}}) - f(\boldsymbol{x}^*)\rvert)$ after visiting several points, with the maximum simulation budget of 25,000.	110
B.1	Quantile plots of ASTRO for the functions BEALE, BIGGS6 and BOX3.	157
B.2	Quantile plots of ASTRO for the functions BROWNDEN, CUBE and DENSCHNB.	158
B.3	Quantile plots of ASTRO for the functions DENSCHNC, DENSCHND and DENSCHNE.	159
B.4	Quantile plots of ASTRO for the functions DENSCHNF, ENGVAL2 and HATFLDD.	160
B.5	Quantile plots of ASTRO for the functions HATFLDE, HELIX and HIMMELBF.	161
B.6	Quantile plots of ASTRO for the functions KOWOSB, PALMER5C and PALMER6C.	162

Figure Page

B.7 Quantile plots of ASTRO for the functions PALMER7C, PALMER8C and ROSENBR. 163

B.8 Quantile plots of ASTRO for the functions S308, SINEVAL and YFITU. . . 164

B.9 Quantile plots of ASTRO-DF for the functions BEALE, BIGGS6 and BOX3. 166

B.10 Quantile plots of ASTRO-DF for the functions BROWNDEN, CUBE and DENSCHNB. 167

B.11 Quantile plots of ASTRO-DF for the functions DENSCHNC, DENSCHND and DENSCHNE. 168

B.12 Quantile plots of ASTRO-DF for the functions DENSCHNF, ENGVAL2 and HATFLDD. 169

B.13 Quantile plots of ASTRO-DF for the functions HATFLDE, HELIX and HIMMELBF. 170

B.14 Quantile plots of ASTRO-DF for the functions KOWOSB, PALMER5C and PALMER6C. 171

B.15 Quantile plots of ASTRO-DF for the functions PALMER7C, PALMER8C and ROSENBR. 172

B.16 Quantile plots of ASTRO-DF for the functions S308, SINEVAL and YFITU. 173

SYMBOLS

$f(\cdot)$	deterministic objective function
$F(\cdot)$	stochastic observation of the objective function
$\boldsymbol{g}$	deterministic function gradient
$\boldsymbol{G}$	stochastic observation of the function gradient
$\mathscr{B}$	approximated function Hessian
$m(\cdot)$	deterministic model
$M(\cdot)$	stochastic model
n	deterministic sample size
N	adaptive (stochastic) sample size
d	dimension
$\mathcal{B}(a;r)$	open ball centered around a with radius r
k	iteration number
$\boldsymbol{x}_k$	k-th iterate (incumbent solution)
$\boldsymbol{s}$	search step
$\boldsymbol{p}$	search step
ω	sample path seed
Δ	trust-region radius
ρ	success ratio
η	threshold for the model "fitness"
μ	trust-region and model gradient balance constant
β	model gradient inflation constant
$\ell(\cdot)$	Lagrange polynomial
γ_1	expansion factor of the trust-region radius
γ_2	contraction factor of the trust-region radius

λ	deterministic inflation factor for the sample size
$\mathcal{Y}$	sample set
κ_{ef}	error constant in the function value
κ_{eg}	error constant in the function gradient
κ_{eH}	error constant in the function Hessian
κ_{fcd}	fraction of Cauchy decrease
κ_{bhm}	upper bound on the norm of model Hessian
κ_{f}	function adaptive sampling constant, used in ASTRO
κ_{g}	gradient adaptive sampling constant, used in ASTRO
κ_{oas}	outer adaptive sampling constant, used in ASTRO-DF
κ_{ias}	inner adaptive sampling constant, used in ASTRO-DF
ν_{gL}	Lipschitz constant of the function gradient
ν_{HL}	Lipschitz constant of the function Hessian
ν_{gL}^{m}	Lipschitz constant of the model gradient
ν_{HL}^{m}	Lipschitz constant of the model Hessian
i.o.	infinitely often

ABBREVIATIONS

ASTRO	Adaptive Sampling Trust-Region Optimization
ASTRO-DF	Adaptive Sampling Trust-Region Optimization — Derivative-Free
DFO	Derivative-Free Optimization
FL	Fully-Linear
FQ	Fully-Quadratic
SAA	Sample Average Approximation
SA	Stochastic Approximation
SO	Simulation Optimization
STORM	Stochastic Optimization with Random Models
STRONG	Stochastic Trust-Region Response Surface Method
TRO-DF	Trust-Region Optimization — Derivative-Free

ABSTRACT

Shashaani, Sara PhD, Purdue University, December 2016. Adaptive Sampling Trust-Region Methods for Derivative-Based and Derivative-Free Simulation Optimization Problems. Major Professors: Raghu Pasupathy, Susan Hunter.

We consider unconstrained optimization problems where only "stochastic" estimates of the objective function are observable as replicates from a Monte Carlo simulation oracle. In the first study we assume that the function gradients are directly observable through the Monte Carlo simulation. We propose ASTRO, which is an adaptive sampling based trust-region optimization method where a stochastic local model is constructed, optimized, and updated iteratively. ASTRO is a derivative-based algorithm and provides almost sure convergence to a first-order critical point with good practical performance. In the second study the Monte Carlo simulation is assumed to provide no direct observations of the function gradient. We present ASTRO-DF, which is a class of derivative-free trust-region algorithms, where the stochastic local model is obtained through interpolation. Function estimation (as well as gradient estimation) and model construction within ASTRO and ASTRO-DF are *adaptive* in the sense that the extent of Monte Carlo sampling is determined by continuously monitoring and balancing metrics of sampling and structural errors within ASTRO and ASTRO-DF. Such error balancing is designed to ensure that the Monte Carlo effort within ASTRO and ASTRO-DF is sensitive to algorithm trajectory, sampling more whenever an iterate is inferred to be close to a critical point and less when far away. We demonstrate the almost-sure convergence of ASTRO-DF's iterates to a first-order critical point when using quadratic stochastic interpolation models. The question of using more complicated models, e.g., regression or stochastic kriging, in combination with adaptive sampling is worth further investigation and will benefit from the methods of proof we present. We investigate the implementation of ASTRO and ASTRO-DF along with the

heuristics that enhance the implementation of ASTRO-DF, and report their finite-time performance on a series of low-to-moderate dimensional problems in the CUTEr framework. We speculate that the iterates of both ASTRO and ASTRO-DF achieve the canonical Monte Carlo convergence rate, although a proof remains elusive.

1. INTRODUCTION

We consider unconstrained continuous simulation optimization (SO) problems, that is, optimization problems in continuous space where the objective function can only be expressed implicitly via a Monte Carlo oracle. We formally state the problem as follows.

$$\begin{aligned} \text{Problem } P: \\ \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathbb{X}, \end{aligned}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a differentiable function that is bounded from below. For each $\boldsymbol{x} \in \mathbb{X} \subseteq \mathbb{R}^d$, the function $f(\boldsymbol{x})$ is estimated using the consistent estimator $f(\boldsymbol{x}, n)$ satisfying $f(\boldsymbol{x}, n) \xrightarrow{wp1} f(\boldsymbol{x})$ as $n \to \infty$, where n is a measure of Monte Carlo effort.

Often the estimator $f(\boldsymbol{x}, n)$ is a simple sample mean, in which case we use the notation $\bar{F}(\boldsymbol{x}, n) = n^{-1} \sum_{j=1}^{n} F_j(\boldsymbol{x})$, where $F_j(\boldsymbol{x}), j = 1, 2, \ldots, n$ are n independent and identically distributed (iid) replicates obtained by "executing" a Monte Carlo simulation at the point $\boldsymbol{x}$. The estimated standard error for the function estimator $\bar{F}$ is $\hat{\sigma}_F(\boldsymbol{x}, n) / \sqrt{n}$ where

$$\hat{\sigma}_F^2(\boldsymbol{x}, n) = n^{-1} \sum_{j=1}^{n} \left(F_j(\boldsymbol{x}) - \bar{F}(\boldsymbol{x}, n) \right)^2.$$

An algorithm for solving Problem P will be evaluated based on its ability to return a (random) sequence of iterates $\{\boldsymbol{X}_k\}$ converging in some rigorously defined probabilistic metric to a solution of Problem P. SO algorithms that return iterates $\{\boldsymbol{X}_k\}$ guaranteed to converge with probability one to a critical point will be called *consistent*. Furthermore, unlike deterministic contexts where the convergence rate of a converging sequence of iterates is measured as a function of the number of iterations, the convergence rate of the random sequence of iterates $\{\boldsymbol{X}_k\}$ will be measured with respect to total simulation effort W_k extended after k iterations. In this regard, the phrase "canonical rate" is used throughout this document to refer to the fastest achievable convergence rate (for the iterates of an algorithm)

under generic Monte Carlo sampling. Specifically, we will say that the random sequence of iterates $\{\boldsymbol{X}_k\}$ achieves the canonical Monte Carlo rate if $\sqrt{W_k}\,\|\nabla f(\boldsymbol{X}_k)\| = \mathcal{O}_p(1)$, where W_k is the total simulation effort at the end of the k-th iteration. (See Section 1.5 for a formal definition of the notion of probabilistic complexity $\mathcal{O}_p(\cdot)$ and related concepts.)

1.1 Motivation

SO has recently gathered attention due to its versatile formulation, allowing the user to specify functions involved in an optimization problem implicitly, e.g., through a stochastic simulation. As a result, virtually any level of problem complexity can be embedded within the problem, leading to wide adoption.

A specific example from epidemic modeling serves to illustrate the versatility of SO particularly well. The question of how best to control an epidemic, e.g., influenza, has recently gained attention as a crucial part of the national response to health crises. In this regard, *contact network modeling* (CNM) has shown particular promise [1] as an analytic technique that is effective in modeling disease propagation within large populations. In CNM, each person (or each group of people) in a city is modeled as a node in a graph; and contacts among nodes are modeled as edges, often giving rise to graphs having several million nodes and edges [2,3]. The edges emanating from a node constitute channels along which parasitic transmission might occur, and their structure dictates the manner in which epidemic spreading is modeled. Given such a setup, and understanding that government policy can explicitly affect contact network structure, important modeling and optimization questions can be posed. For example, what disease parameters, e.g., transmissibility, incubation period, infectious period, best represent an ongoing epidemic in a population? How should a limited stockpile of vaccines aimed at thwarting a spreading virus be distributed across nodes and across time so as to minimize the expected peak disease exposure within the graph? The decision variables implicit in the former question are the disease parameters; the decision variables implicit in the latter question are the vaccine allocation proportions across the nodes in the contact network. For both questions, the simulation out-

put may be a function of the epidemic curve representing the number of infected individuals per day within the given population.

It is easy to find further examples of SO formulations in widespread applications such as telecommunication networks [4], traffic control [5] and health care [6]. So much so that recent editions of the Winter Simulation Conference (`www.informs-sim.org`) have dedicated an entire track to the SO problem and its various flavors. For a library of SO problems, see `www.simopt.org` and [7, 8].

1.2 Key Complications

A crucial element characterizing SO problems is the presence of a Monte Carlo oracle. The incorporation of a Monte Carlo oracle lends flexibility to the SO problem formulation, but it also brings with it a simply-stated complication having far-reaching effects on solution development: Monte Carlo oracles within SO, unlike their deterministic counterparts, provide only stochastic observations of the involved functions, and in some cases gradients. Thus, estimates of the objective function (and gradient) values can provide only probabilistic precision guarantees that depend on the amount of Monte Carlo effort expended. Specifically, suppose $f(\boldsymbol{x},n)$ is the Monte Carlo estimate of the unknown but desired function value $f(\boldsymbol{x})$ at the point $\boldsymbol{x}$, and n represents the Monte Carlo effort (or the sample size). Then, simple probability arguments reveal that deterministic guarantees of the sort $\|f(\boldsymbol{x},n)-f(\boldsymbol{x})\| \leq \varepsilon, \varepsilon > 0$ do not hold irrespective of the size of n; instead, in SO problems, we have to be satisfied with probabilistic precision guarantees of the form $\mathbb{P}\{\|f(\boldsymbol{x},n)-f(\boldsymbol{x})\| > \varepsilon\} \leq \alpha$ for $n \geq N(\alpha)$, or $\lim_{n\to\infty}\mathbb{P}\{\|f(\boldsymbol{x},n)-f(\boldsymbol{x})\| > \varepsilon\} = 0$.

The lack of deterministic guarantees on function estimates in the SO context presents some unique challenges when constructing and analyzing numerical algorithms. The first complication arises from the standpoint of analyzing SO algorithms. Deterministic nonlinear programming algorithms are constructed so that during each defined iteration, at most a few function oracle calls are expended. Such constancy of oracle effort across iterations, along with the fact that function calls are usually computationally inexpensive, justifies

analyzing algorithm behavior as a function of the number of iterations. By contrast, in the SO context, Monte Carlo oracle calls can be computationally burdensome compared with basic mathematical operations on a computer; and, depending on the nature of the SO algorithm, different number of Monte Carlo calls may be expended across iterations, e.g., constant as in Stochastic Approximation (SA) [9], varying but predetermined as in Retrospective Approximation (RA) [10], or random as in Sampling Controlled Stochastic Recursion (SCSR) [11]. This means that the elemental measure of effort in SO — the number of Monte Carlo oracle calls — may not have a simple relationship with the notion of "iterations" defined within the specific SO algorithm, forcing a need for more careful book-keeping. This is why iterative SO algorithms measure convergence and convergence rates not in terms of the number of iterations as deterministic iterative structures do, but rather in terms of the total number of Monte Carlo calls. A more subtle issue is that the stochastic element within SO imposes a certain upper bound on the maximum achievable convergence rate (also known as *canonical rate* [12]) of SO algorithms. As we will see in greater detail, the correct way to measure (asymptotic) SO algorithm performance is with respect to their ability to achieve the canonical convergence rate.

The second complication is specific to the case where no derivative information is available directly from the Monte Carlo oracle. In such cases search mechanisms such as derivative approximations that are routine in the deterministic nonlinear programming context often become unreliable. If the derivative estimate $\hat{\nabla} f(\boldsymbol{x}) := (\hat{\nabla}_1 f(\boldsymbol{x}), \hat{\nabla}_2 f(\boldsymbol{x}), \ldots, \hat{\nabla}_q f(\boldsymbol{x}))$ is constructed using a central-difference approximation as

$$\hat{\nabla}_i f(\boldsymbol{x}) = 2c_n^{-1}(f(\boldsymbol{x} + c_n \boldsymbol{e}_i, n) - f(\boldsymbol{x} - c_n \boldsymbol{e}_i, n)), i = 1, 2, \ldots, q,$$

then, as in the function estimation context, no uniform guarantees on the accuracy of $\hat{\nabla} f(\boldsymbol{x})$ are available in general. Furthermore, the rate at which $\hat{\nabla} f(\boldsymbol{x})$ converges to $\nabla f(\boldsymbol{x})$ depends crucially on the delicate choice of the step-size for finite-differencing $\{c_n\}$ in the presence of stochastic error, with the best possible rate $\mathcal{O}(n^{-1/3})$ under generic Monte Carlo sampling being much slower than the corresponding $\mathcal{O}(n^{-1/2})$ rate for function estimation (See [13] for this and related results.) Most importantly, implementing such finite-difference derivative estimates within an SO algorithm is well recognized to be a delicate

issue, easily causing instabilities. Moreover, the lack of uniform deterministic guarantees in the SO context can results in accumulation of estimation error across iterations and threaten algorithm convergence. As we will detail further, algorithms for SO usually contend with this issue by either introducing a carefully tuned sequence of parameters, e.g., stochastic approximation (SA) [9], or perform adequate Monte Carlo replications [11, 14] at each visited point.

Remark 1. *Throughout this document, we use the term "sampling" to refer to the act of obtaining replicates using multiple runs of the Monte Carlo oracle at a fixed point. This is not to be confused with sampling design points in the search region. So, when we say that the sample size is n, we mean that n amount of Monte Carlo effort was expended to obtain the function estimate* at a fixed point.

Another complication within SO, but one that it shares with black-box deterministic optimization contexts, is the lack of information about function structure. Properties such as convexity, uni-modality, and differentiability, if known to be present, can be exploited when designing algorithms. Such properties are usually assumed (and not inferred) within the deterministic context, and an appropriate solution algorithm devised. In SO, however, structural assumptions about the underlying true objective and constraint function, even if correct, may not provide as much leverage during algorithm development. This is because, due to the presence of stochastic error, the true objective and constraint functions are never directly observed; and, making structural assumptions about their (observed) sample-paths is far more suspect.

1.3 Related Work

Much progress has been made in recent years on solving various flavors of the SO problem. The predominant solution methods in the literature fall into two broad categories called Stochastic Approximation (SA) and Sample-Average Approximation (SAA). SA and SAA date back approximately six decades and two decades respectively; accordingly, the literature on SA and SAA algorithmic variations and their properties is enormous. More

recently, newer classes of algorithms that can be described as "stochastic versions" of iterative structures in the deterministic context have emerged.

In what follows, we provide a very brief description of SA and SAA and provide entry points into the literature. We also discuss some of the more recent work that is directly related to what we propose here, called adaptive sampling.

1.3.1 SA and SAA

The rudimentary Stochastic Approximation (SA) type methods have the following generic form.

$$\boldsymbol{X}_{k+1} = \Pi_{\mathcal{D}}\left(\boldsymbol{X}_k - a_k \boldsymbol{G}_k\right), \tag{1.1}$$

where $\Pi_{\mathcal{D}}(\boldsymbol{x})$ is the projection of the point $\boldsymbol{x}$ onto the set $\mathcal{D}$, $\{a_k\}$ is a user-chosen positive-valued scalar sequence, and $\boldsymbol{G}_k$ is an estimator of the gradient $\nabla f(\boldsymbol{X}_k)$ of the function f at the point $\boldsymbol{X}_k$. When direct Monte Carlo observations of the objective function f are available, the most common expression for $\boldsymbol{G}_k = \left(G_k^1, G_k^2, \cdots, G_k^d\right)$ is either the central-difference approximation $G_k^i = (2c_k)^{-1}(F(\boldsymbol{X}_k + c_k\boldsymbol{e}_i) - F(\boldsymbol{X}_k - c_k\boldsymbol{e}_i))$ or the forward-difference approximation $G_k^i = c_k^{-1}(F(\boldsymbol{X}_k + c_k\boldsymbol{e}_i) - F(\boldsymbol{X}_k))$ of the gradient $\nabla f(\boldsymbol{X}_k)$, where $\{c_k\}$ is a positive-valued sequence and $F : \mathbb{R}^d \to \mathbb{R}$ is the observable estimator of the objective function $f : \mathbb{R}^d \to \mathbb{R}$. The resulting recursion is the famous Kiefer-Wolfowitz process [15, 16]. More recent recursions include an estimated Hessian $\mathscr{H}_k(\cdot)$ of the function f at the point $\boldsymbol{X}_k$:

$$\boldsymbol{X}_{k+1} = \Pi_{\mathcal{D}}\left(\boldsymbol{X}_k - a_k \mathscr{H}_k^{-1} \boldsymbol{G}_k\right), \tag{1.2}$$

making the resulting recursion in (1.1) look closer to the classical Newton's iteration in the deterministic context. The Hessian estimator $\mathscr{H}_k(\cdot)$ has d^2 entries, and hence, most methods that use (1.2) estimate $\mathscr{H}_k(\cdot)$ either using a parsimonious design (e.g., [17, 18]), or construct it from the history of observed points.

As can be seen in (1.1), the SA recursion is very simply stated and implemented, and little has changed in its basic structure since 1951, when it was first introduced by Robbins

and Monro [19] for the context of finding a zero of a "noisy" vector function. Instead, much of the research over the ensuing decades has focused on questions such as convergence and convergence rates of SA type algorithms, the effect of averaging on the consistency and convergence rates of the iterates, and efforts to choose the sequence $\{a_k\}$ in an adaptive fashion. Some good entry points into various aspects of the SA literature include [20–24].

An alternative to SA for solving local continuous SO problems is Sample Average Approximation (SAA), an algorithmic framework independently introduced by Rubinstein and Shapiro [25] and by Healy and Schruben [26]. (We call SAA a *framework* because it is not an algorithm per se, and all steps to solving a problem are not well defined.) SAA is easily stated: (i) explicitly or implicitly "generate" an approximation to the objective function $f(\boldsymbol{x})$ through sampling, thereby obtaining what is called the sample-path problem; and (ii) using any mathematical programming technique, "solve" the generated approximate problem as a deterministic optimization problem to within a desired tolerance. More rigorously, the SAA method substitutes the original problem

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) = \mathbb{E}[F(\boldsymbol{x}, \boldsymbol{\xi})] \\ & h(\boldsymbol{x}) \leq 0, \\ & \boldsymbol{x} \in \mathcal{D}; \end{aligned} \tag{1.3}$$

with the sample-path problem

$$\begin{aligned} \text{minimize} \quad & F_m(\boldsymbol{x}) = m^{-1} \left(\sum_{i=1}^{m} F(\boldsymbol{x}, \boldsymbol{\xi}_i) \right) \\ & h(\boldsymbol{x}) \leq 0, \\ & \boldsymbol{x} \in \mathcal{D}. \end{aligned} \tag{1.4}$$

The sample-path problem in (1.4) is realized by constructing an estimator of the objective function f using a user-specified sample size m, and solved using a user-specified algorithmic procedure. (See [14] and references therein for examples of SAA application.)

SAA has been the subject of a tremendous amount of theoretical and empirical research over the last two decades. For example, the conditions that allow the transfer of structural

properties from the sample-path $F(\boldsymbol{x}, \xi)$ to the limit function $f(\boldsymbol{x})$ [14, Propositions 1,3,4]; the sufficient conditions for the consistency of the optimal value and solution of (1.4) assuming the numerical procedure in use within SAA can produce global optima [27, Theorem 5.3]; consistency of the set of stationary points of (1.4) [27, 28]; convergence rates for the optimal value [27, Theorem 5.7] and optimal solution [14, Theorem 12]; expressions for the minimum sample size m that provides probabilistic guarantees on the optimality gap of the sample-path solution [29, Theorem 5.18]; methods for estimating the accuracy of an obtained solution [30–32]; and quantifications of the trade-off between searching and sampling [33], have all been thoroughly studied.

There appears to be general consensus that SAA as a method can be useful but only when there is known or discernible structure in the sample-path problems. When such structure is present, the power of deterministic nonlinear programming techniques can be brought to bear, leading to efficiencies. (See [14] for more on when the SAA method might be appropriate.) There also appears to be general consensus that SAA methods can pose implementation difficulties arising from the choice of sample size m in (1.4). Specifically, it has been observed [34] that results on the minimum sample size m needed to guarantee a solution of specified quality tends to be so high as to render the resulting procedure not viable.

1.3.2 Adaptive Sampling

While SA and SAA solution methods are reasonable, they face complications when implemented. Specifically, since all observations of the function and gradient, if available, are based on Monte Carlo oracle, the question of how much sampling to perform (how many times to run the oracle at a given design point) arises. Too little Monte Carlo effort threatens convergence due to accumulated stochastic and deterministic errors; and too much Monte Carlo sampling means reduced overall efficiency. Adaptive sampling is a way to ensure that "just adequate" Monte Carlo sampling is performed. A simple adaptive sampling rule that balances sampling error with structural error, for use whenever function estimates are

needed within the algorithm is desired. For example Monte Carlo sampling is adaptive in the sense that it continues until a certain continuously monitored metric of structural error exceeds a metric of sampling variability. We believe that such adaptive sampling paves the way for efficiency because it reacts to the observed algorithm trajectory and, as we shall see, keeps the different sources of error within the algorithm in lock-step. The resulting algorithm remains practical because of the simplicity of the sampling rule. Adaptive sampling as an idea is not new and has been used with great success in other areas such as sequential confidence interval construction [35, 36] and SO on finite spaces [37].

Despite its intuitive concept and simplistic implementation, adaptive sampling introduces substantial complications when analyzing algorithm behavior. Akin to what happens during sequential sampling in the context of confidence interval construction [35, 38, 39], the explicit dependence of the extent of Monte Carlo sampling on algorithm trajectory causes systematic early stopping and consequent bias in the function and/or gradient estimates obtained within the algorithm. In other words, when using adaptive sampling, $\mathbb{E}[f(\boldsymbol{x},n)] \neq f(\boldsymbol{x})$ in general since the sample size n is a stopping time that will depend on $f(\boldsymbol{x},n)$. Demonstrating with probability one convergence of an adaptive sampling based algorithm then entails demonstrating that the bias effects of adaptive sampling, wear away asymptotically. The latter is specifically non-trivial when used within the derivative-free context. In the proposed algorithms in this dissertation we accomplish this by first (generically) characterizing a relationship between the moments of the adaptive sample size and the function and gradient estimates at stopping, and then showing that the errors induced in model construction, function estimation, and algorithm recursion remain in lock-step throughout the algorithm's evolution.

One of the preliminary studies on stopping rules is the fixed-width confidence interval estimation of the mean in [35] with pre-assigned confidence coefficient. We include this result in Theorem 1 for review. A nonparametric approach in [38] establishes a relationship between the moments of the sample size and stopping threshold, and as a result approximates the $\mathbb{E}[\bar{X}_N^2]$. Relying on the latter's arguments reviewed in Theorem 2 rooted on a weak sample-size lower bound, we can demonstrate that the proposed algorithm is indeed

consistent, i.e. its iterates converge to a first-order (or second-order) critical point of the objective function.

Theorem 1 (Chow and Robbins, 1965)**.** *Suppose random variables $X_i, i = 1,2,\ldots$ are iid with variance $\sigma < \infty$, $\bar{X}_n = n^{-1}\sum_{i=1}^n X_i$, $\hat{\sigma}_n^2 = n^{-1}\sum_{i=1}^n (X_i - \bar{X}_n)^2$, and $\{a_n\}$ a sequence of positive constants such that $a_n \to a$ as $n \to \infty$. If*

$$N = \inf\left\{ n \geq 1 : \frac{\hat{\sigma}}{\sqrt{n}} \leq \frac{d}{a_n} \right\},$$

then $d^2 N / \left(a^2\sigma^2\right) \xrightarrow{wp1} 1$ and $\hat{\sigma}_N/\sigma \xrightarrow{wp1} 1$ as $d \to 0$.

Theorem 2. *Suppose random variables $X_i, i = 1,2,\ldots$ are iid with $\mathbb{E}[X_1] = 0, \mathbb{E}[X_1^2] = \sigma^2 > 0$, and $\mathbb{E}[|X_1|^{4v}] < \infty$ for some $v \geq 2$. Let $\hat{\sigma}_n^2 = n^{-1}\sum_{i=1}^n (X_i - \bar{X}_n)^2$, where $\bar{X}_n = n^{-1}\sum_{i=1}^n X_i$. If*

$$N = \inf\left\{ n \geq \lambda^\gamma : \frac{\hat{\sigma}_n}{\sqrt{n}} \leq \frac{\kappa}{\sqrt{\lambda}} \right\}, \gamma \in (0,1]$$

then the following hold.

(i) As $\lambda \to \infty$,

$$\mathbb{P}\{N < \infty\} = 1 \text{ and } N \xrightarrow{wp1} \infty.$$

(ii) As $\lambda \to \infty$,

$$\frac{N\kappa^2}{\sigma^2\lambda} \xrightarrow{wp1} 1.$$

(iii) As $\lambda \to \infty$ and for every $s < v$,

$$\mathbb{E}[N^s] \sim \sigma^{2s}\kappa^{-2s}\lambda^s.$$

(iv) For every $\varepsilon \in (0,1)$,

$$\mathbb{P}\{N \leq \sigma^2\kappa^{-2}\lambda(1-\varepsilon)\} = \theta\lambda^{-(v-1)\gamma},$$

where θ is a constant that might depend only on v and moments of X_1.

(v) As $\lambda \to \infty$,

$$\mathbb{E}[\bar{X}_N^2] \sim \kappa^2\lambda^{-1}.$$

1.4 Methodological Overview

Our particular focus in this research is that of developing a class of algorithms for solving low to moderate dimensional SO problems that have no readily discernible structure, when direct observations of both the sample-path function and derivatives are available, e.g., through infinitesimal perturbation analysis [12, 40, 41], and when direct derivative observations are unavailable. The model-based methods we propose eschew the direct computation and use of derivatives for searching, and instead rely on constructed models of guaranteed quality to find the search step for the next iteration.

We are inspired by analogous problems in the deterministic context that have spurred the development of a special and arguably very useful class of globally converging optimization methods called trust-region methods. The classic deterministic trust-region (DTRO) algorithms [42, 43] evolve by repeatedly constructing and optimizing a local and analytically convenient model based on the Taylor-series expansion of the objective function around the latest iterate within a dynamic trust-region, implicitly restricting the distance between the successive iterates returned by the algorithm. In the deterministic model-based trust-region derivative-free (DTRO-DF) algorithms [44–47] models constructed during past iterations can be re-used with some updating, and no effort is expended for explicit estimation of derivatives. (In Chapter 2 we review both DTRO and DTRO-DF along with their convergence analyses.)

We propose stochastic versions of DTRO and DTRO-DF called ASTRO and ASTRO-DF, where the basic deterministic trust-region machinery is combined with a crucial sampling idea called *adaptive sampling*. ASTRO and ASTRO-DF work as follows. During each iteration k, construct a local (and analytically convenient) stochastic model of the objective function within a trust-region around the current iterate. Optimize the constructed model to identify a candidate solution that is accepted if passing a certain threshold of quality. If the candidate solution is accepted, the trust-region is expanded by a factor as a signal of confidence in the constructed model. Otherwise, that is, if the candidate solution does not pass the test of quality, the incumbent solution is retained and the trust-region is shrunk

by a factor. Efficiency within ASTRO and ASTRO-DF is facilitated by adaptive sampling which strives to keep the algorithmic, model, and sampling errors in lock-step.

ASTRO and ASTRO-DF while easily explained, are difficult to analze primarily due to the introduction of adaptive sampling ideas. Our central thesis, however, is that adaptive sampling is crucial for efficiency of algorithms in the SO context; through the combined and careful use of derivative-based or derivative-free trust-region machinery with adaptive sampling, low to moderate dimensional continuous and unconstrained SO problems can be solved to local optimality starting from any initial solution and with probability one, while ensuring canonical Monte Carlo convergence rates.

1.5 Notation and Convention

We will adopt the following notation throughout the dissertation.

1. We use bold font for vectors, script font for sets, and calligraphic fonts for matrices; lower case letters for real numbers and upper case letters for random variables. Components of a vector are denoted by the same regular font but with superscripts that denote the label of the component. Hence:
   - If $\boldsymbol{x} \in \mathbb{R}^d$ is a vector, then its components are denoted through $\boldsymbol{x} = \left(x^1, x^2, \ldots, x^d\right)$.
   - $\mathscr{X} = \left[\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \cdots, \boldsymbol{x}_p^T\right]$ is a matrix of p vectors $\boldsymbol{x}_i$.
   - $\{\boldsymbol{X}_k\}$ denotes a sequence of random vectors in $\mathbb{R}^d$.
   - $\mathcal{X} := \left\{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_p\right\}$ denotes a set of p random vectors.
2. For a sequence of random vectors $\{\boldsymbol{X}_k\}$, we say $\boldsymbol{X}_k \xrightarrow{D} \boldsymbol{X}$ if $\{\boldsymbol{X}_k\}$ converges to $\boldsymbol{X}$ weakly or in distribution, $\boldsymbol{X}_k \xrightarrow{p} \boldsymbol{X}$ if $\{\boldsymbol{X}_k\}$ converges to $\boldsymbol{X}$ in probability, and $\boldsymbol{X}_k \xrightarrow{wp1} \boldsymbol{X}$ if $\{\boldsymbol{X}_k\}$ converges to $\boldsymbol{X}$ with probability 1 or almost surely.
3. $\mathcal{B}(\boldsymbol{x}; r) = \left\{\boldsymbol{y} \in \mathbb{R}^d : \|\boldsymbol{y} - \boldsymbol{x}\| \leq r\right\}$ denotes a closed ball of radius $r > 0$ with center $\boldsymbol{x}$. Throughout the document by the norm $\|.\|$ we mean the standard Euclidean norm.

4. For a sequence of real numbers $\{a_k\}$, we say $a_k = \mathcal{O}(1)$ if $\lim_{k\to\infty} a_k = 0$, and $a_k = \mathcal{O}(1)$ if $\{a_k\}$ is bounded, that is, there is a constant $M > 0$ with $|a_k| < M$ for large enough k.

5. For a sequence of random vectors $\{\boldsymbol{X}_k\}$, we say $\boldsymbol{X}_k = o_p(1)$ if $\boldsymbol{X}_k \xrightarrow{p} 0$, and $\boldsymbol{X}_k = \mathcal{O}_p(1)$ if $\{\boldsymbol{X}_k\}$ is stochastically bounded, that is, for given $\varepsilon > 0$ there exists $\delta(\varepsilon) \in (0,\infty)$ with $\mathbb{P}\{|\boldsymbol{X}_k| < \delta(\varepsilon)\} > 1 - \varepsilon$.

6. For sequences of real numbers $\{a_k\}$, $\{b_k\}$, we say that $a_k \sim b_k$ if $\lim_{k\to\infty} a_k/b_k = 1$.

1.6 Organization of the Document

The dissertation is organized into chapters. The next chapter provides background on detrministic trust region optimization methods in the derivate-based and derivative-free contexts. Chapters 3 and 4 detail the ASTRO and ASTRO-DF algorithms respectively, and form the primary contributions of this dissertation. In Chapter 5, we discuss a long list of heuristics that have proven effective in implementing our proposed algorithms, particularly for the derivative-free context. Finally in Chapter 6 we summarize the thesis and conclude with suggestions for future research.

2. DETERMINISTIC TRUST-REGION OPTIMIZATION

In this chapter, we provide a brief overview of trust-region methods for the deterministic context, or DTRO, to facilitate exposition in the SO context. DTRO methods have recently become very popular as stable numerical optimization methods that guarantee the (global) attainment of a first-order or second-order stationary point. We will first review the derivative-based DTRO framework in Section 2.1. Our presentation and notation closely follow that of Nocedal and Wright [48]. In Section 2.2 we discuss derivative-free DTRO, or DTRO-DF, along with steps to prove convergence of its iterates.

2.1 The General DTRO Framework

In the basic derivative-based DTRO, a sequence of models are constructed at each iteration to approximate the objective function in some "trustable neighborhood" around the incumbent solution $\boldsymbol{x}_k$. Such models are usually simpler functions that can be minimized with reasonably low effort. Minimization of the constructed model yields a candidate point $\tilde{\boldsymbol{x}}_{k+1}$ lying within the trustable neighborhood. Such identification is followed by a crucial step involving the evaluation of $\tilde{\boldsymbol{x}}_{k+1}$ using the predicted and actual reductions in the objective function value. Depending on the outcome of such evaluation, the candidate point is either accepted or rejected and the incumbent trust region shrunk or expanded. The process continues and can be shown to generate a sequence of iterates that converge to a statinary point under reasonable structural assumptions on the objective function. In what follows, we provide more details on this procedure along with key ideas and steps involved in demonstrating the consistency of iterates generated by DTRO.

2.1.1 Definitions

The following are the frequently used terms in the DTRO framework. Recall that we use $\boldsymbol{g}$ and $\mathscr{B}$ for function gradient and approximation of the Hessian.

Definition 1. *(Cauchy Point) $\boldsymbol{s}_k^c$ is called the Cauchy point if*

$$\boldsymbol{s}_k^c = -\tau_k \frac{\Delta_k}{\|\boldsymbol{g}_k\|} \boldsymbol{g}_k$$

where

$$\tau_k = \begin{cases} 1 & \text{if } \boldsymbol{g}_k^T \mathscr{B}_k \boldsymbol{g}_k \leq 0, \\ \min\left\{ \frac{\|\boldsymbol{g}_k\|^3}{\boldsymbol{g}_k^T \mathscr{B}_k \boldsymbol{g}_k}, 1 \right\} & \text{otherwise.} \end{cases}$$

In other words the Cauchy point is in the direction of the steepest descent if the model is not convex, and the unconstrained minimizer or the boundary value if the model is convex. Later we will show a result about the reduction obtained in the model by taking $\boldsymbol{s}_k^c$, known as the Cauchy reduction.

Definition 2. *(Level Sets) Define the level set*

$$\mathcal{S} = \{\boldsymbol{x} |\ f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\}$$

and

$$\mathcal{S}(R_0) = \{\boldsymbol{x} |\ \|\boldsymbol{x} - \boldsymbol{y}\| \leq R_0 \text{ for some } \boldsymbol{y} \in \mathcal{S}\}$$

as well-defined sets.

Definition 3. *(Lipschitz Continuous Gradients) The function f has Lipschitz continuous gradients on a set $\mathcal{S}(R_0)$ if $\|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{y})\| \leq \nu_{gL} \|\boldsymbol{x} - \boldsymbol{y}\|$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}(R_0)$, where $\boldsymbol{g}$ is the function gradient. ν_{gL} is called the Lipschitz constant of the gradient.*

2.1.2 Algorithm Listing

We list the steps of the derivative-based DTRO in Algorithm 1. Assuming the model function and gradient are available at every point, in the trust-region framework one uses a local model of the form

$$m_k(\boldsymbol{s}) = f_k + \boldsymbol{g}_k^T \boldsymbol{s} + \frac{1}{2} \boldsymbol{s}^T \mathscr{B}_k \boldsymbol{s}, \tag{2.1}$$

where $f_k = f(\boldsymbol{x}_k), \boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$ and $\mathscr{B}_k$ is a symmetric matrix. If f is twice differentiable and the Hessian information is also available, $\mathscr{B}_k$ can be replaced with the Hessian matrix. This model is optimized inside a closed ball $\mathcal{B}(\boldsymbol{x}_k;\Delta_k) = \left\{\boldsymbol{x} \in \mathbb{R}^d | \ \|\boldsymbol{x}-\boldsymbol{x}_k\| \leq \Delta_k\right\}$, where $\|\cdot\|$ is the ℓ_2-norm, known as the trust-region with radius Δ_k (Steps 2–3).. The global convergence of the DTRO methods demands that the constrained minimization of the local model provides sufficient reduction in the model at $\boldsymbol{s}_k$. The reduction in the model by moving from the current iterate to the candidate point must be at least a fraction of the reduction that is obtained by moving to the Cauchy point (Definition 1), in order for DTRO to converge. Then we evaluate the function at this candidate point and calculate the ratio of the actual reduction to the predicted reduction and call it success ratio

$$\rho_k = \frac{f_k - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{0}) - m_k(\boldsymbol{s}_k)}$$

that indicated the goodness of the model (Step 4). Next we adjust the trust-region radius based on the ratio of the actual objective value "reduction" to the objective value reduction predicted by the model (Steps 5–11). If the model is not "good enough," we shrink the trust-region radius. If the model is "good enough" *and* the new candidate point lies on the boundary of the trust-region, we expand the trust-region radius to obtain larger steps. If the model is "good enough" but the candidate point is not on the boundary, the trust-region radius remains the same. Finally we determine whether to accept the candidate point (Steps 12–16). If the actual objective value reduces sufficiently at the candidate point, iteration k is deemed *successful* and we accept the point as the new iterate, that is $\boldsymbol{x}_{k+1} = \tilde{\boldsymbol{x}}_{k+1}$. Otherwise, iteration k is deemed *unsuccessful* and we remain at $\boldsymbol{x}_k$ with a reduced trust-region radius.

Figure 2.1 illustrates an example of this procedure. Panel (a) depicts the beginning of iteration k with the incumbent solution $\boldsymbol{x}_k$ and trust-region radius Δ_k. A model is constructed and optimized within $\mathcal{B}(\boldsymbol{x}_k;\Delta_k)$ to obtain the model minimizer. In panel (b), the function is evaluated at the model minimizer and is accepted as the new incumbent solution for iteration $k+1$, and the trust-region radius is expanded for the following iteration. Then similar to the previous iteration, a new model is constructed at $\boldsymbol{x}_{k+1}$ and optimized within $\mathcal{B}(\boldsymbol{x}_{k+1};\Delta_{k+1})$ to obtain the model minimizer. This time, since a sufficient decrease was

Algorithm 1 Deterministic Trust-Region Optimization (DTRO) Algorithm

Require: Initial guess $\boldsymbol{x}_0 \in \mathbb{R}^d$, initial trust-region radius $\Delta_0 > 0$ and maximum radius $\Delta_{\max} > 0$, model "fitness" thresholds $0 < \eta_3 < \eta_2 < 0.5 < \eta_1$, trust-region expansion constant $\gamma_1 > 1$ and contraction constant $\gamma_2 \in (0,1)$.

1: **for** $k = 0, 1, 2, \ldots$ **do**

Construct, optimize, and evaluate the model in the trust-region:

2: Construct the quadratic model $m_k(\boldsymbol{p}) = f(\boldsymbol{x}_k) + \boldsymbol{g}^T(\boldsymbol{x}_k)\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T \mathscr{B}_k \boldsymbol{p}$ at the current point $\boldsymbol{x}_k$.

3: Obtain the kth step by minimizing the model over the trust-region radius Δ_k, $\boldsymbol{p}_k = \operatorname{argmin}_{\|\boldsymbol{p}\| \le \Delta_k} m_k(\boldsymbol{p})$.

4: Evaluate success ratio

$$\rho_k = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{p}_k)}{m_k(\boldsymbol{0}) - m_k(\boldsymbol{p}_k)}. \quad \begin{array}{l} \leftarrow \text{actual objective value "reduction;" could be positive or negative} \\ \leftarrow \text{objective value reduction predicted by quadratic model; always positive} \end{array}$$

Adjust the trust-region radius Δ_k based on the success ratio ρ_k:

5: **if** $\rho_k < \eta_2 < 0.5$ **then** {If the actual objective value reduction is much less than the model predicted,}

6: $\Delta_{k+1} = \gamma_2 \Delta_k$. {the model is not "good enough;" reduce Δ_k by a factor $\gamma_2 \in (0,1)$.}

7: **else if** $\rho_k > \eta_1 \ge 0.5$, $\|\boldsymbol{p}_k\| = \Delta_k$ **then** {If the model is "good enough" *and* the step size equals Δ_k,}

8: $\Delta_{k+1} = \min\{\gamma_1 \Delta_k, \Delta_{\max}\}$. {increase Δ_k by a factor $\gamma_1 > 1$, ensuring the new radius is not larger than $\Delta_{\max}$.}

9: **else** {If the model is "good enough" but the step size was less than Δ_k,}

10: $\Delta_{k+1} = \Delta_k$. {keep the trust-region radius the same.}

11: **end if**

Determine whether to accept the step determined by optimizing the model in the trust-region:

12: **if** $\rho_k > \eta_3$ **then** {If the actual objective value reduces "enough" relative to the model ($0 < \eta_3 < \eta_2 < 0.5$),}

13: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$. {accept the step specified by $\boldsymbol{p}_k$.}

14: **else** {If the actual objective value does not reduce or if the reduction is too small relative to the model,}

15: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$. {remain at $\boldsymbol{x}_k$; since $\rho_k < \eta_3 < \eta_2$, then Δ_k was reduced in Step 6.}

16: **end if**

17: **end for**

not observed, he next iterate $\boldsymbol{x}_{k+2}$ (depicted in panel (c)) will remain the current incumbent solution and the trust region radius for the proceeding iteration Δ_{k+2} shrinks. Then a new model is constructed at the new iterate and optimized within $\mathcal{B}(\boldsymbol{x}_{k+2}; \Delta_{k+2})$. Panel

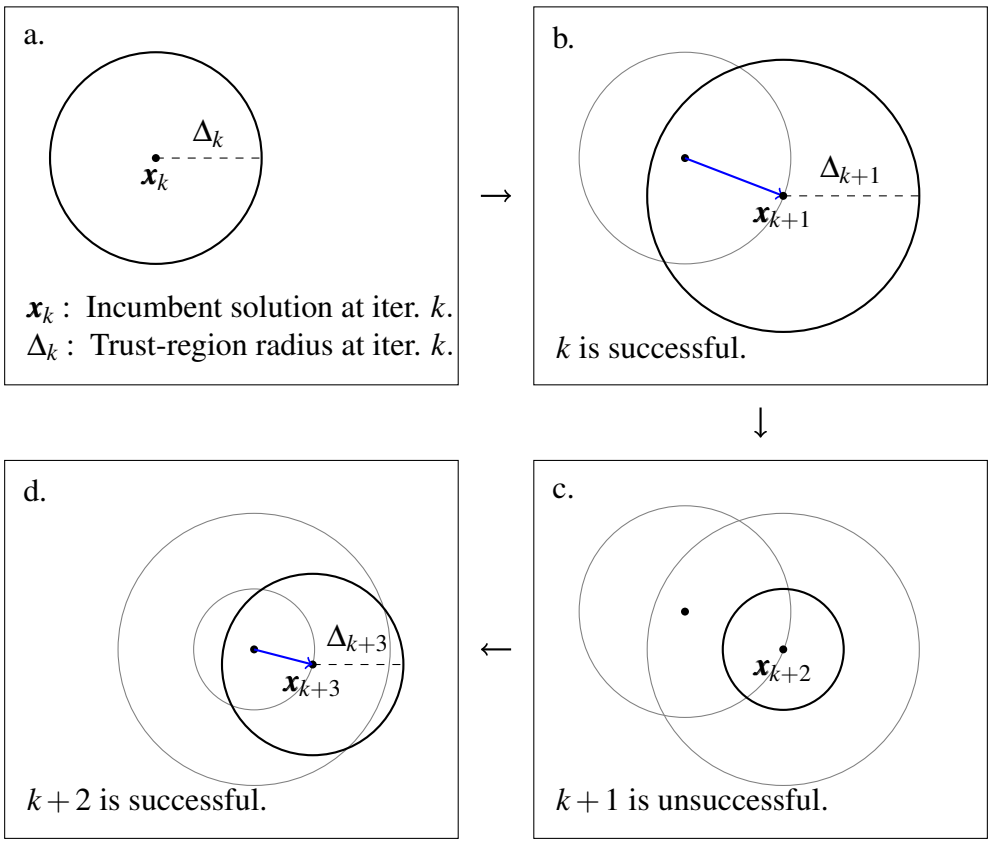


Figure 2.1. Deterministic Trust-Region Illustration.

(d) shows that the new model's minimizer is a new incumbent solution for the function and hence the trust-region moves to the new center point $\boldsymbol{x}_{k+3}$ and expands for iteration $k+3$.

2.1.3 Convergence Results

The key minimum requirement for global convergence is sufficient reduction during the optimization step, as we explain soon. Assuming Cauchy reduction, a natural "lock step" between the trust-region radius Δ_k and the function gradient is maintained. In addition, the efficiency of the deterministic DTRO algorithms is dependent on the trust-region subproblem, particularly in high dimensions. Numerical methods such as dogleg method and two-dimensional subspace minimization have been suggested to decrease the computation

under certain settings [48]. It has been shown in [43] that further assumptions on the second-order derivatives of the model and function keep Δ_k bounded away from zero and the iterations eventually expanding. Therefore the rate of convergence solely depends on the method used for model step computation. For example in the case of $\mathscr{B}_k$ being the Hessian and $\|\boldsymbol{s}_k\| < \Delta_k$ the DTRO converges quadratically.

Throughout the search we make the following assumptions:

Assumption 1. *There exists a uniform bound κ_{bhm} such that $\|\mathscr{B}_k\| \leq \kappa_{bhm}$ for all k.*

Assumption 2. *In Step 1 of Algorithm 1, the solution to the model step problem for some $c_1 \in (0,1)$ satisfies*

$$m_k(\mathbf{0}) - m_k(\boldsymbol{s}_k) \geq c_1 \|\boldsymbol{g}_k\| \min\left\{\Delta_k, \frac{\|\boldsymbol{g}_k\|}{\|\mathscr{B}_k\|}\right\}. \tag{2.2}$$

In the case of using Cauchy point $c_1 = 1/2$ (known as Cauchy reduction), shown in the next Lemma.

Lemma 1. *The Cauchy point $\boldsymbol{s}_k^c$ satisfies (2.2) with $c_1 = 1/2$.*

Proof. In this proof we remove the subscript k for readability. When $\boldsymbol{g}^T\mathscr{B}\boldsymbol{g} \leq 0$ we have

$$m(\boldsymbol{s}^c) - m(\mathbf{0}) = -\Delta\|\boldsymbol{g}\| + \frac{1}{2}\frac{\Delta^2}{\|\boldsymbol{g}\|}\boldsymbol{g}^T\mathscr{B}\boldsymbol{g} \leq -\Delta\|\boldsymbol{g}\|.$$

When $\boldsymbol{g}^T\mathscr{B}\boldsymbol{g} > 0$ and $\frac{\|\boldsymbol{g}\|^3}{\boldsymbol{g}^T\mathscr{B}\boldsymbol{g}} \leq 1$, then

$$m(\boldsymbol{s}^c) - m(\mathbf{0}) = -\frac{\|\boldsymbol{g}\|^4}{\boldsymbol{g}^T\mathscr{B}\boldsymbol{g}} + \frac{1}{2}\frac{\boldsymbol{g}^T\mathscr{B}\boldsymbol{g}\,\|\boldsymbol{g}\|^4}{(\boldsymbol{g}^T\mathscr{B}\boldsymbol{g})^2} \leq -\frac{1}{2}\frac{\|\boldsymbol{g}\|^2}{\|\mathscr{B}\|}.$$

Finally when $\boldsymbol{g}^T\mathscr{B}\boldsymbol{g} > 0$ and $\frac{\|\boldsymbol{g}\|^3}{\boldsymbol{g}^T\mathscr{B}\boldsymbol{g}} > 1$ we have

$$m(\boldsymbol{s}^c) - m(\mathbf{0}) = -\Delta\|\boldsymbol{g}\| + \frac{1}{2}\frac{\Delta^2}{\|\boldsymbol{g}\|^2}\boldsymbol{g}^T\mathscr{B}\boldsymbol{g} < -\frac{1}{2}\Delta\|\boldsymbol{g}\|,$$

since $\boldsymbol{g}^T\mathscr{B}\boldsymbol{g} < \frac{\|\boldsymbol{g}\|^3}{\Delta}$. ∎

Note that when $\Delta_k < \frac{\|\boldsymbol{g}_k\|}{\|\mathscr{B}_k\|}$ then the Cauchy reduction looks like Armijo condition in the line search methods where the reduction is proportional to the gradient and the step size.

The following results from [48] show the lim-inf type convergence of the DTRO under the assumption of sufficient decrease and uniformly bounded $\mathscr{B}_k$.

Theorem 3. *Let $\eta_3 = 0$ in Algorithm 1 and Assumptions 1 and 2 hold for every iteration k. Suppose f is bounded from below on the level set $\mathcal{S}$, and has Lipschitz continuous gradients on $\mathcal{S}(R_0)$ for some $R_0 > 0$. Then $\liminf_{k\to\infty} \|\boldsymbol{g}_k\| = 0$.*

Proof. Using Taylor's theorem we can write

$$|\rho_k - 1| = \left| \frac{m_k(\boldsymbol{s}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{0}) - m_k(\boldsymbol{s}_k)} \right| \leq \frac{\frac{1}{2}\Delta_k^2 (\kappa_{bhm} + \nu_{gL})}{c_1 \|\boldsymbol{g}_k\| \min\left\{\Delta_k, \frac{\|\boldsymbol{g}_k\|}{\kappa_{bhm}}\right\}}$$

where ν_{gL} is the Lipschitz constant of the gradient on $\mathcal{S}(R_0)$. Suppose there exists $\varepsilon > 0, K$ such that $\|\boldsymbol{g}_k\| \geq \varepsilon$ for $k \geq K$. Then

$$|\rho_k - 1| \leq \frac{\frac{1}{2}\Delta_k^2 (\kappa_{bhm} + \nu_{gL})}{c_1 \varepsilon \min\left\{\Delta_k, \frac{\varepsilon}{\kappa_{bhm}}\right\}}.$$

Let $\bar{\Delta} = \min\left\{R_0, \frac{c_1\varepsilon}{(\kappa_{bhm}+\nu_{gL})}\right\}$. If $\Delta_k < \bar{\Delta}$, then $\Delta_k < \varepsilon/\kappa_{bhm}$ since $c_1 < 1$ and hence $|\rho_k - 1| \leq 1/2$. As a result we must have that $\rho_k > \eta_2$ which implies that $\Delta_{k+1} \geq \Delta_k$. In fact we can write

$$\Delta_k \geq \min\left\{\Delta_K, \bar{\Delta}\gamma_2\right\} \tag{2.3}$$

for all $k \geq K$.

Now let's suppose that there is a subsequence $\mathcal{K}$ such that $\rho_k > \eta_2$ for all $k \in \mathcal{K}$. It follows from

$$f_k - f_{k+1} \geq \eta_2 c_1 \varepsilon \min\{\Delta_k, \varepsilon/\kappa_{bhm}\}$$

that since f_k is bounded from below and decreasing, $\Delta_k \to 0$ as $k \to \infty$ which contradicts (2.3). Therefore such a subsequence cannot exist. Now suppose $\rho_k < \eta_2$ and hence $\Delta_{k+1} < \Delta_k$ for sufficiently large k which also contradicts (2.3). This proves the statement of the theorem. ■

Finally the lim-type convergence result is as follows:

Theorem 4. *Let $\eta_2 > \eta_3 > 0$ in Algorithm 1 Assumptions 1 and 2 hold for every iteration k. Suppose f is bounded from below on the level set $\mathcal{S}$, and has Lipschitz continuous gradients on $\mathcal{S}(R_0)$ for some $R_0 > 0$. Then $\lim_{k\to\infty} \|\boldsymbol{g}_k\| = 0$.*

Proof. Let m be such that $\|\boldsymbol{g}_m\| \neq 0$. Then define $\varepsilon = \|\boldsymbol{g}_m\|/2$ and $R = \min\{R_0, \varepsilon/\nu_{gL}\}$ where ν_{gL} is the Lipschitz constant of $\boldsymbol{g}$ on $\mathcal{S}(R_0)$. We know if $\boldsymbol{x} \in \mathcal{B}(\boldsymbol{x}_m, R)$ then

$$\|\boldsymbol{g}(\boldsymbol{x})\| \geq \|\boldsymbol{g}_m\| - \|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}_m\| \geq 2\varepsilon - \nu_{gL} R \geq \varepsilon.$$

Due to Theorem 3 we cannot have $\boldsymbol{x}_k \in \mathcal{B}(\boldsymbol{x}_m, R)$ for all $k \geq m$. So $\{\boldsymbol{x}_k\}_{k \geq m}$ eventually leaves $\mathcal{B}(\boldsymbol{x}_m, R)$. Letting $\ell \geq m$ be such that $\boldsymbol{x}_{\ell+1}$ is the first iteration after $\boldsymbol{x}_m$ that is outside $\mathcal{B}(\boldsymbol{x}_m, R)$ we have $\|\boldsymbol{g}_k\| \geq \varepsilon$ for $k = m, m+1, \cdots, \ell$ and therefore

$$\begin{aligned} f(\boldsymbol{x}_m) - f(\boldsymbol{x}_{\ell+1}) &= \sum_{k=m}^{\ell} f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \\ &\geq \sum_{\substack{k=m \\ \boldsymbol{x}_k \neq \boldsymbol{x}_{k+1}}}^{\ell} \eta_3 c_1 \varepsilon \min\left\{\Delta_k, \frac{\varepsilon}{\kappa_{bhm}}\right\}. \end{aligned}$$

If $\Delta_k \leq \frac{\varepsilon}{\kappa_{bhm}}$ for all $k = m, m+1, \cdots, \ell$ then $f(\boldsymbol{x}_m) - f(\boldsymbol{x}_{\ell+1}) \geq \eta_3 c_1 \varepsilon R$. If $\Delta_k > \frac{\varepsilon}{\kappa_{bhm}}$ for some $k = m, m+1, \cdots, \ell$ then $f(\boldsymbol{x}_m) - f(\boldsymbol{x}_{\ell+1}) \geq \eta_3 c_1 \varepsilon \frac{\varepsilon}{\kappa_{bhm}}$.

On the other hand since $f(\boldsymbol{x}_k)$ is decreasing and bounded below we know $f(\boldsymbol{x}_k) \searrow f^*$ for some $f^* > -\infty$. We conclude that

$$\begin{aligned} f(\boldsymbol{x}_m) - f^* &\geq f(\boldsymbol{x}_m) - f(\boldsymbol{x}_{\ell+1}) \\ &\geq \eta_3 c_1 \varepsilon \min\left\{\frac{\varepsilon}{\nu_{gL}}, \frac{\varepsilon}{\kappa_{bhm}}, R_0\right\} \\ &= \eta_3 c_1 \frac{\|\boldsymbol{g}_m\|}{2} \min\left\{\frac{\|\boldsymbol{g}_m\|}{2\nu_{gL}}, \frac{\|\boldsymbol{g}_m\|}{2\kappa_{bhm}}, R_0\right\} \end{aligned}$$

which implies that $\|\boldsymbol{g}_m\| \to 0$ as $m \to \infty$. ■

2.2 DTRO — Derivarive-Free

The model-based DTRO derivative-free algorithms, which we call DTRO-DF for short, have received notable attention as methods for solving optimization problems in low to moderate dimensions. They are "derivative-free" in the sense that function derivatives are not directly observed, even though they are inferred through the constructed local model. ("Derivative-free" optimization is often confused with "derivative-less" or non-differentiable optimization, where the objective function is not differentiable.)

Popularity of DTRO-DF algorithms stems primarily from their stability derived through the use of a trust-region and the consequent eschewing of large steps near the solution. Such a conservative approach hurts convergence rates but has come to be favored in optimization settings where the objective function is poorly behaved and explicit derivative estimation is expensive. While their predecessors, DTRO algorithms, have existed for several decades and represent a mature class of algorithms, DTRO-DF methods are still undergoing development on a number of subtle unresolved issues (e.g., convergence rates, nature of the trust-region model and subproblem optimization). Their utility is, nevertheless, unquestionable particularly as measured by their use in "real-world" settings [5,45,46].

2.2.1 Definitions

For DTRO-DF we follow the same framework as in [44] where the model is defined to have the same origin as the function as shown in the following.

Definition 4. *(Lagrange Polynomials) Let $\mathcal{P}_d^d$ be the space of polynomials of degree $\leq d$ in $\mathbb{R}^d$ and p be the dimension of this space. Given a set $\mathcal{Y} = \{\boldsymbol{y}_i \in \mathcal{B}(\boldsymbol{x};\Delta),\ i = 1,\ldots,p\}$, a basis of p polynomials $\ell_j(\boldsymbol{x})$, $j = 1,\ldots,p$ in $\mathcal{P}_d^d$, is called a basis of Lagrange polynomials if*

$$\ell_j(\boldsymbol{y}_i) = \delta_{ij} = \begin{cases} 1 & if\ i = j \\ 0 & if\ i \neq j \end{cases}.$$

Definition 5. *(Poised and Λ-Poised Sets) Let $\mathcal{Y} = \{\boldsymbol{y}_i \in \mathcal{B}(\boldsymbol{x};\Delta),\ i = 1,\ldots,p\}$ be a finite set of points given $\boldsymbol{x} \in \mathbb{X}$ and $\Delta > 0$, and $\Phi(\boldsymbol{z}) = \left(\phi^1(\boldsymbol{z}), \phi^2(\boldsymbol{z}), \ldots, \phi^q(\boldsymbol{z})\right)$ of $\mathcal{P}_d^d$ be a polynomial basis on $\mathbb{X} \subseteq \mathbb{R}^d$. Then define*

$$\mathscr{P}(\Phi,\mathcal{Y}) = \begin{bmatrix} \phi^1(\boldsymbol{y}_1) & \phi^2(\boldsymbol{y}_1) & \ldots & \phi^q(\boldsymbol{y}_1) \\ \phi^1(\boldsymbol{y}_2) & \phi^2(\boldsymbol{y}_2) & \ldots & \phi^q(\boldsymbol{y}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi^1(\boldsymbol{y}_p) & \phi^2(\boldsymbol{y}_p) & \ldots & \phi^q(\boldsymbol{y}_p) \end{bmatrix}.$$

$\mathcal{Y}$ is said to be a "poised set" in $\mathcal{B}(\boldsymbol{x};\Delta)$ if the matrix $\mathscr{P}(\Phi,\mathcal{Y})$ is nonsingular. If $\mathcal{Y}$ is poised then Lagrange polynomials exist, are unique, and have a number of useful proper-

ties. A poised set $\mathcal{Y}$ is said to be "Λ-poised" in $\mathcal{B}(\boldsymbol{x};\Delta)$ if and only if given $\Lambda > 0$, for the basis of Lagrange polynomials associated with $\mathcal{Y}$,

$$\Lambda \geq \max_{j=1,\cdots,p} \max_{\boldsymbol{z}\in\mathcal{B}(\boldsymbol{x};\Delta)} \left|\ell_j(\boldsymbol{z})\right|.$$

Remark 2. *Given the matrix $\mathscr{P}(\Phi,\mathcal{Y})$, the value of the Lagrange polynomials can be calculated by*

$$|\ell_i(\boldsymbol{z})| = \frac{|\det(\mathscr{P}(\Phi,\mathcal{Y}_i(\boldsymbol{z})))|}{|\det(\mathscr{P}(\Phi,\mathcal{Y}))|} = \frac{\mathrm{vol}(\Phi(\mathcal{Y}_i(\boldsymbol{z})))}{\mathrm{vol}(\Phi(\mathcal{Y}))},$$

where $\mathcal{Y}_i(\boldsymbol{z}) = \mathcal{Y}\backslash\{\boldsymbol{y}_i\}\cup\boldsymbol{z}$. In other words the absolute value of the i^{th} Lagrange polynomial at a given point $\boldsymbol{z}$ is the change in the volume of the simplex of vertices in $\Phi(\mathcal{Y})$ when $\boldsymbol{z}$ replaces $\boldsymbol{y}_i$ [44, p. 41].

Definition 6. *(Polynomial Interpolation Models) Let $f:\mathbb{X}\subseteq\mathbb{R}^d\to\mathbb{R}$ be a real-valued function and let $\mathcal{Y}$ and Φ be as defined in Definition 5 with $p=q$. We can find a set of scalars $\boldsymbol{\alpha}=(\alpha^1,\ldots,\alpha^p)$ such that $\mathscr{P}(\Phi,\mathcal{Y})\boldsymbol{\alpha} = \left(f(\boldsymbol{y}_1),\ldots,f(\boldsymbol{y}_p)\right)^T$. Such an $\boldsymbol{\alpha}$ is guaranteed to exist if $\mathscr{Y}$ is poised. Then a function $m(\boldsymbol{z}):\mathcal{B}(\boldsymbol{x};\Delta)\to\mathbb{R}$ with $m(\boldsymbol{z}) = \sum_{j=1}^{p}\alpha^j\phi^j(\boldsymbol{z})$ is said to be a polynomial interpolation model of f on $\mathcal{B}(\boldsymbol{x};\Delta)$. The following special cases are of particular importance:*

(i) $m(\boldsymbol{z})$ is said to be a linear interpolation model of f on $\mathcal{B}(\boldsymbol{x};\Delta)$, if $p=d+1$ and $\Phi(\boldsymbol{z})$ is the linear basis on $\mathbb{X}\subseteq\mathbb{R}^d$, that is, $\Phi(\boldsymbol{z}) = \left(1,z^1,z^2,\ldots,z^d\right)$. For simplicity we denote $\mathscr{P}(\Phi,\mathcal{Y})$ as $\mathscr{L}(\mathcal{Y})$, and we have

$$\mathscr{L}(\mathcal{Y}) = \begin{bmatrix} 1 & y_1^1 & y_1^2 & \cdots & y_1^d \\ 1 & y_2^1 & y_2^2 & \cdots & y_2^d \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & y_{d+1}^1 & y_{d+1}^2 & \cdots & y_{d+1}^d \end{bmatrix}.$$

We find $\left(\alpha^1,\ldots,\alpha^{d+1}\right)^T$ by solving $\mathscr{L}(\mathcal{Y})\left(\alpha^1,\cdots,\alpha^{d+1}\right)^T = \left(f(\boldsymbol{y}_1),\cdots,f(\boldsymbol{y}_{d+1})\right)^T$. For all $\boldsymbol{z}\in\mathcal{B}(\boldsymbol{x};\Delta)$,

$$m(\boldsymbol{z}) = \alpha^1 + \boldsymbol{z}^T\boldsymbol{g}$$

where $\boldsymbol{g} = \left(\alpha^2, \ldots, \alpha^{d+1}\right)^T$. *The gradient of the linear interpolation model is* $\nabla m(\boldsymbol{z}) = \boldsymbol{g}$ *and the Hessian of the linear interpolation model is* $\nabla^2 m(\boldsymbol{z}) = \mathbf{0}$. *Therefore using Taylor expansion we can write*

$$m(\boldsymbol{x}+\boldsymbol{s}) = m(\boldsymbol{x}) + \boldsymbol{s}^T \boldsymbol{g}$$

for all $\boldsymbol{s} \in \mathcal{B}(\mathbf{0};\Delta)$. *In the context of Lagrange polynomials we say that if* $m(\boldsymbol{z})$ *interpolates* $f(\boldsymbol{z})$ *at points of* $\mathcal{Y}$, *then*

$$m(\boldsymbol{z}) = \sum_{j=1}^{p} \ell_j(\boldsymbol{z}) f\left(\boldsymbol{y}_j\right),$$

for all $\boldsymbol{z} \in \mathbb{X}$.

(ii) $m(\boldsymbol{z})$ *is said to be a quadratic interpolation model of* f *on* $\mathcal{B}(\boldsymbol{x};\Delta)$ *if* $p = (d+1)(d+2)/2$ *and* $\Phi(\boldsymbol{z})$ *is a quadratic basis on* $\mathbb{X} \subseteq \mathbb{R}^d$. *The monomial quadratic basis is* $\Phi(\boldsymbol{z}) = \left(1, z^1, \ldots, z^d, \frac{1}{2}(z^1)^2, z^1 z^2, \ldots, \frac{1}{2}(z^d)^2\right)$. *For simplicity we denote* $\mathscr{P}(\Phi, \mathcal{Y})$ *as* $\mathscr{Q}(\mathcal{Y})$, *and we have*

$$\mathscr{Q}(\mathcal{Y}) = \begin{bmatrix} 1 & y_1^1 & \ldots & y_1^d & \frac{1}{2}\left(y_1^1\right)^2 & y_1^1 y_1^2 & \ldots & \frac{1}{2}\left(y_1^d\right)^2 \\ 1 & y_2^1 & \ldots & y_2^d & \frac{1}{2}\left(y_2^1\right)^2 & y_2^1 y_2^2 & \ldots & \frac{1}{2}\left(y_2^d\right)^2 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & y_p^1 & \ldots & y_p^d & \frac{1}{2}\left(y_p^1\right)^2 & y_p^1 y_p^2 & \ldots & \frac{1}{2}\left(y_p^d\right)^2 \end{bmatrix}.$$

For example when $d = 2$ *and* $p = 6$, *we have*

$$\mathscr{Q}(\mathcal{Y}) = \begin{bmatrix} 1 & y_1^1 & y_1^2 & \frac{1}{2}\left(y_1^1\right)^2 & y_1^1 y_1^2 & \frac{1}{2}\left(y_1^2\right)^2 \\ 1 & y_2^1 & y_2^2 & \frac{1}{2}\left(y_1^2\right)^2 & y_2^1 y_2^2 & \frac{1}{2}\left(y_2^2\right)^2 \\ 1 & y_3^1 & y_3^2 & \frac{1}{2}\left(y_1^3\right)^2 & y_3^1 y_3^2 & \frac{1}{2}\left(y_3^2\right)^2 \\ 1 & y_4^1 & y_4^2 & \frac{1}{2}\left(y_1^4\right)^2 & y_4^1 y_4^2 & \frac{1}{2}\left(y_4^2\right)^2 \\ 1 & y_5^1 & y_5^2 & \frac{1}{2}\left(y_1^5\right)^2 & y_5^1 y_5^2 & \frac{1}{2}\left(y_5^2\right)^2 \\ 1 & y_6^1 & y_6^2 & \frac{1}{2}\left(y_1^6\right)^2 & y_6^1 y_6^2 & \frac{1}{2}\left(y_6^2\right)^2 \end{bmatrix}.$$

We find $\left(\alpha^1, \ldots, \alpha^p\right)^T$ *by solving* $\mathscr{Q}(\mathcal{Y})\left(\alpha^1, \ldots, \alpha^p\right)^T = \left(f(\boldsymbol{y}_1), \ldots, f\left(\boldsymbol{y}_p\right)\right)^T$.

For all $\boldsymbol{z} \in \mathcal{B}(\boldsymbol{x};\Delta)$,

$$m(\boldsymbol{z}) = \alpha^1 + \boldsymbol{z}^T \boldsymbol{g} + \frac{1}{2}\boldsymbol{z}^T \mathscr{B} \boldsymbol{z},$$

where $\boldsymbol{g} = \left(\alpha^2, \ldots, \alpha^{d+1}\right)^T$ *and* $\mathscr{B}$ *is a symmetric positive-definite matrix. The gradient of the quadratic interpolation model is* $\nabla m(\boldsymbol{z}) = \boldsymbol{g} + \mathscr{B}\boldsymbol{z}$ *and the Hessian of the quadratic interpolation model is* $\nabla^2 m(\boldsymbol{z}) = \mathscr{B}$*. Therefore using Taylor expansion we can write*

$$m(\boldsymbol{x}+\boldsymbol{s}) = m(\boldsymbol{x}) + \boldsymbol{s}^T(\boldsymbol{g} + \mathscr{B}\boldsymbol{s}) + \frac{1}{2}\boldsymbol{s}^T\mathscr{B}\boldsymbol{s}, \tag{2.4}$$

for all $\boldsymbol{s} \in \mathcal{B}(\mathbf{0};\Delta)$.

(iii) For higher order polynomial interpolation models see [44, p. 35].

When referring to the model $m(\boldsymbol{z}) : \mathcal{B}(\boldsymbol{x};\Delta) \to \mathbb{R}$ *of the function* $f(\boldsymbol{x})$*, we will often drop the phrase "of the function* $f(\boldsymbol{x})$*" unless the context is unclear.*

In the derivative-free context we often use the model defined in (2.4). Notice that this definition of the model is different from that for the derivative-based context defined in (2.1). The reason we use different definitions for the model in the derivative-based and derivative-free contexts is to maintain our notation similar to that of the corresponding framework used for each context.

Remark 3. *When* $p \neq q$ *one option is to use regression models to compute the coefficients of the polynomial model. When* $p > q$ *the regression model is overdetermined and when* $p < q$ *the regression model is under-determined.*

Definition 7. *(Fully-linear and Fully-quadratic Models) Given* $\boldsymbol{x} \in \mathbb{X}$*,* $m(\boldsymbol{z}) : \mathcal{B}(\boldsymbol{x};\Delta) \to \mathbb{R}$*,* $m \in \mathcal{C}^1$ *is said to be a* $(\kappa_{ef}, \kappa_{eg})$*-fully-linear model of* f *on* $\mathcal{B}(\boldsymbol{x};\Delta)$ *if it has Lipschitz continuous gradient with Lipschitz constant* ν_{gL}^m*, and there exists set of constants* κ_{ef} *and* κ_{eg} *such that the error bounds between the model and the function, and the gradient of the model and the gradient of the function, respectively, satisfy*

$$\begin{aligned} |f(\boldsymbol{z}) - m(\boldsymbol{z})| &\leq \kappa_{ef}\Delta^2; \\ \|\nabla f(\boldsymbol{z}) - \nabla m(\boldsymbol{z})\| &\leq \kappa_{eg}\Delta, \end{aligned} \tag{2.5}$$

with κ_{ef} *and* κ_{eg} *independent of* $\boldsymbol{x}$ *and* Δ*.*

Similarly $m(\boldsymbol{z}) : \mathcal{B}(\boldsymbol{x};\Delta) \to \mathbb{R}$, $m \in \mathcal{C}^2$ *is said to be a* $\left(\kappa_{ef}, \kappa_{eg}, \kappa_{eh}\right)$*-fully-quadratic model of* f *on* $\mathcal{B}(\boldsymbol{x};\Delta)$ *if it has Lipschitz continuous Hessian with Lipschitz constant* ν^m_{HL}*, and there exist* κ_{ef}*,* κ_{eg} *and* κ_{eH} *independent of* $\boldsymbol{x}$ *and* Δ *such that*

$$\begin{aligned} |f(\boldsymbol{z}) - m(\boldsymbol{z})| &\leq \kappa_{ef}\Delta^3; \\ \|\nabla f(\boldsymbol{z}) - \nabla m(\boldsymbol{z})\| &\leq \kappa_{eg}\Delta^2; \\ \left\|\nabla^2 f(\boldsymbol{z}) - \nabla^2 m(\boldsymbol{z})\right\| &\leq \kappa_{eH}\Delta. \end{aligned} \tag{2.6}$$

A set of model functions $\mathcal{M}_{fl} = \left\{m : \mathbb{R}^d \to \mathbb{R},\ m \in \mathcal{C}^1\right\}$ *is called a fully-linear class of models and* $\mathcal{M}_{fq} = \left\{m : \mathbb{R}^d \to \mathbb{R},\ m \in \mathcal{C}^2\right\}$ *is called a fully-quadratic class of models if in addition to the conditions above there exists an algorithm that in finite uniformly bounded (with respect to* $\boldsymbol{x}$ *and* Δ*) number of steps either*

- *certifies that* m *is a* $\left(\kappa_{ef}, \kappa_{eg}\right)$*-fully-linear or* $\left(\kappa_{ef}, \kappa_{eg}, \kappa_{eH}\right)$*-fully-quadratic model on* $\mathcal{B}(\boldsymbol{x};\Delta) \to \mathbb{R}$*, or*
- *from* m *finds a* $\left(\kappa_{ef}, \kappa_{eg}\right)$*-fully-linear or* $\left(\kappa_{ef}, \kappa_{eg}, \kappa_{eH}\right)$*-fully-quadratic model* $\bar{m}$ *on* $\mathcal{B}(\boldsymbol{x};\Delta) \to \mathbb{R}$*.*

This algorithm will be referred to as a "Model Improvement Algorithm," as in [44, p. 89].

Remark 4. *In interpolation models, if the set of points on which the model is constructed is poised as in Definition 5, then the model is* $\left(\kappa_{ef}, \kappa_{eg}\right)$*-fully-linear, and if second derivatives exist and* $\mathcal{Y}$ *is* Λ*-poised with* Λ *given,* $\left(\kappa_{ef}, \kappa_{eg}, \kappa_{eh}\right)$*-fully-quadratic.*

Definition 8. *(Cauchy Reduction) Step* $\boldsymbol{s}$ *is said to achieve* κ_{fed} *fraction of Cauchy reduction for* $m(\cdot)$ *on* $\mathcal{B}(\boldsymbol{x};\Delta)$ *with some* $\Delta > 0$*, if*

$$m(\boldsymbol{x}) - m(\boldsymbol{x}+\boldsymbol{s}) \geq \frac{\kappa_{fcd}}{2}\|\nabla m(\boldsymbol{x})\| \min\left\{\frac{\|\nabla m(\boldsymbol{x})\|}{\|\nabla^2 m(\boldsymbol{x})\|}, \Delta\right\}, \tag{2.7}$$

where $\nabla m(\boldsymbol{x})$ *and* $\nabla^2 m(\boldsymbol{x})$ *are the model gradient and the model Hessian at point* $\boldsymbol{x}$*. We assume* $\|\nabla m(\boldsymbol{x})\| / \left\|\nabla^2 m(\boldsymbol{x})\right\| = +\infty$ *when* $\nabla^2 m(\boldsymbol{x}) = \boldsymbol{0}$*. Cauchy step with* $\kappa_{fcd} = 1$ *will be obtained if the model is minimized along the steepest descent direction within* $\mathcal{B}(\boldsymbol{x};\Delta)$*. The details are shown in Lemma 1.*

Definition 9. *(Set of all Possible Points) We define* $\mathcal{S}' = \bigcup_{\boldsymbol{x}\in\mathcal{S}} \mathcal{B}(\boldsymbol{x};\Delta_{\max})$ *to be a set that contains every possible point that could be generated in the DTRO-DF algorithm. Later we will assume the function having Lipschitz continuous gradients over an open domain containing this set.*

2.2.2 Algorithm Listing

In the DTRO-DF context, the function can be approximated via polynomial interpolation or regression using an appropriate set of neighboring points. The main difficulty is ensuring sufficient model quality, that is, sufficiently good bounds on the error between the model and the true function. It turns out the error bounds can be characterized purely based on the geometry of the selected neighboring points. Powell designed a heuristic algorithm called UOBYQA in [45] that involves constructing Lagrange polynomials over what he called an *adequate* set, to ensure sufficient model quality. Conn, et. al. [46] show that the necessary condition for convergence of the DTRO-DF algorithm to a local solution is that the model satisfies Taylor-like error bounds, defined through the notions of $\left(\kappa_{ef},\kappa_{eg}\right)$-fully-linear models and $\left(\kappa_{ef},\kappa_{eg},\kappa_{eH}\right)$-fully-quadratic models, as detailed in Definition 7. The constants depend on the geometry of the sample points (see Definition 5) as well as assumptions (or knowledge) on the curvature of the function, specifically function gradient's Lipschitz constant ν_{gL}, or if twice-differentiable, function Hessian's Lipschitz constant ν_{HL}. Sufficient conditions for convergence stipulate that in addition to the model quality, a lock step between the trust-region radius Δ_k and the model gradient $\nabla m_k(\boldsymbol{x}_k)$ is maintained. To summarize the main difference between DTRO and DTRO-DF the two points below are noteworthy:

(i) In DTRO the model quality becomes better when Δ becomes smaller due to Taylor expansion, whereas in DTRO-DF this is not necessarily true. We require to reduce Δ only when failing in the comparison test between the model reduction and the function reduction at the candidate point is caused by large step size and not poor model quality. Then we can say that reduction in Δ indicates improvement in the model

Algorithm 2 Deterministic DTRO-DF Algorithm

Require: Initial set ($\boldsymbol{x}_0 \in \mathbb{R}^d$, $\Delta_0 > 0$ and $m_k(\boldsymbol{x}_0 + \boldsymbol{s})$), maximum radius $\Delta_{\max} > 0$, model "fitness" thresholds $0 < \eta_0 \leq \eta_1 < 1$, trust-region expansion constant $\gamma_1 > 1$, contraction constant $\gamma_2 \in (0,1)$, criticality threshold $\varepsilon \in (0,1)$, *model improvement algorithm* from [44, ch. 6], lock-step coefficients $0 < \beta < \mu$, and model sufficiency contraction constant $w \in (0,1)$.

for $k = 0, 1, 2, \ldots$ **do**
  if $\|\boldsymbol{g}_k\| \leq \varepsilon$ and either model has insufficient quality or $\Delta_k > \mu \|\boldsymbol{g}_k\|$, **then**
    Go to Algorithm 3 to find a sufficient quality model $\tilde{m}_k$ with $\tilde{\Delta}_k \leq \mu \|\tilde{\boldsymbol{g}}_k\|$.
    Let $m_k = \tilde{m}_k$ and $\Delta_k = \min\left\{\max\left\{\tilde{\Delta}_k, \beta \|\tilde{\boldsymbol{g}}_k\|\right\}, \Delta_k\right\}$.
  end if
  Obtain the kth step $\boldsymbol{s}_k = \underset{\|\boldsymbol{s}\| \leq \Delta_k}{\arg\min}\, m_k(\boldsymbol{x}_k + \boldsymbol{s})$, and let $\tilde{\boldsymbol{x}}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$.
  Evaluate $f(\tilde{\boldsymbol{x}}_{k+1})$. Then compute

$$\rho_k = \frac{f(\boldsymbol{x}_k) - f(\tilde{\boldsymbol{x}}_{k+1})}{m_k(\boldsymbol{x}_k) - m_k(\tilde{\boldsymbol{x}}_{k+1})}.$$

  if $\rho_k \geq \eta_1$, **then**
    $\Delta_{k+1} = \min\{\gamma_1 \Delta_k, \Delta_{\max}\}$,
  else
    apply the *model improvement algorithm* to make one or more improvement steps.
    if $m_k(\cdot)$ has sufficient quality, **then**
      $\Delta_{k+1} = \gamma_2 \Delta_k$.
    else
      $\Delta_{k+1} = \Delta_k$.
    end if
  end if
  if $\rho_k \geq \eta_1$ or $\rho_k \geq \eta_0$ and $m_k(\cdot)$ has sufficient quality, **then**
    $\boldsymbol{x}_{k+1} = \tilde{\boldsymbol{x}}_{k+1}$. Update the model to include the candidate point and call it m_{k+1}.
  else
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$.
  end if
end for

Algorithm 3 Criticality Step

Require: Current model m_k (call it $m_k^{(0)}$), trust region radius Δ_k, and counter $i = 0$. Parameters from DTRO-DF: μ, w and the selected *model improvement algorithm*.

1: **repeat**
2: Set $i = i + 1$.
3: Improve $m_k^{(i-1)}$ until it is $(\kappa_{ef}, \kappa_{eg})$-fully-linear on $\mathcal{B}\left(\boldsymbol{x}_k; w^{i-1}\Delta_k\right)$ and call it $m_k^{(i)}$.
4: **until** $w^{i-1}\Delta_k \leq \mu \left\| \boldsymbol{g}_k^{(i)} \right\|$.
5: **return** $\tilde{\Delta}_k = w^{i-1}\Delta_k$ and $\tilde{m}_k = m_k^{(i)}$.

predictability. The quality of the model is codified by Taylor like error bounds as defined in Definition 7 that can be certified or obtained using a "model improvement" algorithm; see [44, ch. 3 & 6].

(ii) Unlike DTRO that keeps the trust-region radius bounded away from zero unless the iterates reach the local minimizer, in DTRO-DF the trust-region radius goes to zero when the algorithm converges to a local solution. This is due to the dual role of the trust-region radius in DTRO-DF, for it both restricts the minimizing step size in the sub-problem and specifies the region in which the points are sampled for the construction of the model. The lock step between Δ_k and $\nabla m_k(\boldsymbol{x}_k)$ provides a guarantee that close to the local solution the model is more accurate. Conn et. al. embody a so-called "criticality step" in [46] to reduce the trust-region radius until sufficient accuracy (predictability) in the model quality is achieved. This condition eventually forces Δ_k to zero.

All of the steps in DTRO-DF are listed in Algorithm 2, in which we label a model that is $\left(\kappa_{ef}, \kappa_{eg}\right)$-fully-linear or $\left(\kappa_{ef}, \kappa_{eg}, \kappa_{eH}\right)$-fully-quadratic (under the assumption of twice-differentiability of f) as a "sufficient quality model."

Step 2 in Algorithm 2 is known as the "criticality step" that invokes Algorithm 3 with small norm of the model gradient - representing a measure of stationarity - to find a poised interpolation set and consequently a sufficient quality model while keeping the trust-region radius and the model gradient in lock-step. It is notable that the model does not need to

have sufficient quality in every iteration. Specifically when $\boldsymbol{g}_k$ (the model gradient at $\boldsymbol{s} = 0$) is not small, it is possible that the algorithm proceeds with a model that does not have sufficient quality. However in the case of a small $\boldsymbol{g}_k$, the additional effort spent on model construction before the new candidate point is computed in Step 6 accelerates the overall optimization.

Adjustments to the trust-region radius Δ_k are made based on the success ratio evaluated in Step 7. Steps 8 – 17 determine that the trust-region expands whenever the reduction in the actual function is significant compared to that in the model, even if the model does not have certifiably sufficient quality. On the other hand the trust-region contracts when the true versus predicted reduction is not significant but the model has sufficient quality. In all other cases, the trust-region radius does not change. Figure 2.2 depicts the trust-region radius adjustment under different scenarios.

Acceptance of the new candidate point in the trust-region is decided upon through Steps 18 – 22, where despite failing to certify a sufficient quality model new step is accepted so long as the reduction in the function value is significant. When the new step is accepted, where we say that the iteration was successful, it replaces another point in the interpolation set and with the inclusion of the new point the model is updated. Otherwise the iteration is unsuccessful, but the model might still change due to model improvement algorithm invoked in Step 11.

In presence of noise, we combine the criticality step with the model improvement and adaptive sampling rules in a new algorithm which we will discuss in Chapter 4.

2.2.3 Convergence Results

As mentioned earlier the sufficient quality model (when close to a first-order critical point) and lock step between Δ_k and $\boldsymbol{g}_k$ is required for convergence in the results presented by [46]. In the following we brief the steps to convergence of DTRO-DF. Note that depending on the success ratio ρ_k and the quality of the model an iteration can be one of the four types shown in Figure 2.2. When the decrease in the function is not large enough and the

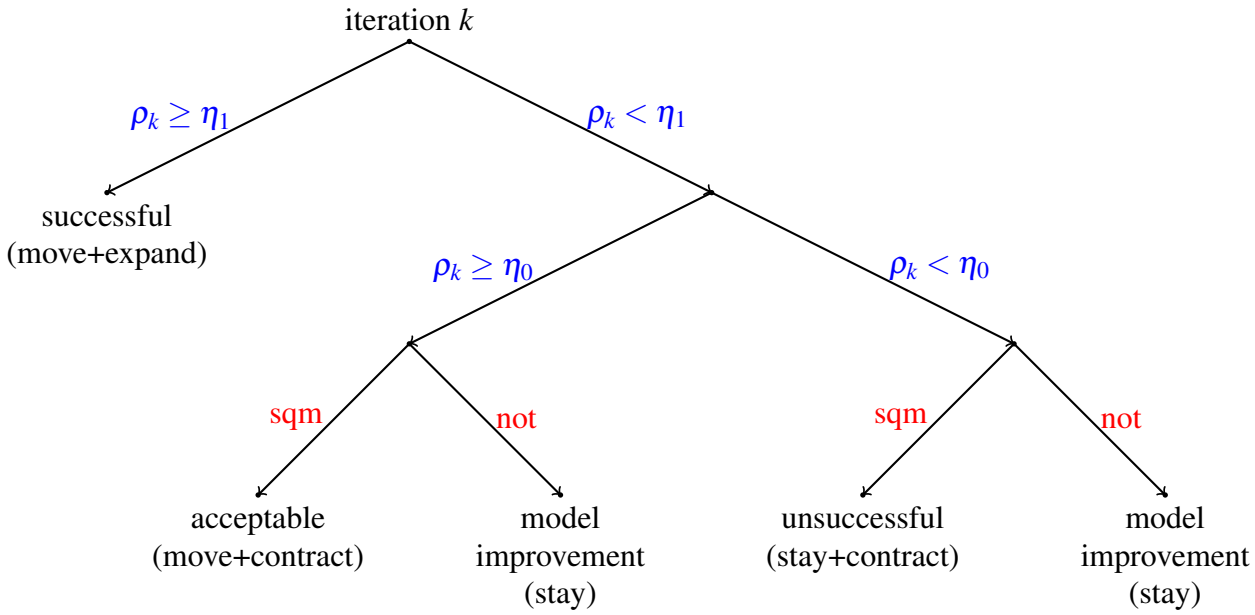


Figure 2.2. Iteration types in the deterministic DTRO-DF Algorithm. *sqm* stands for sufficient quality model. *stay* refers to $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ and *move* means $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$, while *expand* refers to $\Delta_{k+1} > \Delta_k$ and *contract* refers to $\Delta_{k+1} < \Delta_k$. When neither *expand* nor *contract*, it refers to the trust-region remaining unchanged.

model has poor quality, then no changes are made except improving the model for the next iteration ("model improvement"). But when the model quality is sufficient then failing to show enough decrease in the function implies that step size is large and hence contraction of the trust-region is enforced while the iterate remain unchanged ("unsuccessful"). Similarly sufficient model quality and enough function decrease results in updating the iterate and either expanding if the decrease is significant ("successful"), and contracting otherwise ("acceptable"). Note that when $\eta_1 = \eta_0$ there will be no acceptable iterations.

Lemma 2. *Step 1 of Algorithm 2 is terminated in finite number of steps when* $\nabla f(\boldsymbol{x}_k) \neq 0$.

Proof. If Algorithm 3 runs infinite times it means that $w^{i-1}\Delta_k > \mu_2 \left\| \boldsymbol{g}_k^{(i)} \right\|$ for all $i \geq 1$. This implies that $\left\| \boldsymbol{g}_k^{(i)} \right\| \to 0$ as $i \to \infty$. Since in each round of Algorithm 3 model $m^{(i)}$ has sufficient quality we have

$$\begin{aligned} \|\nabla f(\boldsymbol{x}_k)\| &\leq \left\| \nabla f(\boldsymbol{x}_k) - \boldsymbol{g}_k^{(i)} \right\| + \left\| \boldsymbol{g}_k^{(i)} \right\| \\ &\leq \left(\kappa_{eg} + \frac{1}{\mu_2} \right) w^{i-1} \Delta_k, \end{aligned}$$

for all $i \geq 1$. Therefore we must have $\nabla f(\boldsymbol{x}_k) = 0$. ■

In the following results suppose Assumption 1 holds and a fraction of the Cauchy decrease (Definition 8) is achieved in Step 2 of Algorithm 2.

Lemma 3. *If m_k has sufficient quality and*

$$\Delta_k \leq \min \left\{ \frac{\|\boldsymbol{g}_k\|}{\kappa_{bhm}}, \frac{\kappa_{fcd} \|\boldsymbol{g}_k\| (1-\eta_1)}{2\kappa_{ef}} \right\}$$

then the iteration k becomes successful.

Proof. First we note that $m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k) \Delta_k \geq 2^{-1} \kappa_{fcd} \|\boldsymbol{g}_k\| \Delta_k$ since $\Delta_k \leq \kappa_{bhm}^{-1} \|\boldsymbol{g}_k\|$. Next since $f(\boldsymbol{x}_k) = m_k(\boldsymbol{x}_k)$ and m_k has sufficient quality we observe the following:

$$\begin{aligned} |\rho_k - 1| &= \left| \frac{f(\boldsymbol{x}_k + \boldsymbol{s}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)} \right| \\ &\leq \frac{2\kappa_{ef} \Delta_k^2}{\kappa_{fcd} \|\boldsymbol{g}_k\| \Delta_k} \leq 1 - \eta_1. \end{aligned}$$

■

Lemma 4. *If $\|\boldsymbol{g}_k\|$ is bounded away from 0, then so is Δ_k.*

Proof. Suppose there is a constant $c_1 > 0$ such that $\|\boldsymbol{g}_k\| \geq c_1$ for all k. We know that at the end of Step 1 of every iteration, $\Delta_k \geq \mu_1 \|\boldsymbol{g}_k\|$. But we also know that if $\Delta_k < c_2$ where

$$c_2 = \min \left\{ \frac{c_1}{\kappa_{bhm}}, \frac{\kappa_{fcd} c_1 (1-\eta_1)}{2\kappa_{ef}} \right\},$$

then either iteration k is successful and Δ_{k+1} expands (when m_k has sufficient quality) or $\Delta_{k+1} = \Delta_k$ (when m_k does not have sufficient quality). So we can say $\Delta_k \geq \min\{\Delta_0, \mu_1 c_1, \gamma_1 c_2\}$.

■

Lemma 5. *Suppose that f has Lipschitz continuous gradients with Lipschitz constant κ_{bhm} in an open domain containing $\mathcal{S}'$, defined in Definition 9. If there are only finitely many successful iterations, then* $\lim_{k\to\infty} \|\nabla f(\boldsymbol{x}_k)\| = 0$.

Proof. Consider all the iterations after the last successful iteration. Suppose that there are at most N iterations until the model is fully linear (finite and uniformly bounded number of steps to full-linearity as in Definition 7). So there are finitely many acceptable or unsuccessful iteration until the next model-improving iteration (see Figure 2.2) after both of which the trust-region contracts. So we conclude that $\Delta_k \to 0$ as $k \to \infty$. Now for some iteration j let i_j be the first iteration after j that gives a sufficient quality model. First note that $\left\|\boldsymbol{x}_j - \boldsymbol{x}_{i_j}\right\| \leq N\Delta_j \to \infty$. Next we observe that

$$\left\|\nabla f\left(\boldsymbol{x}_j\right)\right\| \leq \left\|\nabla f\left(\boldsymbol{x}_j\right) - \nabla f\left(\boldsymbol{x}_{i_j}\right)\right\| + \left\|\nabla f\left(\boldsymbol{x}_{i_j}\right) - \boldsymbol{g}_{i_j}\right\| + \left\|\boldsymbol{g}_{i_j}\right\|$$

converges to zero since all the right hand side terms converge to zero by Lipschitz continuity of the gradient, model sufficient quality (fully-linearity) and Lemma 3 (since otherwise would lead to a successful iteration that is a contradiction) respectively. ■

Lemma 6. *In Algorithm 2* $\lim_{k\to\infty} \Delta_k = 0$.

Proof. Let $\mathcal{K}$ be the set of all the successful iterations. When $\mathcal{K}$ is finite the statement is proven in Lemma 5. Suppose $\mathcal{K}$ is infinite. Using Assumption 1 and knowing from Step 1 of Algorithm 2 that $\|\boldsymbol{g}_k\| \geq \min\left\{\mu_2^{-1}\Delta_k, \varepsilon\right\}$ we get

$$\begin{aligned} f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) &\geq \eta_1 \left(m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_{k+1})\right) \\ &\geq \eta_1 \frac{\kappa_{fcd}}{2} \min\left\{\mu_2^{-1}\Delta_k, \varepsilon\right\} \min\left\{\frac{\min\left\{\mu_2^{-1}\Delta_k, \varepsilon\right\}}{\kappa_{bhm}}, \Delta_k\right\}. \end{aligned}$$

We know f is bounded below, implying that $\lim_{\substack{k\to\infty \\ k\in\mathcal{K}}} \Delta_k = 0$. We also know that the trust-region radius only expands during successful iterations. So the statement of the Lemma is proven. ■

Lemma 7. *In Algorithm 2* $\liminf_{k\to\infty} \|\boldsymbol{g}_k\| = 0$.

Proof. If there is a constant $c_1 > 0$ such that $\|\boldsymbol{g}_k\| > c_1$ for all k, then Lemma 6 will be contradicted. ■

Lemma 8. *A subsequence* $\{k_j\}$ *with* $\lim_{j\to\infty}\left\|\boldsymbol{g}_{k_j}\right\| = 0$ *also has* $\lim_{j\to\infty}\left\|\nabla f\left(\boldsymbol{x}_{k_j}\right)\right\| = 0$.

Proof. First we know that for j large enough $\left\|\boldsymbol{g}_{k_j}\right\| < \varepsilon$ and therefore by Step 1 of Algorithm 2 and criticality step in Algorithm 3 m_{k_j} has sufficient quality on $\mathcal{B}\left(\boldsymbol{x}_{k_j};\Delta_{k_j}\right)$ with $\Delta_{k_j} \leq \mu_2\left\|\boldsymbol{g}_{k_j}\right\|$. Se we have

$$\begin{aligned}\left\|\nabla f\left(\boldsymbol{x}_{k_j}\right)\right\| &\leq \left\|\nabla f\left(\boldsymbol{x}_{k_j}\right) - \boldsymbol{g}_{k_j}\right\| + \left\|\boldsymbol{g}_{k_j}\right\| \\ &\leq \kappa_{eg}\Delta_{k_j} + \left\|\boldsymbol{g}_{k_j}\right\| \\ &\leq \left(\kappa_{eg}\mu_2 + 1\right)\left\|\boldsymbol{g}_{k_j}\right\|.\end{aligned}$$

■

Theorem 5. *In Algorithm 2* $\liminf_{k\to\infty}\|\nabla f\left(\boldsymbol{x}_k\right)\| = 0$.

Proof. This is the immediate result of Lemma 7 and Lemma 8. ■

Theorem 6. *Suppose that* f *has Lipschitz continuous gradients with Lipschitz constant* κ_{eg} *in an open domain containing* $\mathcal{S}'$*, defined in Definition 9. Then* $\lim_{k\to\infty}\|\nabla f\left(\boldsymbol{x}_k\right)\| = 0$.

Proof. When the successful iterations are finite the statement of the Theorem is proven in Lemma 5. Suppose now that there are infinite successful iterations all contained in the set $\mathcal{K}$. To establish a contrapositive suppose there exists a subsequence $\{k_j\}$ with $\left\|\nabla f\left(\boldsymbol{x}_{k_j}\right)\right\| \geq \varepsilon_0$ for some $\varepsilon_0 > 0$. Then Lemma 8 implies that $\left\|\boldsymbol{g}_{k_j}\right\| \geq \varepsilon_1$ choosing $\varepsilon_1 \leq \min\left\{\varepsilon, \frac{\varepsilon_0}{2\left(2+\kappa_{eg}\mu_2\right)}\right\}$ where ε is the input parameter in Algorithm 2 used in Step 1. To justify the choice of ε notice that

- either $\left\|\boldsymbol{g}_{k_j}\right\| \geq \varepsilon$,
- or m_{k_j} is of sufficient quality and $\left\|\boldsymbol{g}_{k_j}\right\|\mu_2 \geq \Delta_{k_j}$ in which case

$$\begin{aligned}\left\|\boldsymbol{g}_{k_j}\right\| &\geq \left\|\nabla f\left(\boldsymbol{x}_{k_j}\right)\right\| - \left\|\nabla f\left(\boldsymbol{x}_{k_j}\right) - \boldsymbol{g}_{k_j}\right\| \\ &\geq \varepsilon_0 - \kappa_{eg}\Delta_{k_j} \geq \varepsilon_0 - \kappa_{eg}\mu_2\left\|\boldsymbol{g}_{k_j}\right\| \\ \Rightarrow \left\|\boldsymbol{g}_{k_j}\right\| &\geq \frac{\varepsilon_0}{\left(1+\kappa_{eg}\mu_2\right)} \geq \frac{1}{2}\frac{\varepsilon_0}{\left(2+\kappa_{eg}\mu_2\right)}.\end{aligned}$$

Lemma 7 guarantees that there exists another subsequence $\{\ell_j\}$ such that $\|\boldsymbol{g}_k\| \geq \varepsilon_1$ for $k_j \leq k < \ell_j$ and $\|\boldsymbol{g}_k\| < \varepsilon_1$ for large enough j. Define $\mathcal{K}' = \bigcup_j \{k : \ k_j \leq k < \ell_j\}$. Lemma 6 shows that $\Delta_k \to 0$ as $k \to \infty$; therefore we conclude that by Lemma 3 large k are successful if the model has sufficient quality, and are model improving otherwise (see Figure 2.2).

Now using Lemma 6 again we observe that for large enough $k \in \mathcal{K} \cap \mathcal{K}'$, $\Delta_k < \frac{\varepsilon_1}{\kappa_{bhm}}$ and thus

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \eta_1 \frac{\kappa_{fcd}}{2} \varepsilon_1 \Delta_k.$$

Next we write for large enough j

$$\left\|\boldsymbol{x}_{k_j} - \boldsymbol{x}_{\ell_j}\right\| \leq \sum_{\substack{k=k_j \\ k \in \mathcal{K} \cap \mathcal{K}'}}^{\ell_j - 1} \Delta_k \leq \frac{2}{\eta_1 \kappa_{fcd} \varepsilon_1} \left[f\left(\boldsymbol{x}_{k_j}\right) - f\left(\boldsymbol{x}_{\ell_j}\right)\right]$$

implying that $\left\|\boldsymbol{x}_{k_j} - \boldsymbol{x}_{\ell_j}\right\| \to 0$ as $j \to \infty$ since the right hand side of the above must converge to zero due to the fact that f is lower bounded. Finally using Lipschitz continuity of the gradient and knowing that the criticality step ensures that m_{ℓ_j} has sufficient quality on $\mathcal{B}\left(\boldsymbol{x}_{\ell_j}; \mu_2 \left\|\boldsymbol{g}_{\ell_j}\right\|\right)$ we conclude that for large enough j

$$\begin{aligned}
\left\|\nabla f\left(\boldsymbol{x}_{k_j}\right)\right\| &\leq \left\|\nabla f\left(\boldsymbol{x}_{k_j}\right) - \nabla f\left(\boldsymbol{x}_{\ell_j}\right)\right\| + \left\|\nabla f\left(\boldsymbol{x}_{\ell_j}\right) - \boldsymbol{g}_{\ell_j}\right\| + \left\|\boldsymbol{g}_{\ell_j}\right\| \\
&\leq \nu_{gL} \left\|\boldsymbol{x}_{k_j} - \boldsymbol{x}_{\ell_j}\right\| + \kappa_{eg} \mu_2 \varepsilon_1 + \varepsilon_1 \\
&\leq (2 + \kappa_{eg} \mu_2) \varepsilon_1 \leq \frac{1}{2} \varepsilon_0.
\end{aligned}$$

But this contradicts the definition of $\{k_j\}$ falsifying its existence. This completes the proof.

■

3. ASTRO: ADAPTIVE SAMPLING TRUST-REGION OPTIMIZATION

The primary contribution of this dissertation is developing algorithmic frameworks for unconstrained continuous simulation optimization both in the presence and absence of unbiased (Monte Carlo) estimates of the gradient of the objective function. (Unbiased estimates of the objective function are always assumed to be available.) In this chapter, we present an algorithm for the former context, that is, unconstrained continuous simulation optimization where unbiased estimates of both the function and its gradient are assumed to be available through a Monte Carlo oracle.

The family of algorithms (ASTRO) we propose in this chapter follows logic that is analogous to corresponding DTRO algorithms where the direct gradient information is assumed to be available. ASTRO is an iterative algorithm where, during each iteration k, a local (and analytically convenient) model of the objective function is constructed within a trust-region around the current iterate $\boldsymbol{X}_k$, by obtaining Monte Carlo estimates of the objective function and gradient at $\boldsymbol{X}_k$. After such model construction, the constructed model is used within an optimization step to identify the next candidate solution $\tilde{\boldsymbol{X}}_{k+1}$. The candidate solution $\tilde{\boldsymbol{X}}_{k+1}$ is not immediately accepted. Instead, if the model-predicted and Monte Carlo-estimated function decrease values from the current iterate $\boldsymbol{X}_k$ to the candidate point $\tilde{\boldsymbol{X}}_{k+1}$ are comparable in a certain sense, then $\tilde{\boldsymbol{X}}_{k+1}$ is accepted as the next iterate $\boldsymbol{X}_{k+1}$ and the trust-region expanded; otherwise, $\tilde{\boldsymbol{X}}_{k+1}$ is rejected and the trust-region is shrunk and the estimators updated with new observations, in an attempt to improve the quality of the local model around $\boldsymbol{X}_k$. As we shall see, the number of Monte Carlo calls at each design point is *adaptive* — just enough to ensure that the sampling variability of function observations at the point is commensurate with the estimated model error.

The salient feature of ASTRO is its adaptability, sampling more when the incumbent solution is inferred to be close to a stationary point and less otherwise. While this ensures practical efficiency, the resulting asymptotic behavior and its analysis is becomes nontrivial. We show that ASTRO globally converges to a first-order critical point almost surely. We also provide evidence that it functions as intended in low to moderate dimensional SO problems.

In the remainder of this chapter, we restate the problem along with notation in sections 3.1 and 3.2. In Section 3.3 we summarize the related research in the area of stochastic TRO in presence of the sample path derivatives. The ASTRO algorithm is discussed in Section 3.4 and the theoretical convergence results are presented in Section 3.5. Finally Section 3.6 provides implementation remarks and results from numerical experiments.

3.1 Problem Statement and Notations

Recall the SO problem:

$$\begin{aligned} \text{Problem } P: \qquad \text{minimize} \quad & f(\boldsymbol{x}) := \mathbb{E}[F(\boldsymbol{x})] \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathbb{R}^d, \end{aligned}$$

where $f(\cdot)$ is known only through a Monte Carlo simulation capable of generating copies of the random variable $F(\boldsymbol{x})$ for each $\boldsymbol{x} \in \mathbb{R}^d$. The corresponding estimator is denoted as $f_n(\boldsymbol{x})$, where the label n represents some measure of simulation effort, and is used in place of $f(\boldsymbol{x})$. Typically, $f_n(\boldsymbol{x})$ is constructed as the sample mean of i.i.d observations.

We use the sample mean estimators for the function value and gradient. In what follows we give the notations used for these estimators.

- Let

$$\bar{F}(\boldsymbol{x}, n) = n^{-1} \sum_{i=1}^{n} F_i(\boldsymbol{x}) \tag{3.1}$$

be the Monte Carlo estimate of the unknown function value $f(\boldsymbol{x})$ at point $\boldsymbol{x} \in \mathbb{R}^d$, where $F_i(\boldsymbol{x})$ are i.i.d and n represents the replication size (number of observations obtained at $\boldsymbol{x}$). The estimate of the standard error of $\bar{F}(\boldsymbol{x},n)$ is $n^{-1/2}\hat{\sigma}_f(\boldsymbol{x},n)$ where

$$\hat{\sigma}_f(\boldsymbol{x},n) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(F_i(\boldsymbol{x}) - \bar{F}(\boldsymbol{x},n)\right)^2}. \tag{3.2}$$

- Let

$$\bar{\boldsymbol{G}}(\boldsymbol{x},n) = n^{-1}\sum_{i=1}^{n}\boldsymbol{G}_i(\boldsymbol{x}) \tag{3.3}$$

be the Monte Carlo estimate of the unknown function gradient $\boldsymbol{g}: \mathbb{R}^d \to \mathbb{R}^d$ at point $\boldsymbol{x}$, where $\boldsymbol{G}_i(\boldsymbol{x})$ are i.i.d. The estimate of the standard error of $\bar{\boldsymbol{G}}(\boldsymbol{x},n)$ is $n^{-1/2}\hat{\sigma}_g(\boldsymbol{x},n)$ where $\hat{\sigma}_g(\boldsymbol{x},n)$ is the element-wise square root of $\hat{\sigma}_g^2(\boldsymbol{x},n)$ which is a d-dimensional vector consisting of the diagonal values of the sample covariance matrix

$$\hat{\Sigma}_g = \frac{1}{n-1}\sum_{i=1}^{n}\left(\boldsymbol{G}_i(\boldsymbol{x}) - \bar{\boldsymbol{G}}(\boldsymbol{x},n)\right)\left(\boldsymbol{G}_i(\boldsymbol{x}) - \bar{\boldsymbol{G}}(\boldsymbol{x},n)\right)^T.$$

Particularly one can write

$$\hat{\sigma}_g^2(\boldsymbol{x},n) = \begin{bmatrix} \frac{1}{n-1}\sum_{i=1}^{n}\left(G_i^1(\boldsymbol{x}) - \bar{G}^1(\boldsymbol{x},n)\right)^2 \\ \frac{1}{n-1}\sum_{i=1}^{n}\left(G_i^2(\boldsymbol{x}) - \bar{G}^2(\boldsymbol{x},n)\right)^2 \\ \vdots \\ \frac{1}{n-1}\sum_{i=1}^{n}\left(G_i^d(\boldsymbol{x}) - \bar{G}^d(\boldsymbol{x},n)\right)^2 \end{bmatrix} \tag{3.4}$$

$G_i^j(\boldsymbol{x})$ being the j-th element of $\boldsymbol{G}_i(\boldsymbol{x})$ and $\bar{G}^j(\boldsymbol{x},n)$ being the j-th element of $\bar{\boldsymbol{G}}(\boldsymbol{x},n)$, is derived using the sample covariance matrix.

Note that from this chapter on we use upper case for the estimators, model, iterates and candidate points as they are no longer deterministic values but random variables.

3.2 Useful Results

The two following lemmas are frequently used in the majority of our theoretical analysis.

Lemma 9. *(Boole's Inequality) For a countable set of events $A_1, A_2, \cdots$, we have*

$$\mathbb{P}\left(\bigcup_i A_i\right) \leq \sum_i \mathbb{P}(A_i).$$

Specifically if $X \leq X_1 + X_2 + \cdots + X_q$ for some integer q and random variables X and $X_i,\ i = 1, 2, \cdots, q$, then

$$\begin{aligned}
(X > c) &\subseteq \left(X_1 + X_2 + \cdots X_q > c\right) \\
&\subseteq \left(X_1 > \frac{c}{q}\right) \cup \left(X_2 > \frac{c}{q}\right) \cup \cdots \cup \left(X_q > \frac{c}{q}\right); \\
\Rightarrow \mathbb{P}\{X > c\} &\leq \mathbb{P}\left\{\bigcup_{i=1}^{q} \left(X_i > \frac{c}{q}\right)\right\} \\
&\leq \sum_{i=1}^{q} \mathbb{P}\left\{X_i > \frac{c}{q}\right\}.
\end{aligned}$$

Lemma 10. *(Borel-Cantelli's First Lemma) For a countable set of events $A_1, A_2, \cdots$, if $\sum_{n=1}^{\infty} \mathbb{P}\{A_n\} < \infty$, then $\mathbb{P}\{A_n \text{ i.o.}\} = \mathbb{P}\{\limsup_{n\to\infty} A_n\} = 0$, where $\limsup_{n\to\infty} A_n = \lim_{m\to\infty} \bigcup_{n=m}^{\infty} A_n = \{\omega \in \Omega$ that are in infinitely many $A_n\}$.*

3.3 Related Work

A number of recently proposed algorithms are noteworthy in their relationship to what we investigate here. STRONG or Stochastic Trust-Region Response-Surface Method [49], for instance, is an adaptive sampling trust-region algorithm for solving SO problems that is in the spirit of what we propose here. Like ASTRO, STRONG adapts a trust-region framework where a local model of the objective function is constructed and updated through Monte Carlo sampling. A key feature of STRONG is local model construction through a design of experiments combined with a hypothesis testing procedure. STRONG assumes that the error in the derivative observations are additive and have a Gaussian distribution. Amos et. al. [50] and Bastin et. al. [51] treat a similar setting where unbiased observations of the gradient assumed to be available. (The former, in fact, assumes that unbiased estimates of the Hessian of the objective function are available.) Bastin et. al. [51] is specific to the problem of estimation within mixed-logit models. In the context of multi-objective

functions a relatively recent study by Kim and Ryu [52] adopts trust-region framework with SAA that uses fixed sample size at every point.

3.4 Algorithm Listing

Algorithm 4 lists the steps of ASTRO, in which the general operations during each iteration are encapsulated within four repeating stages that are modified versions of their DTRO counterpart: (i) local (stochastic) model construction through adaptive sampling; (ii) constrained optimization of the constructed model (within the trust-region) for identifying the next candidate solution; (iii) re-estimation of the objective function at the candidate solution through adaptive sampling and evaluation of the candidate solution; and (iv) (stochastic) sufficient decrease check by comparing predicted and estimated function decrease, and iterate and trust-region update. Note that the main two additions to ASTRO algorithm are the sampling rules prescribed in Step 2 and Step 5.

We now describe each step of Algorithm 4. Before model construction ASTRO obtains a Monte Carlo estimate of the function value and gradient at $\boldsymbol{X}_k$ in Step 2; the amount of sampling is lower bounded by some function of the trust-region radius and a pre-defined deterministic sequence λ_k. Th deterministic sequence λ_k is chosen as a multiple of $k^{3(1+\varepsilon)}$ for some small $\varepsilon > 0$, that inflates the sample size by some amount at every iteration. Such inflating component is common in the SO context, whose primary role is to ensure that the sampling errors diminish fast enough as the algorithm evolves through the search space. The sample size also adapts to the estimates of the function value and function gradient as well as the trust-region radius, that is sample until both the estimated function standard error and the L_∞ norm of the estimated gradient standard error fall below some threshold as a function of the trust-region size. The threshold for the estimated function standard error is the deflated square of the trust-region radius while the threshold for the L_∞ norm of the estimated gradient standard error is the deflated trust-region radius itself, as shown in (3.5). The deflation factor is $1/\sqrt{\lambda_k}$. The obtained function estimate and gradient estimate are then used to construct the model in Step 3, where $\hat{\mathscr{B}}_k$ is a symmetric matrix that

Algorithm 4 Adaptive Sampling Trust-Region Optimization (ASTRO) Algorithm

Require: *Parameters from DTRO*: Initial point $\boldsymbol{X}_0$, model "fitness" parameters $0 < \eta_3 < \eta_2 < 0.5 < \eta_1$, trust-region radius increasing factor $\gamma_2 > 1$ and decreasing factor $0 < \gamma_1 < 1$, the initial trust-region radius $\Delta_0 > 0$, and the maximum radius $\Delta_{\max} > 0$; *ASTRO-specific parameters*: $\gamma \in (.4, 1)$, $k^{3(1+\varepsilon)} = O(\lambda_k)$, and adaptive sampling constants for the function and gradient $\kappa_f, \kappa_g \geq 1$.

1: **for** $k = 0, 1, 2, \ldots$ **do**

Construct, optimize, and evaluate the model in the trust-region:

2: **Sample** to obtain objective function and gradient estimators $\bar{F}(\boldsymbol{X}_k, \widetilde{N}_k)$ and $\bar{\boldsymbol{G}}(\boldsymbol{X}_k, \widetilde{N}_k)$, respectively, where

$$\tilde{N}_k = \inf\left\{ n \geq \max\left\{ 2, \left(\frac{\lambda_k}{\min\left(\Delta_k^2, \Delta_k^4\right)} \right)^{\gamma} \right\} : \; n \geq \max\left\{ \frac{\hat{\sigma}_f^2(\boldsymbol{X}_k, n)}{\kappa_f^2 \Delta_k^4}, \frac{\left\| \hat{\sigma}_g^2(\boldsymbol{X}_k, n) \right\|_\infty}{\kappa_g^2 \Delta_k^2} \right\} \lambda_k \right\}, \tag{3.5}$$

and $\|\hat{\sigma}_g^2(\boldsymbol{X}_k, n)\|_\infty$ is the maximum diagonal value of the sample covariance matrix.

3: Construct the quadratic model $M_k(\boldsymbol{p}) = \bar{F}(\boldsymbol{X}_k, \widetilde{N}_k) + \bar{\boldsymbol{G}}^T(\boldsymbol{X}_k, \widetilde{N}_k)\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T \hat{\mathscr{B}}_k \boldsymbol{p}$.

4: Obtain the kth step as $\boldsymbol{P}_k = \operatorname{argmin}_{\|\boldsymbol{p}\| \leq \Delta_k} M_k(\boldsymbol{p})$, and set the candidate point $\tilde{\boldsymbol{X}}_{k+1} = \boldsymbol{X}_k + \boldsymbol{P}_k$.

5: **Sample** at $\tilde{\boldsymbol{X}}_{k+1}$ to obtain $\bar{F}(\tilde{\boldsymbol{X}}_{k+1}, \widetilde{N}_{k+1})$, where

$$\tilde{N}_{k+1} = \inf\left\{ n \geq \max\left\{ 2, \left(\frac{\lambda_k}{\Delta_k^4} \right)^{\gamma} \right\} : \; \frac{\hat{\sigma}_f\left(\tilde{\boldsymbol{X}}_{k+1}, n\right)}{\sqrt{n}} \leq \frac{\kappa_f \Delta_k^2}{\sqrt{\lambda_k}} \right\}. \tag{3.6}$$

6: Evaluate the success ratio

$$\hat{\rho}_k = \frac{\bar{F}(\boldsymbol{X}_k, \widetilde{N}_k) - \bar{F}(\boldsymbol{X}_{k+1}, \widetilde{N}_{k+1})}{M_k(\boldsymbol{0}) - M_k(\boldsymbol{P}_k)}. \quad \begin{array}{l} \leftarrow \text{estimated actual objective value "reduction;" could be positive or negative} \\ \leftarrow \text{estimated objective value reduction predicted by quadratic model; always positive.} \end{array}$$

Adjust the trust-region radius Δ_k based on the estimated success ratio $\hat{\rho}_k$:

7: **if** $\hat{\rho}_k < \eta_2 < 0.5$, **then** $\Delta_{k+1} = \gamma_1 \Delta_k$. {Using the same criteria as DTRO.}

8: **else if** $\hat{\rho}_k > \eta_1 \geq 0.5$, $\|\boldsymbol{P}_k\| = \Delta_k$, **then** $\Delta_{k+1} = \min\{\gamma_2 \Delta_k, \Delta_{\max}\}$.

9: **else**, $\Delta_{k+1} = \Delta_k$.

10: **end if**

Determine whether to accept the step specified by optimizing the model in the trust-region:

11: **if** $\hat{\rho}_k > \eta_3$, **then** $\boldsymbol{X}_{k+1} = \tilde{\boldsymbol{X}}_{k+1}$, and $N_{k+1} = \tilde{N}_{k+1}$. {Using the same criteria as DTRO.}

12: **else** $\boldsymbol{X}_{k+1} = \boldsymbol{X}_k$, and $N_{k+1} = \tilde{N}_k$.

13: **end if**

14: **end for**

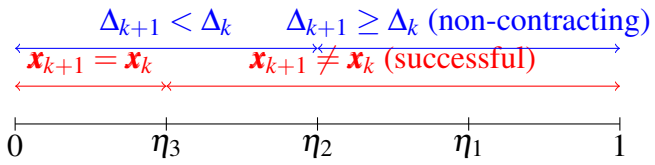


Figure 3.1. Illustration of the $\hat{\rho}_k$ possibilities. If $\eta_2 = \eta_3$ then the successful and non-contracting iterations overlap.

approximates the Hessian. In Step 4, that is also known as the sub-problem step, a constrained minimization of the constructed model within the trust-region $\mathcal{B}(\boldsymbol{X}_k; \Delta_k)$ yields the new candidate point $\tilde{\boldsymbol{X}}_{k+1}$. Again, the estimated function value at the candidate point is obtained following the sampling rule specified in Step 5. As (3.6) stipulates enough sampling is performed so that the estimated function standard error falls below the threshold discussed above. Note that the restriction on the ℓ_∞ norm of the estimated gradient standard error is removed at this Step. The success of the model is evaluated in Step 6 by the estimated success ratio $\hat{\rho}_k$, that compares the reduction in the estimated function values with the reduction predicted by the model. Depending on the value of this ratio the iterate and trust-region management for the next iteration is performed. The adjustments are made to the trust-region radius in Steps 7 – 10 based on the estimated success ratio and the norm of the new step calculated in the sub-problem. In Steps 11 – 13 the candidate point $\tilde{\boldsymbol{X}}_{k+1}$ is either accepted as the new iterate if $\hat{\rho}_k$ exceeds the specified threshold, or rejected otherwise.

Note that an iteration might be successful, meaning the new candidate point accepted, but the trust-region radius contracting as shown in Figure 3.1. This is because the trust-region update has a more stringent requirement, that is sufficiently large value of the ratio $\hat{\rho}_k$ are required to allow for larger step size in the next iteration, while accepting a candidate point as the new iterate can occur more easily and for lower values of the ratio $\hat{\rho}_k$. This distinction is useful in the implementation but makes the analysis slightly more complicated.

In the following sections we will describe the necessary assumptions for the consistency of ASTRO in some probabilistic sense. We present the crucial theorems that justify the

sampling rule used in ASTRO and provide the convergence theorems corresponding to those in the deterministic counterpart of the algorithm (see Theorems 5 and 6). Lastly we implement ASTRO on a suite of problems to gauge its finite-time performance.

3.5 Convergence Results

Throughout the optimization we make the following assumptions:

Assumption 3. *(Simulation Error) The Monte Carlo oracle, when executed at $\boldsymbol{X}_k$, generates independent and identically distributed (i.i.d) random variates*

$$F_j(\boldsymbol{X}_k) = f(\boldsymbol{X}_k) + \xi_j | \mathcal{F}_k,$$

where $\xi_1, \xi_2, \cdots$ is a martingale-difference sequence adapted to $\mathcal{F}_k$ such that $\mathbb{E}\left[\xi_j^2 \mid \mathcal{F}_k\right] = \sigma^2 < \infty$ and $\sup_k \mathbb{E}\left[\left|\xi_j\right|^{4v} \mid \mathcal{F}_k\right] < \infty$ for some $v \geq 2$. Similarly the Monte Carlo oracle generates i.i.d observations

$$\boldsymbol{G}_j(\boldsymbol{X}_k) = \boldsymbol{g}(\boldsymbol{X}_k) + \left[\zeta_j^1 | \mathcal{F}_k, \zeta_j^2 | \mathcal{F}_k, \cdots, \zeta_j^d | \mathcal{F}_k\right]^T,$$

where ζ_j^i are martingale-difference sequences adapted to $\mathcal{F}_k$ such that $\mathbb{E}\left[\left(\zeta_j^i\right)^2 \mid \mathcal{F}_k\right] = \sigma^2$ and $\sup_k \mathbb{E}\left[\left|\zeta_j^i\right|^{4v} \mid \mathcal{F}_k\right] < \infty$.

Assumption 4. *(Cauchy Reduction) In Step 3, the solution to the model step problem for some $c_1 \in (0,1)$ satisfies*

$$M_k(\mathbf{0}) - M_k(\boldsymbol{S}_k) \geq c_1 \left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right)\right\| \min\left\{\Delta_k, \frac{\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right)\right\|}{\left\|\hat{\mathscr{B}}_k\right\|}\right\}. \tag{3.7}$$

Assumption 5. *($\hat{\mathscr{B}}_k$ uniform bound) There exists $\kappa_{bhm} > 0$ such that $\mathbb{P}\left\{\left\|\hat{\mathscr{B}}_k\right\| \leq \kappa_{bhm}\right\} = 1$ for all k.*

We first include the main results that are adopted from Lemma 2 and Theorem 1 in [38] that we explicitly use in our analysis.

Theorem 7. *Suppose $X_i, i = 1,2,\ldots$ are iid random variables with $\mathbb{E}[X_1] = 0, \mathbb{E}[X_1^2] = \sigma^2 > 0$, and $\mathbb{E}[|X_1|^{4v}] < \infty$ for some $v \geq 2$. Let $\hat{\sigma}_n^2 = n^{-1}\sum_{i=1}^n (X_i - \bar{X}_n)^2$, where $\bar{X}_n = n^{-1}\sum_{i=1}^n X_i$ and $\kappa > 0$ be some constant. If*

$$N = \inf\left\{ n \geq \left(\frac{\lambda}{\kappa^2}\right)^{\gamma} : \frac{\hat{\sigma}_n}{\sqrt{n}} \leq \frac{\kappa}{\sqrt{\lambda}} \right\}, \gamma \in \left(\frac{2}{(v+3)}, 1\right)$$

then the following hold.

(i) $\mathbb{P}\{N < \infty\} = 1$ *and as* $\lambda \to \infty$, $N \xrightarrow{wp1} \infty$.

(ii) As $\lambda \to \infty$,

$$\frac{N\kappa^2}{\sigma^2\lambda} \xrightarrow{wp1} 1.$$

(iii) As $\lambda \to \infty$ *and for every* $m < v$,

$$\mathbb{E}[N^m] \sim \left(\sigma^2\lambda\kappa^{-2}\right)^m.$$

(iv) For every $\varepsilon \in (0,1)$,

$$\mathbb{P}\{N \leq \lambda\kappa^{-2}\sigma^2(1-\varepsilon)\} = \mathcal{O}\left(\lambda^{\gamma(1-v)}\right).$$

(v) As $\lambda \to \infty$,

$$\mathbb{E}[\bar{X}_N^2] \sim \kappa^2\lambda^{-1}.$$

Proof. Let us define $b = \lambda\kappa^{-2}$ and $n^* = b\sigma^2$. Recall that

$$\hat{\sigma}_n^2 = n^{-1}\sum_{i=1}^n (\boldsymbol{X}_i - \bar{\boldsymbol{X}})^2 = n^{-1}\sum_{i=1}^n \boldsymbol{X}_i^2 - (n^{-1}\sum_{i=1}^n \boldsymbol{X}_i)^2 \xrightarrow{wp1} \sigma^2,$$

since $(n^{-1}\sum_{i=1}^n \boldsymbol{X}_i)^2 \xrightarrow{wp1} \mu^2$ by continuous mapping property and $n^{-1}\sum_{i=1}^n \boldsymbol{X}_i^2 \xrightarrow{wp1} \sigma^2 + \mu^2$ since $\boldsymbol{X}_i^2$ are i.i.d with mean $\sigma^2 + \mu^2$. Then proof of (i) follows since

$$\mathbb{P}\{N = \infty\} = \mathbb{P}\left\{\hat{\sigma}_n^2 \geq \frac{n}{b},\ \forall n\right\} = 0,$$

and further observing that $b \to \infty$ and hence $N \xrightarrow{wp1} \infty$ as $\lambda \to \infty$.

In (ii) we can write

$$b\hat{\sigma}_N^2 \leq N \leq 1 + b\hat{\sigma}_{N-1}^2$$

and dividing by n^* we will get

$$\frac{\hat{\sigma}_N^2}{\sigma^2} \leq \frac{N}{n^*} \leq \frac{1}{\sigma^2 b} + \frac{\hat{\sigma}_{N-1}^2}{\sigma^2}.$$

Since $\hat{\sigma}_N^2 \xrightarrow{wp1} \sigma^2$ and $\hat{\sigma}_{N-1}^2 \xrightarrow{wp1} \sigma^2$ as $\lambda \to \infty$, we observe that $N/n^* \xrightarrow{wp1} 1$.

In (iii) we want to prove that

$$\lim_{\lambda \to \infty} \mathbb{E}\left[\left(\frac{N}{n^*}\right)^m\right] = 1.$$

First we choose λ large enough such that $b^\gamma \leq u$ where $u = [n^*(1+\varepsilon)]$. Then we can write

$$\begin{aligned}\mathbb{E}\left[\left(\frac{N}{n^*}\right)^m\right] &\leq (1+\varepsilon)^m \mathbb{P}(N \leq u) + \sum_{n=u+1}^{\infty} \left(\frac{n}{n^*}\right)^m \mathbb{P}(N=n) \\ &\leq (1+\varepsilon)^m \mathbb{P}(N \leq u) + \left(\frac{1}{n^*}\right)^m \sum_{n=u+1}^{\infty} n^m \mathbb{P}(N=n)\end{aligned}$$

and further we can write

$$\begin{aligned}\sum_{n=u+1}^{\infty} n^m \mathbb{P}(N=n) &= (u+1)^m \mathbb{P}(N=u+1) + (u+2)^m \mathbb{P}(N=u+2) + \cdots \\ &= (u+1)^m \mathbb{P}(N>u) + \sum_{n=u+1}^{\infty} \left((n+1)^m - n^m\right) \sum_{i=n+1}^{\infty} \mathbb{P}(N=i) \\ &\leq (u+1)^m \mathbb{P}(N>u) + \sum_{n=u+1}^{\infty} 2^m n^{m-1} \mathbb{P}(N>n),\end{aligned}$$

where using binomial expansion we get $(n+1)^m - n^m = \sum_{k=1}^m \binom{m}{k} n^{m-k} \leq n^{m-1}(1+1)^m$. So we have that

$$\begin{aligned}\mathbb{E}\left[\left(\frac{N}{n^*}\right)^m\right] &\leq (1+\varepsilon)^m \mathbb{P}(N \leq u) \\ &\quad + \left(\frac{1}{n^*}\right)^m \left((u+1)^m \mathbb{P}(N>u) + 2^m \sum_{n=u+1}^{\infty} n^{m-1} \mathbb{P}(N>n)\right).\end{aligned} \tag{3.8}$$

We use $\mathbb{E}\left[\left(\hat{\sigma}_n^2 - \sigma^2\right)^{2v}\right] \leq K n^{-v}$ from [53,54], where K depends on v and moments of X_i's to say that for $n \geq u+1$

$$\begin{aligned}\mathbb{P}(N>n) &\leq \mathbb{P}\left(\frac{\hat{\sigma}_n^2}{n} > \frac{\kappa^2}{\lambda}\right) = \mathbb{P}\left(\hat{\sigma}_n^2 > \frac{n}{n^*}\sigma^2\right) \\ &\leq \mathbb{P}\left(\hat{\sigma}_n^2 > (1+\varepsilon)\sigma^2\right) = \mathbb{P}\left(\hat{\sigma}_n^2 - \sigma^2 > \varepsilon\sigma^2\right) \\ &\leq \mathbb{E}\left[\left(\hat{\sigma}_n^2 - \sigma^2\right)^{2v}\right] \left(\varepsilon\sigma^2\right)^{2v} \leq K' n^{-v},\end{aligned}$$

where $K' = K\left(\varepsilon\sigma^2\right)^{2v}$. Hence

$$\sum_{n=u+1}^{\infty} n^{m-1}\mathbb{P}\left(N > n\right) \leq K' \sum_{n=u+1}^{\infty} n^{m-1-v} \leq K' u^{m-v}$$

since $m < v$ and therefore the second term of the right hand side in (3.8) tends to zero as $\lambda \to 0$ since $K' u^{m-v}\left(2/n^*\right)^m \leq 2^m K'\left(n^*\right)^v \to 0$ as $\lambda \to 0$ and $\mathbb{P}\left(N > u\right) \to 0$ as $u \to \infty$ since from (i) $\mathbb{P}\left(N < \infty\right) = 1$. As a result the second term on (3.8) tends to zero as $\lambda \to 0$ and therefore for an arbitrary $\varepsilon, \tilde{\varepsilon} > 0$ we can write

$$\limsup_{\lambda\to\infty} \mathbb{E}\left[\left(\frac{N}{n^*}\right)^m\right] \leq (1+\varepsilon)^m + \tilde{\varepsilon}$$

and hence

$$\limsup_{\lambda\to\infty} \mathbb{E}\left[\left(\frac{N}{n^*}\right)^m\right] \leq 1.$$

We also use Fatou's Lemma to achieve $\liminf_{\lambda\to\infty} \mathbb{E}\left[\left(\frac{N}{n^*}\right)^m\right] \geq \mathbb{E}\left[\liminf_{\lambda\to\infty}\left(\frac{N}{n^*}\right)^m\right] = 1$ which completes the proof of (iii).

In (iv) we use our rule $N \geq b^\gamma$. Let $w = \left[n^*\left(1-\varepsilon\right)\right]$ and $h = \left[b^\gamma\right] + 1$ with λ chosen large enough such that $h \leq w$. Then

$$\begin{aligned}
\mathbb{P}\left(N \leq n^*\left(1-\varepsilon\right)\right) &= \sum_{n=h}^{w} \mathbb{P}\left(N = n\right) = \sum_{n=h}^{w} \mathbb{P}\left(\frac{\hat{\sigma}_n^2}{n} \leq \frac{1}{b}\right) \\
&= \sum_{n=h}^{w} \mathbb{P}\left(\hat{\sigma}_n^2 \leq \frac{n}{n^*}\sigma^2\right) \leq \sum_{n=h}^{w} \mathbb{P}\left(\hat{\sigma}_n^2 \leq \left(1-\varepsilon\right)\sigma^2\right) \\
&= \sum_{n=h}^{w} \mathbb{P}\left(\hat{\sigma}_n^2 - \sigma^2 \leq -\varepsilon\sigma^2\right) = \sum_{n=h}^{w} \mathbb{P}\left(\sigma^2 - \hat{\sigma}_n^2 \geq \varepsilon\sigma^2\right) \\
&\leq \sum_{n=h}^{w} \mathbb{E}\left[\left(\sigma^2 - \hat{\sigma}_n^2\right)^{2v}\right]\left(\varepsilon\sigma^2\right)^{2v} \leq K' \sum_{n=h}^{w} n^{-v} \\
&\leq K' \sum_{n=h}^{w} \left(b^\gamma\right)^{-v} \leq K' b^{\gamma(1-v)}.
\end{aligned}$$

In (v) we want to show that $\lim_{\lambda\to\infty} \mathbb{E}\left[\bar{X}_N^2 b\right] = 1$. We first write the following

$$\begin{aligned}
\mathbb{E}\left[\bar{X}_N^2 b\right] &= b\mathbb{E}\left[N^{-2}\left(\sum_{i=1}^{N} X_i\right)^2\right] \\
&= \frac{n^*}{\sigma^2}\left(\mathbb{E}\left[\frac{\left(\sum_{i=1}^{N} X_i\right)^2}{\left(n^*\right)^2}\right] + \mathbb{E}\left[\left(\sum_{i=1}^{N} X_i\right)^2\left(\frac{1}{N^2} - \frac{1}{\left(n^*\right)^2}\right)\right]\right).
\end{aligned}$$

From (iii) we observe that $\mathbb{E}[N] < \infty$. So we can use Wald's second Lemma (Theorem 2 in [55]):

$$\mathbb{E}\left[\left(\sum_{i=1}^{N} X_i\right)^2\right] = \sigma^2 \mathbb{E}[N].$$

It follows that

$$\frac{n^*}{\sigma^2}\mathbb{E}\left[\frac{\left(\sum_{i=1}^{N} X_i\right)^2}{(n^*)^2}\right] = \frac{\mathbb{E}[N]}{n^*} \to 1$$

as $\lambda \to \infty$. Therefore it suffices to show that

$$n^*\mathbb{E}\left[\left(\sum_{i=1}^{N} X_i\right)^2 \left(\frac{1}{N^2} - \frac{1}{(n^*)^2}\right)\right] = \mathbb{E}\left[\frac{\left(\sum_{i=1}^{N} X_i\right)^2}{n^*}\left(\left(\frac{n^*}{N}\right)^2 - 1\right)\right] \to 0.$$

We show this in the following three cases given any $\varepsilon \in (0,1)$ and using the fact that $(n^*)^{-1}\left(\sum_{i=1}^{N} X_i\right)^2$ is uniformly integrable in λ since $\mathbb{E}\left[\left|(n^*)^{-1}\left(\sum_{i=1}^{N} X_i\right)^2\right|\right] < \infty$ or in other words $(n^*)^{-1}\left(\sum_{i=1}^{N} X_i\right)^2 \in L^1$:

CASE 1 $|N - n^*| \leq \varepsilon n^*$: Here $(1-\varepsilon)n^* \leq N \leq (1+\varepsilon)n^*$ and therefore

$$\begin{aligned}\left|\left(\frac{n^*}{N}\right)^2 - 1\right| \mathbb{I}_{|N-n^*|\leq \varepsilon n^*} &= \frac{(n^*-N)(n^*+N)}{N^2}\mathbb{I}_{|N-n^*|\leq \varepsilon n^*} \\ &\leq \frac{\varepsilon(2+\varepsilon)}{(1-\varepsilon)^2}.\end{aligned}$$

This along with uniform integrability of $(n^*)^{-1}\left(\sum_{i=1}^{N} X_i\right)^2$ in λ implies that

$$\mathbb{E}\left[(n^*)^{-1}\left(\sum_{i=1}^{N} X_i\right)^2 \left|\left(\frac{n^*}{N}\right)^2 - 1\right| \mathbb{I}_{|N-n^*|\leq \varepsilon n^*}\right] \leq \frac{\varepsilon(2+\varepsilon)}{(1-\varepsilon)^2}\mathbb{E}\left[(n^*)^{-1}\left(\sum_{i=1}^{N} X_i\right)^2\right] \to 0$$

as $\lambda \to \infty$.

CASE 2 $N - n^* > \varepsilon n^*$: We observe that

$$\left|\left(\frac{n^*}{N}\right)^2 - 1\right| \mathbb{I}_{N-n^*>\varepsilon n^*} \leq \mathbb{I}_{N-n^*>\varepsilon n^*},$$

and $\mathbb{E}[\mathbb{I}_{N-n^*>\varepsilon n^*}] = \mathbb{P}(N-n^* > \varepsilon n^*) \to 0$ as $\lambda \to 0$ from (ii). Moreover, using $\mathbb{E}\left[\left(\sum_{i=1}^N X_i\right)^4\right] < \infty$ by Theorem 3 of [55] and Cauchy-Schwarz inequality we get

$$\mathbb{E}\left[(n^*)^{-1}\left(\sum_{i=1}^N X_i\right)^2 \left|\left(\frac{n^*}{N}\right)^2 - 1\right| \mathbb{I}_{N-n^*>\varepsilon n^*}\right]$$

$$\leq \mathbb{E}\left[(n^*)^{-1}\left(\sum_{i=1}^N X_i\right)^2 \mathbb{I}_{N-n^*>\varepsilon n^*}\right] \leq \sqrt{\mathbb{E}\left[(n^*)^{-2}\left(\sum_{i=1}^N X_i\right)^4\right]\mathbb{E}[\mathbb{I}_{N-n^*>\varepsilon n^*}]} \to 0.$$

as $\lambda \to \infty$.

CASE 3 $N-n^* < -\varepsilon n^*$: First knowing that $N \geq b^\gamma = \left(\sigma^{-2}n^*\right)^\gamma$ we have

$$\frac{1}{n^*}\left|\left(\frac{n^*}{N}\right)^2 - 1\right| = \left|\frac{n^*}{N^2} - \frac{1}{n^*}\right| \leq \sigma^{4\gamma}(n^*)^{1-2\gamma}.$$

Now we use Cauchy-Schwarz to write the following:

$$\mathbb{E}\left[\left(\sum_{i=1}^N X_i\right)^2 \mathbb{I}_{N-n^*<-\varepsilon n^*}\right] \leq \mathbb{E}\left[\left(\sum_{i=1}^{N_r} X_i\right)^2 \mathbb{I}_{N\leq r}\right]$$

$$\leq \sqrt{\mathbb{E}\left[\left(\sum_{i=1}^{N_r} X_i\right)^4\right]\mathbb{P}(N \leq r)}$$

where $r = [n^*(1-\varepsilon]$ and $N_r = \min\{N, r\}$. Letting $\mu_3 = \mathbb{E}\left[X_1^3\right]$ and $\mu_4 = \mathbb{E}\left[X_1^4\right]$ and the fact that $\mathbb{E}\left[N_r^2\right] \leq \mathbb{E}\left[r^2\right] < \infty$,by Theorem 7 of [55] we get

$$\mathbb{E}\left[\left(\sum_{i=1}^{N_r} X_i\right)^4\right] \leq 6\sigma^2\mathbb{E}\left[N_r\left(\sum_{i=1}^{N_r} X_i\right)^2\right] + 4\mu_3\mathbb{E}\left[N_r\sum_{i=1}^{N_r} X_i\right] + \mu_4\mathbb{E}[N_r]$$

$$\leq 6\sigma^2 r\sum_{i=1}^{N_r}\mathbb{E}\left[X_i^2\right] + 4\mu_3\sqrt{\mathbb{E}\left[\left(\sum_{i=1}^{N_r} X_i\right)^2\right]} + \mu_4 r$$

$$\leq 6\sigma^4 r^2 + 4\mu_3^{3/2}\sigma + \mu_4 r.$$

Therefore we conclude from (iv) that

$$\mathbb{E}\left[\frac{\left(\sum_{i=1}^{N} X_i\right)^2}{n^*}\left|\left(\frac{n^*}{N}\right)^2 - 1\right| \mathbb{I}_{N-n^*<-\varepsilon n^*}\right] \leq \frac{\sigma^{4\gamma}\sqrt{\left(6\sigma^4 + 4\mu_3\sigma + \mu_4\right)K' b^{\gamma(1-v)}}}{\left(n^*\right)^{2\gamma-1}} \tag{3.9}$$

$$\leq K''\left(n^*\right)^{2-2\gamma+\frac{\gamma(1-v)}{2}}. \tag{3.10}$$

where $K'' = \sqrt{K'\left(\left(6\sigma^4 + 4\mu_3\sigma + \mu_4\right)\right)}\sigma^{\gamma(3+v)}$. Since $v > 2$ and $\gamma \in (2/(3+v), 1)$, the right hand side of (3.9) tends to zero as $\lambda \to \infty$ that completes the proof.

■

We now show a similar result for multivariate random variables:

Corollary 1. *If $\boldsymbol{X}_i$ are i.i.d observations on the probability space $\left(\Omega, \mathbb{R}^d, \mathbb{P}\right)$ with $\mathbb{E}[\boldsymbol{X}_i] = \mu$, μ being a d-dimensional zero vector, and covariance matrix Σ the diagonal values of which being σ^2, then*

$$N = \inf\left\{n \geq \max\left\{2, \left(\frac{\kappa^2}{\lambda}\right)^{\gamma}\right\}, \gamma < 1 : \frac{\left\|\hat{\sigma}_n^2\right\|_{\infty}}{n} \leq \frac{\kappa^2}{\lambda}\right\}, \tag{3.11}$$

where $\hat{\sigma}_n^2$ are the diagonal elements of the sample covariance matrix, implies that

$$\mathbb{E}\left[\left\|\bar{\boldsymbol{X}}_n\right\|^2\right] \sim d\kappa^2\lambda^{-1}$$

as $\lambda \to \infty$.

Proof. Let $\bar{X}_n^i$ be the i-th element of the $\bar{\boldsymbol{X}}_n$ vector. We observe that

$$\begin{aligned}\mathbb{E}\left[\left\|\bar{\boldsymbol{X}}_n\right\|_2^2\right] &= \mathbb{E}\left[\left(\bar{X}_n^1\right)^2 + \left(\bar{X}_n^2\right)^2 + \cdots + \left(\bar{X}_n^d\right)^2\right] \\ &= d\mathbb{E}\left[\left(\bar{X}_n^1\right)^2\right] \sim d\kappa^2\lambda^{-1}\end{aligned}$$

where the last step follows from the sampling rule in (3.11) since for every $\bar{X}_n^i$ the conditions of the Theorem 7 hold. ■

Using the above results we show that the simulation errors associated with the function value and function gradient are bounded almost surely.

Lemma 11. *Let Assumption 3 hold and $\{\boldsymbol{X}_k\}$ be a sequence generated by Algorithm 4. Then for any $c_f > 0$ and $c_g > 0$ the following hold:*

(a) $\mathbb{P}\left\{|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| > c_f \Delta_k^{\alpha} \text{ i.o.}\right\} = 0$ *for* $\alpha = 0, 1, 2$.

(b) $\mathbb{P}\left\{\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \boldsymbol{g}(\boldsymbol{X}_k)\right\| > c_g \Delta_k^{\alpha} \text{ i.o.}\right\} = 0$ *for* $\alpha = 0, 1$.

Proof of (a). We note that the sampling rule in (3.5) ensures that $\tilde{N}_k$ is larger of the two values

$$\tilde{N}_k^1 = \inf\left\{n \geq \left(\frac{\lambda_k}{\Delta_k^4}\right)^{\gamma} : \frac{\hat{\sigma}_f(\boldsymbol{X}_k, n)}{\sqrt{n}} \leq \frac{\kappa_f \Delta_k^2}{\sqrt{\lambda_k}}\right\}$$
$$\tilde{N}_k^2 = \inf\left\{n \geq \left(\frac{\lambda_k}{\Delta_k^2}\right)^{\gamma} : \frac{\left\|\hat{\sigma}_g(\boldsymbol{X}_k, n)\right\|_{\infty}}{\sqrt{n}} \leq \frac{\kappa_g \Delta_k}{\sqrt{\lambda_k}}\right\}.$$

Using Chebyshev's inequality we can write

$$\begin{aligned}\mathbb{P}\left\{|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| > c_f\right\} &= \mathbb{E}\left[\mathbb{P}\left\{|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| > c_f \mid \mathcal{F}_k\right\}\right] \\ &\leq \mathbb{E}\left[c_f^{-2}\mathbb{E}\left[|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)|^2 \mid \mathcal{F}_k\right]\right] \\ &= \mathbb{E}\left[c_f^{-2}\mathbb{E}\left[\left(\frac{1}{N_k}\sum_{j=1}^{N_k}\xi_j\right)^2 \mid \mathcal{F}_k\right]\right].\end{aligned}$$

From part (v) in Theorem 7 we know that $\mathbb{E}\left[\left(\frac{1}{N_k}\sum_{j=1}^{N_k}\xi_j\right)^2 \mid \mathcal{F}_k\right] \sim \kappa_f^2 \Delta_k^4 \lambda_k^{-1}$. Knowing that $\Delta_k \leq \Delta_{\max}$, for a sufficiently large k and some $\delta > 0$ we can then write

$$\mathbb{E}\left[\left(\frac{1}{N_k}\sum_{j=1}^{N_k}\xi_j\right)^2 \mid \mathcal{F}_k\right] \leq (1+\delta)\,\kappa_f^2 \Delta_{\max}^4 \lambda_k^{-1},$$

and since $k^{3(1+\varepsilon)} = \mathcal{O}(\lambda_k)$, it follows that

$$\mathbb{P}\left\{|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| > c_f\right\} \leq c_f^{-2}(1+\delta)\,\kappa_f^2 \Delta_{\max}^4 \lambda_k^{-1} \tag{3.12}$$

is summable. So we can apply Borel-Cantelli's first Lemma (Lemma 10) to arrive at

$$\mathbb{P}\left\{|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| > c_f \text{ i.o.}\right\} = 0.$$

Next, we note that

$$\begin{aligned}\mathbb{P}\left\{|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| > c_f \Delta_k\right\} &\leq \mathbb{E}\left[c_f^{-2}\Delta_k^{-2}\mathbb{E}\left[|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)|^2 \mid \mathcal{F}_k\right]\right] \\ &\leq c_f^{-2}(1+\delta)\,\kappa_f^2 \Delta_{\max}^2 \lambda_k^{-1},\end{aligned}$$

and

$$\mathbb{P}\left\{\left|\bar{F}\left(\boldsymbol{X}_k,N_k\right)-f\left(\boldsymbol{X}_k\right)\right|>c_f\Delta_k^2\right\}\leq\mathbb{E}\left[c_f^{-2}\Delta_k^{-4}\mathbb{E}\left[\left|\bar{F}\left(\boldsymbol{X}_k,N_k\right)-f\left(\boldsymbol{X}_k\right)\right|^2\mid\mathcal{F}_k\right]\right]$$
$$\leq c_f^{-2}\left(1+\delta\right)\kappa_f^2\lambda_k^{-1}$$

for large enough k. Following the same application of Borel-Cantelli's Lemma (Lemma 10) the statement of (a) is proven. ■

Proof of (b). Similar to part (a) and using Corollary 1 we observe that for large k

$$\mathbb{P}\left\{\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k,\tilde{N}_k\right)-\boldsymbol{g}\left(\boldsymbol{X}_k\right)\right\|>c_g\right\}\leq\mathbb{E}\left[c_g^{-2}\mathbb{E}\left[\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k,\tilde{N}_k\right)-\boldsymbol{g}\left(\boldsymbol{X}_k\right)\right\|^2\mid\mathcal{F}_k\right]\right]$$
$$\leq c_g^{-2}\left(1+\delta\right)d\kappa_g^2\Delta_{\max}^2\lambda_k^{-1},$$

and

$$\mathbb{P}\left\{\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k,\tilde{N}_k\right)-\boldsymbol{g}\left(\boldsymbol{X}_k\right)\right\|>c_g\Delta_k\right\}\leq\mathbb{E}\left[c_g^{-2}\Delta_k^{-2}\mathbb{E}\left[\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k,\tilde{N}_k\right)-\boldsymbol{g}\left(\boldsymbol{X}_k\right)\right\|^2\mid\mathcal{F}_k\right]\right]$$
$$\leq c_g^{-2}\left(1+\delta\right)d\kappa_g^2\lambda_k^{-1}.$$

Both of these bounds on the right hand side are summable, leading to application of Borel-Cantelli to obtain the statement of part (b). ■

Lemma 12. *A direct result of Part (a) implies that* $\mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k,N_k\right)\rightarrow-\infty\right\}=0.$

Proof. The proof follows from

$$\mathbb{P}\left\{\bar{F}_k\left(\boldsymbol{X}_k,N_k\right)\rightarrow-\infty\right\}\leq\mathbb{P}\left\{\left|\bar{F}_k\left(\boldsymbol{X}_k,N_k\right)-f\left(\boldsymbol{X}_k\right)\right|>c,\text{ i.o.}\right\}$$

for any $c>0$. ■

For the purpose of proving the lim-inf type convergence recall the sets defined in Definition 2. As shown in Chapter 2 these sets are used to specify a space where we can assume Lipschitz continuous gradients for the underlying function, since making such assumption over the whole domain of f excludes a large range of functions, such as $f\left(\boldsymbol{x}\right)=\boldsymbol{x}^3$. However making an assumption that f has Lipschitz continuous gradients on $\mathcal{S}(R_0)$ represents functions that have locally Lipschitz continuous gradients; this also includes $f\left(\boldsymbol{x}\right)=\boldsymbol{x}^3$. In the deterministic context we know that $f\left(\boldsymbol{x}_k\right)$ is monotonically

decreasing and hence $\{\boldsymbol{x}_k\}$ lives in $\mathcal{S}$. However in the stochastic context $f(\boldsymbol{X}_k)$ may not be monotonically decreasing. To see this more clearly think about a situation where $\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) < \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \eta_3 \left(M_k(\boldsymbol{0}) - M_k(\boldsymbol{S}_k)\right)$ that means $\hat{\rho}_k > \eta_3$ and hence $\tilde{\boldsymbol{X}}_{k+1}$ will be accepted as the new iterate; but we also have $\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) > \bar{F}(\boldsymbol{X}_k, N_k)$ due to new random observations in Step 1 of Algorithm 4 that obtain a larger estimate $\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right)$. In this example $\bar{F}(\boldsymbol{X}_{k+1}, N_{k+1}) > \bar{F}(\boldsymbol{X}_k, N_k)$. As a result and using Lemma 11 one can say that $\{\boldsymbol{X}_k\}$ leaves the set $\mathcal{S}$ with positive probability. We are interested in a larger set $\mathcal{S}'' \supseteq \mathcal{S}$ that contains $\{\boldsymbol{X}_k\}$ with probability one. Then we make the Lipschitz continuous gradients assumption on that set.

It is clear that when the function is bounded from above, finding such a set is trivial ($\mathcal{S}'' = \left\{\boldsymbol{x} : f(\boldsymbol{x}) \leq \sup_{\boldsymbol{z} \in \mathbb{R}^d} f(\boldsymbol{z})\right\}$). The next Lemma provides similar results when the function is not bounded from above.

Lemma 13. *Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable. Then*

$$\mathbb{P}\left\{\limsup_{k \to \infty} f(\boldsymbol{X}_k) < +\infty\right\} = 1$$

where $\{\boldsymbol{X}_k\}$ is generated by Algorithm 4.

Proof. By similar arguments in the proof of Lemma 11, we observe that for some $\delta > 0$ and sufficiently large k,

$$
\begin{aligned}
&\mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{k^{1+\varepsilon}}\right\} \\
&\leq \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{k^{1+\varepsilon}} \mid \hat{\rho}_{k-1} > \eta_3\right\} \\
&\quad + \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{k^{1+\varepsilon}} \mid \hat{\rho}_{k-1} \leq \eta_3\right\} \\
&\leq \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, \tilde{N}_{k-1}\right) > \frac{1}{2k^{1+\varepsilon}} \mid \hat{\rho}_{k-1} > \eta_3\right\} \\
&\qquad + \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_{k-1}, \tilde{N}_{k-1}\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{2k^{1+\varepsilon}} \mid \hat{\rho}_{k-1} > \eta_3\right\} \\
&\quad + \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - f\left(\boldsymbol{X}_k\right) > \frac{1}{2k^{1+\varepsilon}} \mid \hat{\rho}_{k-1} \leq \eta_3\right\} \\
&\qquad + \mathbb{P}\left\{f\left(\boldsymbol{X}_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{2k^{1+\varepsilon}} \mid \hat{\rho}_{k-1} \leq \eta_3\right\} \\
&\leq \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_{k-1}, \tilde{N}_{k-1}\right) - f\left(\boldsymbol{X}_{k-1}\right) > \frac{1}{4k^{1+\varepsilon}}\right\} + \mathbb{P}\left\{f\left(\boldsymbol{X}_{k-1}\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{4k^{1+\varepsilon}}\right\} \\
&\quad + \mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_{k-1}, N_k\right) - f\left(\boldsymbol{X}_{k-1}\right) > \frac{1}{2k^{1+\varepsilon}}\right\} + \mathbb{P}\left\{f\left(\boldsymbol{X}_{k-1}\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > \frac{1}{2k^{1+\varepsilon}}\right\} \\
&\leq 4\left(1+\delta\right)\left(4k^{1+\varepsilon}\right)^2 \kappa_f^2 \Delta_{\max}^4 \lambda_k^{-1} \leq 64\left(1+\delta\right)\kappa_f^2 \Delta_{\max}^4 k^{-(1+\varepsilon)}. \qquad (3.13)
\end{aligned}
$$

Note that in the second and third inequality of (3.13) we have used the Boole's inequality (see Definition 9). Furthermore, notice that in all the terms but the first term in the second inequality, differences of the estimated function value at the same point are considered; and for the first term we can write $\mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, \tilde{N}_{k-1}\right) > 2^{-1}k^{-(1+\varepsilon)} \mid \hat{\rho}_{k-1} > \eta_3\right\} = 0$ from observing that when an iteration is successful it must be true that estimated function value decreases moving from the current iterate to the candidate solution. In the third and fourth terms of the third inequality we have also replaced $\boldsymbol{X}_k$ with $\boldsymbol{X}_{k-1}$ since the iteration is unsuccessful.

As a result of (3.13), Borel-Cantelli (Lemma 10) implies that

$$
\mathbb{P}\left\{\bar{F}\left(\boldsymbol{X}_k, N_k\right) - \bar{F}\left(\boldsymbol{X}_{k-1}, N_{k-1}\right) > k^{-(1+\varepsilon)} \text{ i.o.}\right\} = 0. \qquad (3.14)
$$

Next we re-write $f(\boldsymbol{X}_k) - f(\boldsymbol{X}_{k-1})$ and arrive at the following:

$$\begin{aligned}\mathbb{P}\left\{f(\boldsymbol{X}_k) - f(\boldsymbol{X}_{k-1}) > k^{-(1+\varepsilon)}\right\} &= \mathbb{P}\left\{f(\boldsymbol{X}_k) - \bar{F}(\boldsymbol{X}_k, N_k) > 3^{-1}k^{-(1+\varepsilon)}\right\} \\ &+ \mathbb{P}\left\{\bar{F}(\boldsymbol{X}_k, N_k) - \bar{F}(\boldsymbol{X}_{k-1}, N_{k-1}) > 3^{-1}k^{-(1+\varepsilon)}\right\} \\ &+ \mathbb{P}\left\{\bar{F}(\boldsymbol{X}_{k-1}, N_{k-1}) - f(\boldsymbol{X}_{k-1}) > 3^{-1}k^{-(1+\varepsilon)}\right\},\end{aligned}$$

and hence conclude that $\mathbb{P}\left\{f(\boldsymbol{X}_k) - f(\boldsymbol{X}_{k-1}) > k^{-(1+\varepsilon)} \text{ i.o.}\right\} = 0$ by (3.14) and part (i) of Lemma 11. Therefore for a given $\omega \in \Omega$ there exists $K(\omega)$ with $f(\boldsymbol{X}_k(\omega)) - f(\boldsymbol{X}_{k-1}(\omega)) \leq k^{-(1+\varepsilon)}$ for all $k \geq K(\omega)$. It follows that

$$\begin{aligned}\limsup_{k\to\infty} f(\boldsymbol{X}_k(\omega)) = f(\boldsymbol{x}_0) &+ \sum_{k<K(\omega)} f(\boldsymbol{X}_k(\omega)) - f(\boldsymbol{X}_{k-1}(\omega)) \\ &+ \sum_{k\geq K(\omega)} k^{-(1+\varepsilon)} < +\infty.\end{aligned}$$

In the above $\sum_{k<K(\omega)} f(\boldsymbol{X}_k(\omega)) - f(\boldsymbol{X}_{k-1}(\omega)) < \infty$ since by mean value theorem we have for some $t \in [0,1]$

$$f(\boldsymbol{X}_k(\omega)) - f(\boldsymbol{X}_{k-1}(\omega)) \leq \|\nabla f(t\boldsymbol{X}_k(\omega) + (1-t)\boldsymbol{X}_{k-1}(\omega))\| \Delta_{k-1}(\omega),$$

and $\|\nabla f(\cdot)\| < +\infty$ in a compact neighborhood by the continuous differentiability of f. ■

A direct implication of Lemma 13 is that for every $\omega \in \Omega$ there exists $L''(\omega) < \infty$ such that for all k, $\boldsymbol{X}_k(\omega) \in \mathcal{S}''(\omega)$ where $\mathcal{S}''(\omega) = \{\boldsymbol{x} | f(\boldsymbol{x}) \leq L''(\omega)\}$. Then we assume f to be continuously differentiable with Lipschitz gradients on such a set with $\nu_{gL}(\omega)$ being the Lipschitz constant of the gradient over that set. We are now ready for the lim-inf type proof.

With the results above, the almost sure lim-inf type convergence to a first-order critical point are now available, presented as follows:

Theorem 8. *Suppose f is continuously differentiable and bounded from below on the level set $\mathcal{S}$. Let a sequence $\{\boldsymbol{X}_k\}$ be generated by Algorithm 4 and Assumptions 3, 4 and 5 hold for every iteration k. Suppose further that f has Lipschitz continuous gradients on a set that contains all $\{\boldsymbol{X}_k\}$. Then $\liminf_{k\to\infty} \|\boldsymbol{g}(\boldsymbol{X}_k)\| = 0$ with probability one.*

Proof. From the definition of $\hat{\rho}_k$ we can write

$$|\hat{\rho}_k - 1| = \left| \frac{M_k(\boldsymbol{S}_k) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right)}{M_k(\boldsymbol{0}) - M_k(\boldsymbol{S}_k)} \right|.$$

To bound $|\hat{\rho}_k - 1|$ first we rewrite

$$\begin{aligned}
&\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) - \bar{F}\left(\tilde{\boldsymbol{X}}_k, \tilde{N}_k\right) \\
&= \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) - f\left(\tilde{\boldsymbol{X}}_{k+1}\right) + f\left(\tilde{\boldsymbol{X}}_{k+1}\right) - f(\boldsymbol{X}_k) + f(\boldsymbol{X}_k) - \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right).
\end{aligned}$$

Then we have that

$$\begin{aligned}
\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) = \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) &+ \boldsymbol{g}(\boldsymbol{X}_k)^T \boldsymbol{S}_k \\
&+ \int_0^1 \left(\boldsymbol{g}(\boldsymbol{X}_k + t\boldsymbol{S}_k) - \boldsymbol{g}(\boldsymbol{X}_k)\right)^T \boldsymbol{S}_k dt \\
&+ \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) - f\left(\tilde{\boldsymbol{X}}_{k+1}\right) \\
&+ f(\boldsymbol{X}_k) - \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right).
\end{aligned}$$

And therefore

$$\begin{aligned}
M_k(\boldsymbol{S}_k) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) = \left(\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \boldsymbol{g}(\boldsymbol{X}_k)\right)^T \boldsymbol{S}_k &\\
+ \frac{1}{2} \boldsymbol{S}_k^T \hat{\mathscr{B}}_k \boldsymbol{S}_k &\\
- \int_0^1 \left(\boldsymbol{g}(\boldsymbol{X}_k + t\boldsymbol{S}_k) - \boldsymbol{g}(\boldsymbol{X}_k)\right)^T \boldsymbol{S}_k dt &\\
+ f\left(\tilde{\boldsymbol{X}}_{k+1}\right) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) &\\
+ \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - f(\boldsymbol{X}_k).
\end{aligned}$$

We now bound each of the expressions on the right hand side. Let $\omega \in \Omega$ be any sample path and $c_f, c_g > 0$ be fixed. Lemma 11 (a) implies that except for a set of measure 0 the difference between the function estimate and true function value is bounded above for large k. So we say that there exists $K_f(\omega)$ such that

$$\left|\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}(\omega), \tilde{N}_{k+1}(\omega)\right) - f\left(\tilde{\boldsymbol{X}}_{k+1}(\omega)\right)\right| \leq 2^{-1} c_f \Delta_k^2(\omega)$$

and

$$|\bar{F}(\boldsymbol{X}_k(\omega), N_k(\omega)) - f(\boldsymbol{X}_k(\omega))| \leq 2^{-1} c_f \Delta_k^2(\omega)$$

for all $k \geq K_f(\omega)$. Similarly Lemma 11 (b) implies that there exists $K_g(\omega)$ such that

$$\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k(\omega), \tilde{N}_k(\omega)\right) - \boldsymbol{g}\left(\boldsymbol{X}_k(\omega)\right)\right\| \leq c_g \Delta_k(\omega)$$

for $k \geq K_g(\omega)$. We also let $\nu_{gL}(\omega)$ be the Lipschitz constant for $\boldsymbol{g}$ on the set $\mathcal{S}''(\omega)$, with $\mathbb{P}\{\boldsymbol{X}_k \in \mathcal{S}''(\omega)\} = 1$ which exists according to Lemma 13. Hence

$$\left|M_k(\boldsymbol{S}_k(\omega)) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}(\omega), \tilde{N}_{k+1}(\omega)\right)\right| \leq \Delta_k^2(\omega)\left(c_g + \frac{\kappa_{bhm}}{2} + \nu_{gL}(\omega) + c_f\right)$$

if $k \geq \max\left\{K_f(\omega), K_g(\omega)\right\}$.

Moreover for the purpose of contrapositive define

$$\mathcal{E} = \{\omega : \text{ there exists } \varepsilon(\omega) > 0, K_1(\omega) \ s.t. \ k \geq K_1(\omega) \Rightarrow \|\boldsymbol{g}(\boldsymbol{X}_k(\omega))\| \geq 2\varepsilon(\omega)\}.$$

For $\omega_0 \in \mathcal{E}$, we can also find $\varepsilon(\omega_0)$ and $K'(\omega_0)$ such that

$$\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k(\omega_0), \tilde{N}_k(\omega_0)\right) - \boldsymbol{g}\left(\boldsymbol{X}_k(\omega_0)\right)\right\| \leq \varepsilon(\omega_0)$$

if $k \geq K'(\omega_0)$. Then for $k \geq \max\{K_1(\omega_0), K'(\omega_0)\}$ we have

$$\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k(\omega_0), \tilde{N}_k(\omega_0)\right)\right\| \geq \|\boldsymbol{g}(\boldsymbol{X}_k(\omega_0))\| - \left\|\boldsymbol{g}(\boldsymbol{X}_k(\omega_0)) - \bar{\boldsymbol{G}}\left(\boldsymbol{X}_k(\omega_0), \tilde{N}_k(\omega_0)\right)\right\| \geq \varepsilon(\omega_0)$$

and hence $|M_k(\boldsymbol{0}) - M_k(\boldsymbol{X}_k(\omega_0))| \geq c_1 \varepsilon(\omega_0) \min\left\{\Delta_k(\omega_0), \frac{\varepsilon(\omega_0)}{\kappa_{bhm}}\right\}$.

We can also find the gradient Lipschitz constant $\nu_{gL}(\omega_0)$ on the set $\mathcal{S}''(\omega_0)$ that contains all $\boldsymbol{X}_k(\omega_0)$. Now, let $K(\omega_0) = \max\left\{K_1(\omega_0), K'(\omega_0), K_f(\omega_0), K_g(\omega_0)\right\}$ where $K_f(\omega_0)$ and $K_g(\omega_0)$ are also chosen as explained above. Note that for $k \geq K(\omega_0)$,

$$\begin{aligned} |\hat{\rho}_k(\omega_0) - 1| &= \left|\frac{M_k(\boldsymbol{S}_k(\omega_0)) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}(\omega_0), \tilde{N}_{k+1}(\omega_0)\right)}{M_k(\boldsymbol{0}) - M_k(\boldsymbol{S}_k(\omega_0))}\right| \\ &\leq \frac{\Delta_k^2(\omega_0)\left(c_g + \frac{\kappa_{bhm}}{2} + \nu_{gL}(\omega_0) + c_f\right)}{c_1 \varepsilon(\omega_0) \min\left\{\Delta_k(\omega_0), \frac{\varepsilon(\omega_0)}{\kappa_{bhm}}\right\}}. \end{aligned} \tag{3.15}$$

Define

$$\bar{\Delta}(\omega_0) = \frac{c_1 \varepsilon(\omega_0)}{2\left(c_g + \frac{\kappa_{bhm}}{2} + \nu_{gL}(\omega_0) + c_f\right)}. \tag{3.16}$$

Note that since $c_1 < 1$, $\bar{\Delta}(\omega_0) \leq \kappa_{bhm}^{-1}\varepsilon(\omega_0)$. It follows that if $\Delta_k(\omega_0) \leq \bar{\Delta}(\omega_0)$ then $\Delta_k(\omega_0) \leq \kappa_{bhm}^{-1}\varepsilon(\omega_0)$ and hence $\min\left\{\Delta_k(\omega_0), \frac{\varepsilon(\omega_0)}{\kappa_{bhm}}\right\} = \Delta_k(\omega_0)$ in the denominator of the right hand side in (3.15). Consequently using (3.16) we get

$$\begin{aligned} |\hat{\rho}_k(\omega_0) - 1| &\leq \Delta_k(\omega_0) \frac{c_g + \frac{\kappa_{bhm}}{2} + \nu_{gL}(\omega_0) + c_f}{c_1 \varepsilon(\omega_0)} \\ &\leq \frac{1}{2}, \end{aligned}$$

for all $\Delta_k(\omega_0) \leq \bar{\Delta}(\omega_0)$. In other words the iterations are eventually with $\hat{\rho}_k(\omega_0) \geq \eta_2$ and as a result $\Delta_{k+1}(\omega_0) \geq \Delta_k(\omega_0)$ whenever $\Delta_k(\omega_0) \leq \bar{\Delta}(\omega_0)$. We conclude that

$$\Delta_k(\omega_0) \geq \min\left\{\gamma_1 \bar{\Delta}(\omega_0), \Delta_K(\omega_0)\right\} \tag{3.17}$$

for all $k \geq K(\omega_0)$.

Next, we can write

$$\begin{aligned} \bar{F}(\boldsymbol{X}_{m+1}, N_{m+1}) - \bar{F}(\boldsymbol{X}_1, N_1) &= \sum_{i=1}^{m} \bar{F}(\boldsymbol{X}_{i+1}, N_{i+1}) - \bar{F}(\boldsymbol{X}_i, N_i) \\ &= \sum_{i=1}^{m} (A_i + B_i), \end{aligned}$$

where $A_i = \bar{F}(\boldsymbol{X}_{i+1}, N_{i+1}) - \bar{F}\left(\boldsymbol{X}_i, \tilde{N}_i\right)$ is the change in the estimated function value at the incumbent solution at the beginning and at the end of iteration i and $B_i = \bar{F}\left(\boldsymbol{X}_i, \tilde{N}_i\right) - \bar{F}(\boldsymbol{X}_i, N_i)$ is the change in the estimated function value at $\boldsymbol{X}_i$ with possibly different sample sizes at the end of iteration $i-1$ and the at the beginning of iteration i.

Now Define

$$\mathcal{D} = \{\omega \in \mathcal{E} : \text{ there exists a subsequence } \mathcal{K}(\omega) \text{ with } \hat{\rho}_k(\omega) > \eta_2\},$$

and let $\omega_1 \in \mathcal{D}$ with $\varepsilon(\omega_1)$ and $K_1(\omega_1)$ chosen accordingly. If we further suppose that there exists $\delta(\omega_1) > 0$, $K_d(\omega_1)$ such that $k \geq K_d(\omega_1) \Rightarrow \Delta_k(\omega_1) \geq \delta(\omega_1)$, then the following will hold:

(i) $A_k(\omega_1) = 0$ whenever $\hat{\rho}_k(\omega_1) \leq \eta_3$ since by Step 7 of Algorithm 4, $\boldsymbol{X}_{k+1}(\omega_1) = \boldsymbol{X}_k(\omega_1)$ and $N_{k+1}(\omega_1) = \tilde{N}_k(\omega_1)$.

(ii) $A_k(\omega_1) \leq -\eta_3 c_1 \varepsilon(\omega_1) \min\left\{\delta(\omega_1), \frac{\varepsilon(\omega_1)}{\kappa_{bhm}}\right\}$ by Cauchy reduction in Assumption 4 whenever $\hat{\rho}_k(\omega_1) \geq \eta_3$ and $k \geq \max(K_1(\omega_1), K_d(\omega_1))$.

(iii) For a set of consecutive iterations $i = k_1, k_1+1, \cdots, k_2$ for which $\hat{\rho}_k(\omega_1) < \eta_3$ and hence $\boldsymbol{X}_{k_1}(\omega_1) = \boldsymbol{X}_{k_1+1}(\omega_1) = \cdots = \boldsymbol{X}_{k_2}(\omega_1)$, the summation of $B_i(\omega_1)$'s can be written as

$$\begin{aligned}
\sum_{i=k_1}^{k_2} B_i(\omega_1) &= \sum_{i=k_1}^{k_2} \bar{F}\left(\boldsymbol{X}_i(\omega_1), \tilde{N}_i(\omega_1)\right) - \bar{F}\left(\boldsymbol{X}_i(\omega_1), N_i(\omega_1)\right) \\
&= \bar{F}\left(\boldsymbol{X}_{k_2}(\omega_1), \tilde{N}_{k_2}(\omega_1)\right) - \bar{F}\left(\boldsymbol{X}_{k_2}(\omega_1), N_{k_1}(\omega_1)\right) \\
\Rightarrow \left|\sum_{i=k_1}^{k_2} B_i(\omega_1)\right| &\leq \left|\bar{F}\left(\boldsymbol{X}_{k_2}(\omega_1), \tilde{N}_{k_2}(\omega_1)\right) - f\left(\boldsymbol{X}_{k_2}(\omega_1)\right)\right| \\
&\quad + \left|f\left(\boldsymbol{X}_{k_2}(\omega_1)\right) - \bar{F}\left(\boldsymbol{X}_{k_2}(\omega_1), N_{k_1}(\omega_1)\right)\right| \\
&\leq \frac{2}{3}\eta_3 c_1 \varepsilon(\omega_1) \min\left\{\delta(\omega_1), \frac{\varepsilon(\omega_1)}{\kappa_{bhm}}\right\},
\end{aligned}$$

when k_1 larger than $\max(K_1(\omega_1), K_d(\omega_1))$ and $K_f(\omega_1)$ chosen for

$$c_f = \frac{1}{3}\eta_3 c_1 \varepsilon(\omega_1) \min\left\{\delta(\omega_1), \frac{\varepsilon(\omega_1)}{\kappa_{bhm}}\right\}$$

in Lemma 11 (a).

Suppose $\mathcal{Q}_m$ is the subset of iterations in the first m iterations with $\hat{\rho}_{k_j} \geq \eta_2$ for $k_j \in \mathcal{Q}_m, j = 1, 2, \cdots, Q_m$ (letting $Q(m) = |\mathcal{Q}_m|$); we refer to these iterations as non-contracting iterations as the trust-region size for them either expands or does not change (see Figure 3.1). Between every two non-contracting iteration k_j and k_{j+1} there may be several successful and unsuccessful iterations since $\eta_2 > \eta_3$. Define W_j to be the number of successful iterations between k_j and k_{j+1}. Then define t_j^i for $i = 0, 1, 2, \cdots, W_j + 1$ as follows:

- $t_0^0 = 1$.
- $t_j^0 = k_j$.
- t_j^i is the i-th successful iteration between k_j and k_{j+1} for $i = 1, 2, \cdots, W_j$.
- $t_j^{W_j+1} = k_{j+1}$.

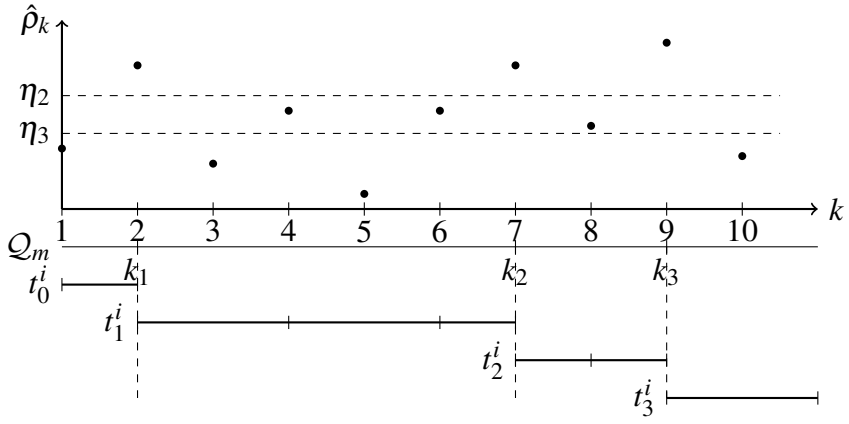


Figure 3.2. An example of the set-up used in the proof of Theorem 8.

- $t_{Q_m}^{W_{Q(m)}+1} = m+1$.

This set up is presented in an example in Figure 3.2. In this figure, we have $m = 10, Q(m) = 3$ and the number of successes between every two non-contracting iterations is $W_0 = 0, W_1 = 2, W_2 = 1, W_3 = 0$. As explained above we also have

$$t_0^0 = 1, t_0^1 = 2;$$
$$t_1^0 = 2, t_1^1 = 4, t_1^2 = 6, t_1^3 = 7;$$
$$t_2^0 = 7, t_2^1 = 8, t_2^2 = 9;$$
$$t_3^0 = 9, t_3^1 = 11.$$

It is now clear that for $K'' = \max\left\{K_1, K_d, K_f\right\}$ and $Q(K'')$ being the number of non-contracting iterations in the first K'' one can write

$$\begin{aligned}
\sum_{k=1}^{m}(A_k+B_k) &= \sum_{j=1}^{Q(m)}\sum_{i=0}^{W_j}\sum_{k=t_j^i}^{t_j^{i+1}-1}(A_k+B_k) \\
&= \sum_{k=1}^{t^0_{Q(K'')}}(A_k+B_k) + \sum_{j=Q(K'')+1}^{Q(m)}\sum_{i=0}^{W_j}\sum_{k=t_j^i}^{t_j^{i+1}-1}(A_k+B_k) \\
&\leq \sum_{k=1}^{t^0_{Q(K'')}}(A_k+B_k) + \sum_{j=Q(K'')+1}^{Q(m)}\sum_{i=0}^{W_j}\left(-\frac{\eta_3 c_1 \varepsilon \min\left\{\delta, \frac{\varepsilon}{\kappa_{bhm}}\right\}}{3}\right) \\
&\leq \sum_{k=1}^{t^0_{Q(K'')}}(A_k+B_k) - \left(Q(m)-Q\left(K''\right)\right)\frac{\eta_3 c_1 \varepsilon \min\left\{\delta, \frac{\varepsilon}{\kappa_{bhm}}\right\}}{3}.
\end{aligned}$$

Note that although we still live in the set $\mathcal{D}$ we have dropped ω_1 for readability. In the second step we have used (i), (ii) and (iii) to say that the difference of the function estimates between (t_j^i)-th and $(t_j^{i+1}-1)$-th iteration, with only the first being a success, is at most $-\frac{1}{3}\eta_3 c_1 \varepsilon \min\left\{\delta, \frac{\varepsilon}{\kappa_{bhm}}\right\}$. We also use $W_j \geq 0$ to remove the second summation. Now we conclude that since for $\omega_1 \in D$ there is an entire subsequence $\mathcal{K}(\omega_1)$ of non-contracting iterations, then $Q(m) \to \infty$ as $m \to \infty$ which implies $\bar{F}\left(\boldsymbol{X}_{m+1}(\omega_1), N_{m+1}(\omega_1)\right) \to -\infty$ as $m \to \infty$. By Lemma 12 though the set of such ω_1 is a set of measure zero. This implies that $\Delta_k(\omega_1) \to 0$ but this also contradicts inequality (3.17). So $\mathbb{P}\{\mathcal{D}\} = 0$.

So we must have that for every $\omega \in \mathcal{E}$, $\hat{\rho}_k(\omega) < \eta_2$ for k sufficiently large. But this implies that the trust-region radius eventually keeps contracting and hence $\Delta_k(\omega) \to 0$ which again contradicts inequality (3.17). The conclusion is that $\mathbb{P}\{\mathcal{E}\} = 0$. In other words $\mathbb{P}\{\liminf_{k\to\infty} \|\boldsymbol{g}(\boldsymbol{X}_k)\| = 0\} = 1$. ■

Finally we prove the almost sure convergence of the ASTRO algorithm to a first-order critical point.

Theorem 9. *Suppose f is continuously differentiable and bounded from below on the level set $\mathcal{S}$. Let a sequence $\{\boldsymbol{X}_k\}$ be generated by Algorithm 4 and Assumptions 3, 4 and 5 hold for every iteration k. Suppose further that f has Lipschitz continuous gradients on a set that contains all $\{\boldsymbol{X}_k\}$. Then $\lim_{k\to\infty} \|\boldsymbol{g}(\boldsymbol{X}_k)\| = 0$ with probability one.*

Proof. We consider two cases: In the first case we assume that there are only finitely many successful iterations ($\hat{\rho}_k > \eta_3$) and in the second case we assume that the successful iterations are infinite.

CASE 1 (finitely many successful iterations): We have shown in Theorem 8 that for any given sample path $\omega \in \Omega$ if there exists a lower bound for the norm of the function gradient $\|\boldsymbol{g}(\boldsymbol{X}_k(\omega))\|$, then there exists a lower bound for the trust-region radius. This lower bound is derived in inequality (3.17). But if the successful iterations are finite, then for sure $\Delta_{k+1}(\omega) < \Delta_k(\omega)$ for large k which implies that $\lim_{k\to\infty} \Delta_k(\omega) = 0$. This then proves that there is no lower bound for the norm of the function gradient.

CASE 2 (infinite successful iterations): The following results hold for any given sample path $\omega \in \Omega$, but for the sake of readability we remove ω. Suppose there is a subsequence of successful iterations $\{t_i\}$ such that $\|\boldsymbol{g}(\boldsymbol{X}_{t_i})\| \geq 3\varepsilon$ for some $\varepsilon > 0$. By Theorem 8 we can find another subsequence $\{\ell_i\}$ where $\ell_i = \ell(t_i)$ is the first successful iteration after t_i such that $\|\boldsymbol{g}(\boldsymbol{X}_{\ell_i})\| < 2\varepsilon$. Let

$$\mathcal{K} = \{k : \hat{\rho}_k > \eta_3, t_i \leq k < \ell_i\}.$$

We know that $\|\boldsymbol{g}(\boldsymbol{X}_k)\| \geq 2\varepsilon$ for all $k \in \mathcal{K}$ and further $\left\|\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right)\right\| \geq \varepsilon$ for sufficiently large $k \in \mathcal{K}$ by Lemma 11. If $\Delta_k > \delta$ for some $\delta > 0$ when k is large, we have seen in the proof of Theorem 8 that the function estimate reduces between the two consecutive successful iterations at least by a fixed amount, leading to a contradiction with Lemma 12 that states the function estimates remain bounded almost surely. This implies that we must have $\Delta_k \to 0$ as $k \to \infty$. Hence for sufficiently large $k \in \mathcal{K}$ we have $\Delta_k < \frac{\varepsilon}{\kappa_{bhm}}$ leading to

$$\begin{aligned}
\bar{F}(\boldsymbol{X}_{k+1}, N_{k+1}) - \bar{F}(\boldsymbol{X}_k, N_k) &= \bar{F}(\boldsymbol{X}_{k+1}, N_{k+1}) - \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) \\
&\quad + \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \bar{F}(\boldsymbol{X}_k, N_k) \\
&\leq -\eta_3 c_1 \varepsilon \min\left\{\Delta_k, \frac{\varepsilon}{\kappa_{bhm}}\right\} + \frac{\eta_3 c_1 \varepsilon}{2}\Delta_k \\
&\leq -\frac{\eta_3 c_1 \varepsilon}{2}\Delta_k \\
\Rightarrow \Delta_k &\leq \frac{-2}{\eta_3 c_1 \varepsilon}\left(\bar{F}(\boldsymbol{X}_{k+1}, N_{k+1}) - \bar{F}(\boldsymbol{X}_k, N_k)\right).
\end{aligned}$$

The second step of the above uses $c_f = \frac{\eta_3 c_1 \varepsilon}{4}$ in part (a) of Lemma 11 suggesting that we eventually have

$$\begin{aligned}\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \bar{F}\left(\boldsymbol{X}_k, N_k\right) &\leq \left|\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - f\left(\boldsymbol{X}_k\right)\right| + \left|f\left(\boldsymbol{X}_k\right) - \bar{F}\left(\boldsymbol{X}_k, N_k\right)\right| \\ &\leq 2\left(\frac{\eta_3 c_1 \varepsilon}{4}\Delta_k\right)\end{aligned}$$

with probability one. Now we deduce for a large enough i

$$\begin{aligned}\left\|\boldsymbol{X}_{\ell_i} - \boldsymbol{X}_{t_i}\right\| &\leq \sum_{\substack{j=t_i \\ j\in\mathcal{K}}}^{\ell_i-1} \left\|\boldsymbol{X}_{j+1} - \boldsymbol{X}_j\right\| \leq \sum_{\substack{j=t_i \\ j\in\mathcal{K}}}^{\ell_i-1} \Delta_j \\ &\leq \frac{-2}{\eta_3 c_1 \varepsilon} \sum_{\substack{j=t_i \\ j\in\mathcal{K}}}^{\ell_i-1} \bar{F}\left(\boldsymbol{X}_{j+1}, N_{j+1}\right) - \bar{F}\left(\boldsymbol{X}_j, N_j\right) \\ &= \frac{-2}{\eta_3 c_1 \varepsilon}\left(\bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i}\right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i}\right) - \sum_{\substack{j=t_i \\ j\notin\mathcal{K}}}^{\ell_i-1} \left(\bar{F}\left(\boldsymbol{X}_{j+1}, N_{j+1}\right) - \bar{F}\left(\boldsymbol{X}_j, N_j\right)\right)\right) \\ &\leq \frac{-2}{\eta_3 c_1 \varepsilon}\left(\bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i}\right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i}\right) + \theta c_f \max_{\substack{t_i \leq j < \ell_i \\ j\in\mathcal{K}}} \Delta_j\right),\end{aligned}$$

where θ is the number of unsuccessful iterations between t_i and ℓ_i and $c_f > 0$ is some positive constant. We observe that the right hand side of the above expression tends to zero since $\max_{t_i \leq j < \ell_i, j\in\mathcal{K}} \Delta_j \to 0$, and in addition $\bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i}\right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i}\right) \to 0$ due to the following

$$\begin{aligned}\bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i}\right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i}\right) &\leq \sum_{\substack{k_j=t_i \\ k_j\in\mathcal{K}}}^{\ell_i-1} \sum_{s=k_j}^{k_{j+1}-1} \bar{F}\left(\boldsymbol{X}_{s+1}, N_{s+1}\right) - \bar{F}\left(\boldsymbol{X}_s, N_s\right) \\ &= \sum_{\substack{k_j=t_i \\ k_j\in\mathcal{K}}}^{\ell_i-1}\left(\bar{F}\left(\boldsymbol{X}_{k_j+1}, N_{k_j+1}\right) - \bar{F}\left(\boldsymbol{X}_{k_j}, \tilde{N}_{k_j}\right) + \sum_{s=k_j}^{k_{j+1}-1} \bar{F}\left(\boldsymbol{X}_s, \tilde{N}_s\right) - \bar{F}\left(\boldsymbol{X}_s, N_s\right)\right) \\ &\leq \sum_{\substack{k_j=t_i \\ k_j\in\mathcal{K}}}^{\ell_i-1}\left(-\eta_3 c_1 \varepsilon \Delta_{k_j} + \sum_{s=k_j}^{k_{j+1}-1} 2c_f \Delta_s\right) \\ &\leq \left(-\eta_3 c_1 \varepsilon |\mathcal{K}| + 2c_f |t_i - \ell_i|\right) \max_{\substack{k_j \leq s < k_{j+1} \\ k_j\in\mathcal{K}}} \Delta_s, \qquad (3.18)\end{aligned}$$

in which again $\max_{k_j \le s < k_{j+1}, k_j \in \mathcal{K}} \Delta_s \to 0$. In the second line of (3.18) we have used the fact that

$$\begin{aligned}\bar{F}\left(\boldsymbol{X}_{k+1}, N_{k+1}\right) - \bar{F}\left(\boldsymbol{X}_k, N_k\right) &= \bar{F}\left(\boldsymbol{X}_{k+1}, N_{k+1}\right) - \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) + \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \bar{F}\left(\boldsymbol{X}_s, N_s\right) \\ &= \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \bar{F}\left(\boldsymbol{X}_s, N_s\right)\end{aligned}$$

for all unsuccessful k; while the only nonzero $\bar{F}\left(\boldsymbol{X}_{k+1}, N_{k+1}\right) - \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right)$ belongs to the successful k's that remain outside the inner summation.

We conclude that $\|\boldsymbol{X}_{\ell_i} - \boldsymbol{X}_{t_i}\| \xrightarrow{wp1} 0$. We also observe from Lemma 11 that

$$\begin{aligned}\left|f\left(\boldsymbol{X}_{\ell_i}\right) - f\left(\boldsymbol{X}_{t_i}\right)\right| &\le \left|\bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i}\right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i}\right)\right| \\ &\quad + \left|f\left(\boldsymbol{X}_{\ell_i}\right) - \bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i}\right)\right| \\ &\quad + \left|f\left(\boldsymbol{X}_{t_i}\right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i}\right)\right| \\ &\xrightarrow{wp1} 0. \end{aligned} \tag{3.19}$$

Therefore by the continuity of the gradient we must have $\|\boldsymbol{g}\left(\boldsymbol{X}_{\ell_i}\right) - \boldsymbol{g}\left(\boldsymbol{X}_{t_i}\right)\| \xrightarrow{wp1} 0$ but this indicates $\|\boldsymbol{g}\left(\boldsymbol{X}_{\ell_i}\right) - \boldsymbol{g}\left(\boldsymbol{X}_{t_i}\right)\| < \varepsilon$ for large i, contradicting the definition of t_i and ℓ_i. ■

3.6 Implementation and Numerical Experiments

Implementation of the ASTRO is straight forward as listed in Algorithm 4, due to the fact that the sample size for estimating the function and gradient values are quite explicitly determined. The remaining question in the implementation is the symmetric matrix $\hat{\mathscr{B}}_k$ that has the role of approximating the Hessian. We use quasi-Newton techniques to obtain this approximation. Quasi-Newton methods obtain some measure of the function curvature by using only gradient information. They are very practical since the second derivative information is not required and have been shown to provide super-linear convergence with $\boldsymbol{s} = -\mathscr{B}^{-1}\boldsymbol{g}$, that is much faster than using just steepest descent methods. The two famous quasi-Newton approaches are BFGS and SR1.

Let us define $\boldsymbol{Y}_k = \bar{\boldsymbol{G}}\left(\boldsymbol{X}_{k+1}, \tilde{N}_{k+1}\right) - \bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right)$ that indicates the change in the estimated gradient when moving from $\boldsymbol{X}_k$ to $\boldsymbol{X}_{k+1}$, where $\bar{\boldsymbol{G}}\left(\boldsymbol{X}_{k+1}, \tilde{N}_{k+1}\right)$ is the estimated gradient at the end of Step 1 of iteration $k+1$ and $\bar{\boldsymbol{G}}\left(\boldsymbol{X}_k, \tilde{N}_k\right)$ is the estimated gradient

value at the end of Step 1 of iteration k, when iteration k is successful that is $\boldsymbol{X}_{k+1} \neq \boldsymbol{X}_k$. Then by the *secant formula* we must have $\hat{\mathscr{B}}_{k+1}\boldsymbol{S}_k = \boldsymbol{Y}_k$ before constructing the new model in Step 2.

In BFGS method the update formula

$$\hat{\mathscr{B}}_{k+1} = \hat{\mathscr{B}}_k - \frac{\hat{\mathscr{B}}_k \boldsymbol{S}_k \boldsymbol{S}_k^T \hat{\mathscr{B}}_k}{\boldsymbol{S}_k^T \hat{\mathscr{B}}_k \boldsymbol{S}_k} + \frac{\boldsymbol{Y}_k \boldsymbol{Y}_k^T}{\boldsymbol{Y}_k^T \boldsymbol{S}_k} \tag{3.20}$$

is used which guarantees a rank-2 approximation by providing a positive definite matrix only under the *curvature condition* that states $\boldsymbol{S}_k^T \boldsymbol{Y}_k > 0$. Curvature condition is part of the Wolfe conditions in the line search methods. However it is not necessarily ensured in our sub-problem optimization scheme in Step 3 of Algorithm 4. In practice we choose to skip the update formula (3.20) when the curvature condition is not satisfied or $\boldsymbol{S}_k^T \boldsymbol{Y}_k$ is very small, say less than 10^{-3}. For the initial value of the Hessian approximation we use

$$\hat{\mathscr{B}}_0 = \left(\frac{\boldsymbol{Y}_0^T \boldsymbol{S}_0}{\boldsymbol{Y}_0^T \boldsymbol{Y}_0} \mathscr{I} \right)^{-1}.$$

The next method we use is SR1 or the symmetric-rank-1 method that unlike BFGS does not maintain positive definiteness of the approximated hessian in the process. SR1 is useful in trust-region methodology in general and often performs better than skipping the update when the curvature condition is damaged, as is suggested in the BFGS method. The SR1 update formula is

$$\hat{\mathscr{B}}_{k+1} = \hat{\mathscr{B}}_k + \frac{\left(\boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k \right) \left(\boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k \right)^T}{\left(\boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k \right)^T \boldsymbol{S}_k}. \tag{3.21}$$

However SR1 also suggests skipping the update when $\boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k = 0$ since $\hat{\mathscr{B}}_k$ already satisfies the secant formula. Moreover, when $\boldsymbol{S}_k^T \left(\boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k \right) = 0$ but $\boldsymbol{Y}_k \neq \hat{\mathscr{B}}_k \boldsymbol{S}_k$ then a rank-2 update such as BFGS is more useful. In practice it has been recommended in [42] that one only applies (3.21) when

$$\left| \boldsymbol{S}_k^T \left(\boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k \right) \right| \geq r \left\| \boldsymbol{S}_k \right\| \left\| \boldsymbol{Y}_k - \hat{\mathscr{B}}_k \boldsymbol{S}_k \right\|,$$

for some small r such as $r = 10^{-6}$.

When iteration k is unsuccessful, that is $\boldsymbol{X}_{k+1} = \boldsymbol{X}_k$ one can recompute $\hat{\mathscr{B}}_{k+1}$ with the $\hat{\mathscr{B}}_{k_j}$, $\bar{\boldsymbol{G}}\left(\boldsymbol{X}_{k_j}, \tilde{N}_{k_j}\right)$ and $\boldsymbol{S}_{k_j}$ from most recent successful iteration k_j, wherein the latest estimate of the gradient in the current iteration is used. Albeit it is recommended in the deterministic context that the update formula (3.20) or (3.21) is performed for the failed $\boldsymbol{S}_k$ and $\boldsymbol{Y}_k$ in the unsuccessful iterations for faster convergence because the failure indicates that the approximated hessian needs improvement that is most affected by observing the change in the gradient in a direction different from the one used for the current approximated value.

In addition to choosing the Hessian approximation methods, we frugally reuse the previous observations of the function and gradient at an already visited point. At any visited point we first access all the previous Monte Carlo observations, and add to them the new incoming observations. We then update the mean and variance of the new estimate as following. Suppose that at $\boldsymbol{x}$ the function is estimated with m replications of the oracle to obtain the mean $\bar{F}(\boldsymbol{x}, m)$ and variance $\hat{\sigma}_f^2(\boldsymbol{x}, m)$. As the new observations are collected to reach the total of n replications, the measure are updated accordingly:

$$\begin{aligned}
\bar{F}(\boldsymbol{x}, n) &= n^{-1}\left(m\bar{F}(\boldsymbol{x}, m) + \sum_{j=m+1}^{n} F_j(\boldsymbol{x})\right), \\
\hat{\sigma}_f^2(\boldsymbol{x}, n) &= (n-1)^{-1}\left((m-1)\,\hat{\sigma}_f^2(\boldsymbol{x}, m) + m\bar{F}^2(\boldsymbol{x}, m) + \sum_{j=m+1}^{n} F_j^2(\boldsymbol{x}) - n\bar{F}^2(\boldsymbol{x}, n)\right).
\end{aligned} \tag{3.22}$$

Similarly we update the gradient estimate and standard error estimates of the gradient estimates with the new incoming observations at visited points:

$$\begin{aligned}
\bar{\boldsymbol{G}}(\boldsymbol{x}, n) &= n^{-1}\left(m\bar{\boldsymbol{G}}(\boldsymbol{x}, m) + \sum_{j=m+1}^{n} \boldsymbol{G}_j(\boldsymbol{x})\right), \\
\hat{\sigma}_g^2(\boldsymbol{x}, n) &= (n-1)^{-1}\left((m-1)\,\hat{\sigma}_g^2(\boldsymbol{x}, m) + m\bar{\boldsymbol{G}}^2(\boldsymbol{x}, m) + \sum_{j=m+1}^{n} \boldsymbol{G}_j^2(\boldsymbol{x}) - n\bar{\boldsymbol{G}}^2(\boldsymbol{x}, n)\right).
\end{aligned} \tag{3.23}$$

In (3.23) the notation $\bar{\boldsymbol{G}}^2(\boldsymbol{x}, n)$ means element-wise product of the vector $\bar{\boldsymbol{G}}(\boldsymbol{x}, n)$ as defined in (3.3) to itself; and the notation $\boldsymbol{G}_j^2(\boldsymbol{x})$ means element-wise product of the vector $\boldsymbol{G}_j(\boldsymbol{x})$ to itself.

Table 3.1.
Input parameters of the numerical experiment with ASTRO.

Parameter	Value
σ^2	1
η_1	0.2
η_2	0.1
η_3	0.01
γ_1	0.9
γ_2	1.1
κ_f, κ_g	1
$\Delta_{\max}$	1E+4
γ	0.5
ε	1E-3

3.6.1 Numerical Results

We experiment ASTRO in a suite of low to moderate dimensional sum of squares problems from the CUTEst problem set in [56]. Table 3.2 lists the 24 problems we use in the experiment, with their dimension and the global optimal values.

The objective function for all problems in the set takes the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} f_i^2(\boldsymbol{x}), \tag{3.24}$$

where each $f_i : \mathbb{R}^d \to \mathbb{R}$ is smooth, and most of the functions f_i are non-convex. The "noisy" observations are obtained by adding a normal random variable $\xi_i \sim \mathrm{N}\left(0, \sigma^2\right)$ to the sum, that is, $F_i(\boldsymbol{x}) = f(\boldsymbol{x}) + \xi_i$.

ASTRO was executed until a specified simulation budget is exhausted. Suppose the specified simulation budget for ASTRO is n_{total} and let $\boldsymbol{X}^i_{k_{max}}$ denotes the solution returned by the i-th execution of ASTRO on a specific problem. If ASTRO is executed m times, resulting in the m returned solutions $\boldsymbol{X}^i_{k_{max}}, i = 1, 2, \ldots, m$, the estimated expectation and

Table 3.2.
Selected problems and their global solutions from the CUTEst problem set with dimensions varying from 2 to 8.

function name	dimension	$f(\boldsymbol{x}^*)$
BEALE	3	0
BIGGS6	6	0.0057
BOX3	3	0
BROWNDEN	4	85822.2
CUBE	2	0
DENSCHNB	2	0
DENSCHNC	2	3.646
DENSCHND	3	0.0002
DENSCHNE	3	0
DENSCHNF	2	0
ENGVAL2	3	0
HATFLDD	3	0
HATFLDE	3	0
HELIX	4	0
HIMMELBF	4	318.57
KOWOSB	4	0.0003
PALMER5C	6	2.128
PALMER6C	8	0.0164
PALMER7C	8	0.602
PALMER8C	8	0.1598
ROSENBR	2	0
S308	2	0
SINEVAL	2	0
YFITU	3	0

estimated square-root variance of the *true* optimality gap of ASTRO's returned solution are given by

$$\hat{\mathbb{E}}[f(\boldsymbol{X}^i_{k_{max}}) - f(\boldsymbol{x}^*)] := m^{-1}\sum_{j=1}^{m} f(\boldsymbol{X}^j_{k_{max}}) - f(\boldsymbol{x}^*);$$
$$\sqrt{\hat{\mathbb{V}}(f(\boldsymbol{X}^j_{k_{max}}) - f(\boldsymbol{x}^*))} := \sqrt{(m-1)^{-1}\sum_{j=1}^{m}(f(\boldsymbol{X}^j_{k_{max}}) - f(\boldsymbol{x}^*))^2}, \tag{3.25}$$

where $f(\boldsymbol{x}^*)$ is the known minimum value attained by the function f. Each row in Table 3.3 corresponds to a specific problem in CUTEst and reports $\hat{\mathbb{E}}[f(\boldsymbol{X}^i_{k_{max}}) - f(\boldsymbol{x}^*)]$ and $\sqrt{\hat{\mathbb{V}}(f(\boldsymbol{X}^j_{k_{max}}) - f(\boldsymbol{x}^*))}$ (in parenthesis) for $m = 20$ independent executions of ASTRO. A calculation similar to (3.25) for true gradient norms is

$$\hat{\mathbb{E}}[\|\nabla f(\boldsymbol{X}^i_{k_{max}})\|] := m^{-1}\sum_{j=1}^{m} \|\nabla f(\boldsymbol{X}^i_{k_{max}})\|;$$
$$\sqrt{\hat{\mathbb{V}}(\|\nabla f(\boldsymbol{X}^i_{k_{max}})\|)} := \sqrt{(m-1)^{-1}\sum_{j=1}^{m}\|\nabla f(\boldsymbol{X}^i_{k_{max}})\|^2}. \tag{3.26}$$

It is important to note that since the convergence theory for ASTRO only guarantees convergence to a stationary point, nothing can be said about the behavior of the true optimality gap even as the budget tends to infinity.

We use several random initial sets $(\boldsymbol{x}_0, \Delta_0)$ in the pre-processing of ASTRO and terminate ASTRO after 100 oracle calls. Then we choose the initial set that obtains the smallest estimated gradient norm. We also use the expansion and contraction coefficients as the reverse of each other. Since for some problems and some starting points more significant contraction and expansion accelerates the progress and convergence rate we also test several values for γ_1 including 0.9 and 0.99 in the pre-processing step. The rest of the input parameters are chosen according to Table 3.1 for the purpose of this chapter. Once the initial values are selected in the pre-processing, we run ASTRO with the selected initial set until the simulation budget of $20,000$ oracle calls is exhausted. The estimated expectation and square-root variance of the optimality gap and true gradient norm, as explained above, are recorded at checkpoints of $500, 1K, 5K, 10K$ and $20K$ simulation budget.

Tables 3.3 and 3.4 provide the results after executing ASTRO to termination 20 times for each problem. As observed in the results the decline in the optimality gap is quite

Table 3.3.
The estimated mean and standard deviation of the true optimality gap at a (random) returned solution of ASTRO, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO on each problem.

d	Problem Name	Initial Function Value	Simulation Budget				
			500	1000	5000	10000	20000
2	CUBE	1,664,640,225.00	2.32 (0.00)	2.32 (0.00)	2.32 (0.00)	2.32 (0.00)	2.32 (0.00)
	DENSCHNB	50,661.00	0.11 (0.35)	0.01 (0.02)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	DENSCHNC	10,713,258,316,706.26	3.59 (0.02)	3.60 (0.02)	3.61 (0.02)	3.62 (0.02)	3.62 (0.02)
	DENSCHNF	6,825,024.00	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	ROSENBR	7,398,689.00	0.11 (0.21)	0.11 (0.21)	0.10 (0.19)	0.10 (0.19)	0.10 (0.17)
	S308	589,825.00	0.94 (0.09)	0.93 (0.09)	0.84 (0.09)	0.81 (0.08)	0.79 (0.05)
	SINEVAL	265,359.79	29.46 (0.49)	29.31 (0.57)	29.11 (0.69)	28.97 (0.80)	28.80 (1.00)
3	BEALE	4,309,937,474.20	0.31 (0.00)	0.31 (0.00)	0.31 (0.00)	0.31 (0.00)	0.31 (0.00)
	DENSCHND	4,880,138,240.00	0.16 (0.04)	0.14 (0.04)	0.10 (0.04)	0.09 (0.04)	0.08 (0.04)
	DENSCHNE	57,857.00	1.03 (0.02)	1.02 (0.02)	1.01 (0.02)	1.01 (0.02)	1.01 (0.01)
	ENGVAL2	83,047,445.00	22.28 (12.54)	18.76 (9.36)	12.55 (4.90)	10.42 (3.57)	8.56 (2.61)
	YFITU	7,532.36	548.51 (224.99)	517.9 (193.6)	403.02 (25.46)	391.43 (3.90)	389.44 (1.11)
4	BROWNDEN	1,109,286,386.27	360.29 (16.76)	274.88 (11.68)	119.59 (4.69)	77.09 (3.26)	45.03 (1.88)
	HELIX	62,036.77	0.89 (0.04)	0.87 (0.05)	0.83 (0.07)	0.81 (0.07)	0.79 (0.08)
	HIMMELBF	18,223,594.79	17,856.9 (23.97)	17,637.33 (2.37)	16,975.15 (13.72)	16,645.96 (3.82)	16,268.06 (3.96)
	KOWOSB	373.13	0.35 (0.15)	0.30 (0.16)	0.16 (0.11)	0.11 (0.05)	0.09 (0.03)
6	BIGGS6	11.40	8.29 (0.39)	8.08 (0.47)	7.07 (1.81)	6.07 (2.09)	4.04 (2.43)
	PALMER5C	17,604.47	0.14 (0.07)	0.07 (0.04)	0.01 (0.01)	0.01 (0.01)	0.01 (0.01)
8	PALMER6C	32,357,294.58	612.58 (70.12)	611.44 (71.32)	608.52 (73.46)	607.63 (73.18)	606.61 (72.85)
	PALMER7C	130,070,963.92	945.57 (24.51)	936.92 (25.11)	913.53 (25.7)	903.01 (25.69)	890.27 (25.34)
	PALMER8C	37,881,644.22	665.94 (27.51)	659.92 (28.30)	642.23 (29.48)	633.66 (29.75)	623.06 (29.65)

remarkable. Table 3.3 shows how fast the optimality gap declines in the first few hundred oracle calls. In the later checkpoints closer to the ultimate simulation budget ($20K$) the decline becomes slower as the iteration size has increased and the trust-region radius has decreased leading to large sample size for the later iterations. In Figures B.1-B.8 we show the values after the completion of 200 oracle calls, as the first few values are pretty large and make visibility of the plots less convenient. In almost all problems the iterates approach the optimal solution within 20K simulation calls, though in some instances the iterates reach a local minima that is different from the global minima reported in Table 3.2 and hence the

Table 3.4.
The estimated mean and standard deviation of the true gradient norm at a (random) returned solution of ASTRO, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO on each problem.

d	Problem Name	Initial Gradient Norm	Simulation Budget				
			500	1000	5000	10000	20000
2	CUBE	626,688,560.79	2.23 (2.65)	1.23 (1.98)	1.02 (1.64)	0.94 (1.65)	0.66 (1.21)
	DENSCHNB	9,568.07	0.48 (0.87)	0.21 (0.20)	0.07 (0.06)	0.06 (0.05)	0.05 (0.06)
	DENSCHNC	21,399,716,500,504.68	0.82 (0.59)	0.67 (0.41)	0.51 (0.32)	0.46 (0.29)	0.40 (0.18)
	DENSCHNF	1,228,253.22	0.72 (0.67)	0.64 (0.93)	0.27 (0.27)	0.24 (0.28)	0.19 (0.26)
	ROSENBR	1,741,683.78	0.66 (0.77)	0.61 (0.79)	0.51 9(0.67)	0.52 (0.56)	0.51 (0.59)
	S308	104,267.14	0.28 (0.10)	0.25 (0.11)	0.20 (0.14)	0.14 (0.15)	0.13 (0.18)
	SINEVAL	45,109.93	10.63 (4.66)	9.48 (4.66)	8.04 (2.51)	7.56 (2.40)	8.33 (2.96)
3	BEALE	1,701,980,751.18	0.14 (0.08)	0.15 (0.14)	0.10 (0.15)	0.07 (0.09)	0.07 (0.09)
	DENSCHND	2,284,598,497.73	0.98 (0.46)	0.89 (0.32)	0.58 (0.27)	0.57 (0.26)	0.54 (0.24)
	DENSCHNE	14,880.03	0.22 (0.08)	0.17 (0.06)	0.12 (0.06)	0.10 (0.07)	0.05 (0.06)
	ENGVAL2	16,721,054.23	66.10 (25.99)	56.72 (21.90)	41.89 (12.51)	36.26 (9.34)	31.42 (7.35)
	YFITU	6,698.93	962.7 (691.7)	706.01 (642.19)	551.29 (910.24)	152.10 (144.00)	97.89 (31.40)
4	BROWNDEN	144,497,890.99	1,354.0 (252.43)	1,110.67 (35.31)	804.76 (17.65)	638.7 (13.44)	484.0 (10.15)
	HELIX	4,989.97	3.23 (1.80)	3.41 (1.93)	2.77 (0.79)	2.83 (1.17)	2.30 (0.56)
	HIMMELBF	1,207,473.61	1305.43 (162.35)	1,101.21 (2.64)	1291.13 (70.88)	1,078.23 (166.04)	1,035.68 (3.32)
	KOWOSB	80.65	0.13 (0.07)	0.11 (0.04)	0.07 (0.03)	0.05 (0.02)	0.04 (0.01)
6	BIGGS6	2.25	0.28 (0.09)	0.27 (0.08)	0.39 (0.25)	0.55 (0.48)	0.67 (0.43)
	PALMER5C	839.76	1.4 (0.48)	0.95 (0.28)	0.44 (0.18)	0.35 (0.15)	0.28 (0.15)
8	PALMER6C	7,070,831.09	476.08 (955.39)	380.75 (248.45)	335.89 (204.67)	246.96 (158.28)	319.27 (176.14)
	PALMER7C	29,768,118.46	1,506.94 (65.19)	2,027.34 (914.05)	1,649.61 (479.29)	1,925.19 (650.81)	2,496.61 (503.7)
	PALMER8C	8,275,792.01	897.48 (164.43)	1,011.43 (273.6)	900.44 (171.73)	1.035.14 (223.91)	1,138.63 (161.72)

optimality gap converges to a non-zero value. The true gradient norm results reported in Table 3.4 also show consistency as in most cases they drop to zero quite fast in the search. However in some problem instances despite the decrease in the optimality gap the gradient norm increases. These situations occur near the cliff-like regions of the objective functions. Furthermore, the $25\%, 50\%, 75\%$ and 90% quantiles of the standard deviation of the true gradient norm in 20 independent runs are illustrated in the Appendix (Figures B.1-B.8).

4. ASTRO-DF: ADAPTIVE SAMPLING TRUST-REGION OPTIMIZATION — DERIVATIVE-FREE

Recall again that the primary contribution of this dissertation is developing algorithmic frameworks for unconstrained continuous simulation optimization both in the presence and absence of unbiased (Monte Carlo) estimates of the gradient of the objective function. The previous chapter outlined a family of algorithms for the derivative-based context. In this chapter, we present a family of algorithms for the derivative-free context, where no unbiased estimates of the objective function's gradient are assumed to be available. Accordingly, the family of algorithms we propose (ASTRO-DF) relies on constructed stochastic interpolation models of the objective function to generate iterates that globally converge to a critical point with probability one.

In what follows we first describe related work on stochastic TRO in the area of derivative-free optimization and the definitions used in the derivative-free stochastic framework. We then describe ASTRO-DF and its behavior in detail.

4.1 Preliminaries

We will look at the relevant state-of-the-art in the derivative-free trust region methods in the stochastic context. We also list the common definitions used throughout this chapter.

4.1.1 Related Work

The algorithm proposed in Deng and Ferris [57, 58], called VNSP, is a competitor to what we present here and has several aspects in common. For example, Deng and Ferris [57, 58] use a quadratic interpolation model within a trust-region optimization framework. The model is derivative-free in the sense that only function estimates are assumed

to be available. Model construction, inference, and improvement, along with sample size updates happen through a Bayesian framework with an assumed Gaussian conjugate prior. Convergence theory for VNSP is accordingly within a Bayesian setting.

The other two algorithms that are particularly noteworthy competitors to what we propose here are STORM [59] and the recently proposed algorithm by Larson and Billups [60] (henceforth LB2014). While the underlying logic in both of these algorithms are very similar to that in ASTRO-DF, key differences arise in terms of what has been assumed about the quality of the constructed models and how such quality can be achieved in practice. A key postulate that guarantees consistency in STORM, for example, is that the constructed models are of a certain specified quality (characterized through the notion of probabilistic full linearity) with a probability exceeding a fixed threshold. The authors provide a way to construct such models using function estimates constructed as sample means. Crucially, the prescribed sample means in STORM use a sample size that is derived using the Chebyshev inequality with an assumed upper bound on the variance. By contrast, the sample sizes in ASTRO-DF are determined adaptively by balancing squared bias and variance estimates for the function estimator. While this makes the sample size in ASTRO-DF a stopping time [61] thereby complicating proofs, such adaptive sampling enables ASTRO-DF to differentially sample across the search space, leading to efficiency.

LB2014, like STORM, uses random models. Unlike STORM, however, the sequence of models constructed in LB2014 are assumed to be accurate (as measured by a certain rigorous notion) with a probability sequence that converges to one. A related version [62] of LB2014 address the issue of differing levels of (spatial) stochastic error through the use of weighted regression schemes, where the weights are chosen heuristically. An important assumption that facilitates convergence guarantees in [62] is that the stochastic function error can be bounded deterministically. In other words, if $f_n(\boldsymbol{x})$ represents the Monte Carlo function estimate of $f(\boldsymbol{x})$ at the point $\boldsymbol{x}$, then Monte Carlo sampling guarantees that $\|f_n(\boldsymbol{x}) - f(\boldsymbol{x})\| \leq \varepsilon$ for any given ε. As noted earlier, we make no such assumptions on the ability to bound the stochastic error in such a deterministic fashion.

4.1.2 Definitions

In this section we provide definitions of stochastic interpolation models that will be used throughout this chapter.

Definition 10. *(Stochastic Linear Interpolation Models) $M(\boldsymbol{z})$ is said to be a stochastic linear interpolation model of f on $\mathcal{B}(\boldsymbol{x};\Delta)$, if $p = d+1$ and $\Phi(\boldsymbol{z}) = \left(1, z^1, z^2, \ldots, z^d\right)$ is a linear basis on $\mathbb{X} \subseteq \mathbb{R}^d$. We set*

$$\begin{bmatrix} 1 & y_1^1 & y_1^2 & \ldots & y_1^d \\ 1 & y_2^1 & y_2^2 & \ldots & y_2^d \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & y_{d+1}^1 & y_{d+1}^2 & \ldots & y_{d+1}^d \end{bmatrix} \begin{bmatrix} \hat{\alpha}^1 \\ \hat{\alpha}^2 \\ \vdots \\ \hat{\alpha}^{d+1} \end{bmatrix} = \begin{bmatrix} \bar{F}(\boldsymbol{y}_1, n(\boldsymbol{y}_1)) \\ \bar{F}(\boldsymbol{y}_2, n(\boldsymbol{y}_2)) \\ \vdots \\ \bar{F}(\boldsymbol{y}_{d+1}, n(\boldsymbol{y}_{d+1})) \end{bmatrix}.$$

For all $\boldsymbol{s} \in \mathcal{B}(\boldsymbol{0};\Delta)$, $M(\boldsymbol{x}+\boldsymbol{s}) = \hat{\alpha}^1 + (\boldsymbol{x}+\boldsymbol{s})^T \boldsymbol{G} = M(\boldsymbol{x}) + \boldsymbol{s}^T \boldsymbol{G}$ where $\boldsymbol{G} = \left(\hat{\alpha}^2, \ldots, \hat{\alpha}^{d+1}\right)^T$. The gradient $\nabla M(\boldsymbol{x})$ of the stochastic linear interpolation model is $\nabla M(\boldsymbol{x}+\boldsymbol{s}) = \boldsymbol{G}$. The Hessian $\nabla^2 M(\boldsymbol{x}+\boldsymbol{s})$ of the stochastic linear interpolation model is $\boldsymbol{0}$.

Definition 11. *(Stochastic Quadratic Interpolation Models) $M(\boldsymbol{z})$ is said to be a stochastic quadratic interpolation model of f on $\mathcal{B}(\boldsymbol{x};\Delta)$, if $p = (d+1)(d+2)/2$ and $\Phi(\boldsymbol{z}) = \left(1, z^1, z^2, \ldots, z^d, \frac{1}{2}(z^1)^2, z^1 z^2, \ldots, \frac{1}{2}(z^d)^2\right)$ is a quadratic basis on $\mathbb{X} \subseteq \mathbb{R}^d$. We set*

$$\begin{bmatrix} 1 & y_1^1 & \ldots & y_1^d & \frac{1}{2}\left(y_1^1\right)^2 & y_1^1 y_1^2 & \ldots & \frac{1}{2}\left(y_1^d\right)^2 \\ 1 & y_2^1 & \ldots & y_2^d & \frac{1}{2}\left(y_2^1\right)^2 & y_2^1 y_2^2 & \ldots & \frac{1}{2}\left(y_2^d\right)^2 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & y_p^1 & \ldots & y_p^d & \frac{1}{2}\left(y_p^1\right)^2 & y_p^1 y_p^2 & \ldots & \frac{1}{2}\left(y_p^d\right)^2 \end{bmatrix} \begin{bmatrix} \hat{\alpha}^1 \\ \hat{\alpha}^2 \\ \vdots \\ \hat{\alpha}^p \end{bmatrix} = \begin{bmatrix} \bar{F}(\boldsymbol{y}_1, n(\boldsymbol{y}_1)) \\ \bar{F}(\boldsymbol{y}_2, n(\boldsymbol{y}_2)) \\ \vdots \\ \bar{F}\left(\boldsymbol{y}_p, n\left(\boldsymbol{y}_p\right)\right) \end{bmatrix}.$$

Then for all $\boldsymbol{s} \in \mathcal{B}(\boldsymbol{0};\Delta)$,

$$M(\boldsymbol{x}+\boldsymbol{s}) = M(\boldsymbol{x}) + \boldsymbol{s}^T \boldsymbol{G} + \frac{1}{2}\boldsymbol{s}^T \hat{\mathscr{B}} \boldsymbol{s},$$

where $\boldsymbol{G} = \left(\hat{\alpha}^2, \ldots, \hat{\alpha}^{d+1}\right)$ and $\hat{\mathscr{B}}$ is a symmetric (positive-definite) matrix. The gradient $\nabla M(\boldsymbol{x}+\boldsymbol{s})$ of the quadratic interpolation model is $\nabla M(\boldsymbol{x}+\boldsymbol{s}) = \boldsymbol{G} + \boldsymbol{s}^T \hat{\mathscr{B}}$ and hence $\nabla M(\boldsymbol{x}) = \boldsymbol{G}$. The Hessian $\nabla^2 M(\boldsymbol{x}+\boldsymbol{s})$ of the quadratic interpolation model is $\nabla^2 M(\boldsymbol{x}+\boldsymbol{s}) = \hat{\mathscr{B}}$.

4.2 Algorithm Listing

ASTRO-DF is an adaptive sampling trust-region derivative-free algorithm whose essence is encapsulated within four repeating steps: (i) local stochastic model construction and certification through adaptive sampling; (ii) constrained optimization of the constructed model for identifying the next candidate solution; (iii) re-estimation of the next candidate solution through adaptive sampling; and (iv) iterate and trust-region update based on a (stochastic) sufficient decrease check. These stages appear as Steps 1-4 in Algorithm 5. In what follows, we describe each of these steps in further detail.

In Step 2 of Algorithm 5, a stochastic model of the function $f(\cdot)$ in the trust-region $\mathcal{B}(\boldsymbol{X}_k;\Delta_k)$ is constructed using Algorithm 6. The aim of Algorithm 6 is to construct a model of a specified quality within a trust-region having radius smaller than a fixed multiple of the model gradient norm. During the j_kth iteration of Algorithm 6, a poised set $\mathcal{Y}_k^{(j_k)} \triangleq \{\boldsymbol{Y}_1^{(j_k)}, \boldsymbol{Y}_2^{(j_k)}, \ldots, \boldsymbol{Y}_p^{(j_k)}\}$ in the "candidate" trust region having radius $\tilde{\Delta}_k w^{j_k-1}$ and center $\boldsymbol{Y}_1^{(j_k)} = \boldsymbol{X}_k$ is chosen (Step 3); Monte Carlo function estimates are then obtained at each of the points in $\mathcal{Y}_k^{(j_k)}$ with $N\left(\boldsymbol{Y}_i^{(j_k)}\right)$ being the sample size at point $\boldsymbol{Y}_i^{(j_k)}$ after the j_kth iteration of the contraction loop. Sampling at each point in $\mathcal{Y}_k^{(j_k)}$ is adaptive and continues (Steps 4–6) until the estimated standard errors $\hat{\sigma}_F\left(\boldsymbol{Y}_i^{(j_k)}, N\left(\boldsymbol{Y}_i^{(j_k)}\right)\right) / \sqrt{N\left(\boldsymbol{Y}_i^{(j_k)}\right)}$ of the function estimates $\bar{F}\left(\boldsymbol{Y}_i^{(j_k)}, N\left(\boldsymbol{Y}_i^{(j_k)}\right)\right)$ drop below a slightly inflated square of the candidate trust-region radius. A linear (or quadratic) interpolation model is then constructed using the obtained function estimates in Step 5. (If a linear interpolation model is constructed, $p = d+1$, and if a quadratic interpolation model is constructed, $p = (d+1)(d+2)/2$.) If the resulting model $M_k^{(j_k)}(\boldsymbol{z}), \boldsymbol{z} \in \mathcal{B}(\boldsymbol{X}_k;\tilde{\Delta}_k w^{j_k-1})$ is such that the candidate trust-region radius $\tilde{\Delta}_k w^{j_k-1}$ is too large compared to the norm of the model gradient $\left\|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\right\|$, that is, if $\tilde{\Delta}_k w^{j_k-1} > \mu \left\|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\right\|$, then the candidate trust region radius is shrunk by a factor w and control is returned back to Step 3. On the other hand, if the candidate trust region radius is smaller than the product of μ and the norm of the model gradient, then the resulting stochastic model is accepted but over an updated incumbent trust-region radius

Algorithm 5 ASTRO-DF Main Algorithm

Require: Initial guess $\boldsymbol{x}_0 \in \mathbf{R}^d$, initial trust-region radius $\tilde{\Delta}_0 > 0$ and maximum radius $\Delta_{\max} > 0$, model "fitness" threshold $\eta_1 > 0$, trust-region expansion constant $\gamma_1 > 1$ and contraction constant $\gamma_2 \in (0,1)$, initial sample size n_0, sample size lower bound sequence $\{\lambda_k\}$ such that $k^{(1+\varepsilon)} = \mathcal{O}(\lambda_k)$, initial sample set $\tilde{\mathcal{Y}}_0 = \{\boldsymbol{x}_0\}$, and outer adaptive sampling constant κ_{oas}.

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: *Model Construction:* Construct the model at $\boldsymbol{X}_k$ by calling Algorithm 6 with the candidate trust-region radius $\tilde{\Delta}_k$ and candidate set of sample points $\tilde{\mathcal{Y}}_k$, $[M_k(\boldsymbol{X}_k + \boldsymbol{s}), \Delta_k, \mathcal{Y}_k] =$ `AdaptiveModelConstruction`$(\tilde{\Delta}_k, \tilde{\mathcal{Y}}_k)$. Set $\widetilde{N}_k = N(\boldsymbol{X}_k)$.

3: *TR Subproblem:* Approximate the kth step by minimizing the model in the trust-region, $\boldsymbol{S}_k = \operatorname{argmin}_{\|\boldsymbol{s}\| \le \Delta_k} M_k(\boldsymbol{X}_k + \boldsymbol{s})$, and set the new candidate point $\tilde{\boldsymbol{X}}_{k+1} = \boldsymbol{X}_k + \boldsymbol{S}_k$.

4: *Evaluate:* Estimate the function at the candidate point using adaptive sampling to obtain $\bar{F}(\tilde{\boldsymbol{X}}_{k+1}, \widetilde{N}_{k+1})$, where

$$\widetilde{N}_{k+1} = \max\left\{\lambda_k, \min\left\{n : \frac{\hat{\sigma}_F\left(\tilde{\boldsymbol{X}}_{k+1}, n\right)}{\sqrt{n}} \le \frac{\kappa_{oas}\Delta_k^2}{\sqrt{\lambda_k}}\right\}\right\}, \tag{4.1}$$

Update:

5: Compute the success ratio $\hat{\rho}_k$ as

$$\hat{\rho}_k = \frac{\bar{F}\left(\boldsymbol{X}_k, \widetilde{N}_k\right) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \widetilde{N}_{k+1}\right)}{M_k(\boldsymbol{X}_k) - M_k(\tilde{\boldsymbol{X}}_{k+1})}.$$

6: **if** $\hat{\rho}_k > \eta_1$ **then**

7: $\boldsymbol{X}_{k+1} = \tilde{\boldsymbol{X}}_{k+1}$, $\tilde{\Delta}_{k+1} = \min\{\gamma_1\Delta_k, \Delta_{\max}\}$, $N_{k+1} = \widetilde{N}_{k+1}$. Set $\tilde{\boldsymbol{Y}}_{\max} := \arg\max_{\boldsymbol{Y}_i \in \mathcal{Y}_k}\left\{\left\|\tilde{\boldsymbol{X}}_{k+1} - \boldsymbol{Y}_i\right\|\right\}$. Update the sample set $\tilde{\mathcal{Y}}_{k+1} = \mathcal{Y}_k \setminus \left\{\tilde{\boldsymbol{Y}}_{\max}\right\} \cup \{\boldsymbol{X}_{k+1}\}$.

8: **else**

9: $\boldsymbol{X}_{k+1} = \boldsymbol{X}_k$, $\tilde{\Delta}_{k+1} = \gamma_2\Delta_k$, $N_{k+1} = \widetilde{N}_k$. Set $\boldsymbol{Y}_{max} := \arg\max_{\boldsymbol{Y}_i \in \mathcal{Y}_k}\{\|\boldsymbol{X}_k - \boldsymbol{Y}_i\|\}$. If $\tilde{\boldsymbol{X}}_{k+1} \neq \boldsymbol{Y}_{max}$, update $\tilde{\mathcal{Y}}_{k+1} = \mathcal{Y}_k \setminus \{\boldsymbol{Y}_{max}\} \cup \left\{\tilde{\boldsymbol{X}}_{k+1}\right\}$.

10: **end if**

11: **end for**

given by Step 11. (Step 11 of Algorithm 6, akin to [46], updates the incumbent trust-region radius to the point in the interval $[\tilde{\Delta}_k w^{j_k - 1}, \tilde{\Delta}_k]$ that is closest to $\beta\left\|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\right\|$).

We emphasize the following three issues pertaining to the model resulting from the application of Algorithm 6.

Algorithm 6 $[M_k(\boldsymbol{X}_k+\boldsymbol{s}),\Delta_k,\mathcal{Y}_k]$=`AdaptiveModelConstruction`$(\tilde{\Delta}_k,\tilde{\mathcal{Y}}_k)$

Require: *Parameters from ASTRO-DF:* candidate trust-region radius $\tilde{\Delta}_k$ and candidate sample set $\tilde{\mathcal{Y}}_k$ (possibly with cardinality $< p$).

Parameters specific to `AdaptiveModelConstruction`: trust-region contraction factor $w \in (0,1)$, trust-region and gradient balance constant μ, gradient inflation constant β with $0 < \beta < \mu$, and inner adaptive sampling constant κ_{ias}.

1: Initialize $j_k = 1$, set $\mathcal{Y}_k^{(j_k)} = \tilde{\mathcal{Y}}_k$, and set $\boldsymbol{Y}_1 = \boldsymbol{X}_k$ where $\boldsymbol{X}_k$ is the first element of $\tilde{\mathcal{Y}}_k$.

Contraction loop:

2: **repeat**

3: Improve $\mathcal{Y}_k^{(j_k)} = \left\{\boldsymbol{Y}_1^{(j_k)}, \boldsymbol{Y}_2^{(j_k)}, \ldots, \boldsymbol{Y}_p^{(j_k)}\right\}$ by appropriately choosing $\boldsymbol{Y}_i^{(j_k)},\ i = 2,3,\cdots,p$ to make it a poised set in $\mathcal{B}(\boldsymbol{X}_k; \tilde{\Delta}_k w^{j_k-1})$.

4: **for** $i = 1$ to p **do**

5: Estimate $\bar{F}\left(\boldsymbol{Y}_i^{(j_k)}, N\left(\boldsymbol{Y}_i^{(j_k)}\right)\right)$, where

$$N\left(\boldsymbol{Y}_i^{(j_k)}\right) = \max\left\{\lambda_k, \min\left\{n : \frac{\hat{\sigma}_F\left(\boldsymbol{Y}_i, n\right)}{\sqrt{n}} \leq \frac{\kappa_{ias}(\tilde{\Delta}_k w^{j_k-1})^2}{\sqrt{\lambda_k}}\right\}\right\}. \tag{4.2}$$

6: **end for**

7: Construct a quadratic model $M_k^{(j_k)}(\boldsymbol{X}_k+\boldsymbol{s})$ via interpolation.

8: Set $j_k = j_k + 1$.

9: **until** $\tilde{\Delta}_k w^{j_k-1} \leq \mu \|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\|$.

10: Set $M_k(\boldsymbol{X}_k+\boldsymbol{s}) = M_k^{(j_k)}(\boldsymbol{X}_k+\boldsymbol{s})$, $\nabla M_k(\boldsymbol{X}_k) = \nabla M_k^{(j_k)}(\boldsymbol{X}_k)$, and $\nabla^2 M_k(\boldsymbol{X}_k) = \nabla^2 M_k^{(j_k)}(\boldsymbol{X}_k)$.

11: **return** $M_k(\boldsymbol{X}_k+\boldsymbol{s})$, $\Delta_k = \min\left\{\tilde{\Delta}_k, \max\left\{\beta\|\nabla M_k(\boldsymbol{X}_k)\|, \tilde{\Delta}_k w^{j_k-1}\right\}\right\}$, and $\mathcal{Y}_k = \mathcal{Y}_k^{(j_k)}$.

(i) Due to the nature of the chosen poised set $\mathcal{Y}_k$, the (hypothetical) limiting model $m_k(\boldsymbol{X}_k)$ constructed from true function observations on $\mathcal{Y}_k$ will be $(\kappa_{ef}, \kappa_{eg})$-fully-linear (or $(\kappa_{ef}, \kappa_{eg}, \kappa_{eh})$-fully-quadratic) on the updated trust region $\mathcal{B}(\boldsymbol{X}_k; \Delta_k)$. Of course, the model $m_k(\boldsymbol{X}_k)$ is unavailable since true function evaluations are unavailable; and it makes no sense to talk about whether the constructed model $M_k(\boldsymbol{X}_k)$ is $(\kappa_{ef}, \kappa_{eg})$-fully-linear (or fully quadratic) since it is constructed from stochastic function estimates.

(ii) By construction, the trust region resulting from the application of Algorithm 6 has a radius that is at most β times the model gradient norm $\|\nabla M_k(\boldsymbol{X}_k)\|$.

(iii) The structure of adaptive sampling in Step 5 of Algorithm 6 is identical to that appearing for estimation in Step 4 of Algorithm 5. The adaptive sampling step simply involves sampling until the estimated standard error of the function estimate comes within a factor of the deflated square of the incumbent trust-region radius. As our convergence proofs will reveal, balancing the estimated standard error to any lower power of the incumbent trust-region radius will threaten consistency of ASTRO-DF's iterates.

Let us now resume our discussion of Algorithm 5. In Step 2, Algorithm 5 executes `AdaptiveModelConstruction` to obtain a model $M_k(\boldsymbol{z}), \boldsymbol{z} \in \mathcal{B}(\boldsymbol{X}_k; \Delta_k)$ whose limiting approximation is $(\kappa_{ef}, \kappa_{eg})$-fully-linear (or $(\kappa_{ef}, \kappa_{eg}, \kappa_{eh})$-fully-quadratic) as observed in (i). Step 3 in Algorithm 5 then approximately solves the constrained optimization problem $\boldsymbol{S}_k = \arg\min_{\|\boldsymbol{s}\| \leq \Delta_k} M_k(\boldsymbol{X}_k + \boldsymbol{s})$ to obtain a candidate point $\tilde{\boldsymbol{X}}_{k+1} = \boldsymbol{X}_k + \boldsymbol{S}_k$ satisfying the κ_{fcd}-Cauchy decrease as defined in Assumption 6.

In preparation for checking if the candidate solution $\tilde{\boldsymbol{X}}_{k+1}$ provides sufficient decrease, Step 4 of Algorithm 5 obtains Monte Carlo samples of the objective function at $\tilde{\boldsymbol{X}}_{k+1}$, until the estimated standard error $\hat{\sigma}_F\left(\tilde{\boldsymbol{X}}_{k+1}, \widetilde{N}_{k+1}\right) / \sqrt{\widetilde{N}_{k+1}}$ of $\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \widetilde{N}_{k+1}\right)$ is smaller than a slightly deflated square of the trust-region radius $\lambda_k^{-1/2} \kappa_{oas} \Delta_k^2$, subject to the sample size being at least as big as λ_k (see Remark 5).

In Step 5 of Algorithm 5, the obtained function estimate is used to check if the ratio $\hat{\rho}_k$ of the predicted to the observed function decrease at the point $\tilde{\boldsymbol{X}}_{k+1}$ exceeds a fixed threshold η_1. If $\hat{\rho}_k$ exceeds the threshold η_1, the candidate $\tilde{\boldsymbol{X}}_{k+1}$ is accepted as the new iterate $\boldsymbol{X}_{k+1}$, the iteration is deemed successful, and the trust-region is expanded (Step 6). If $\hat{\rho}_k$ falls below the specified threshold η_1, the candidate $\tilde{\boldsymbol{X}}_{k+1}$ is rejected (though it may remain in the sample set), the iteration is deemed unsuccessful, and the trust-region is shrunk (Step 9). In either case, $\tilde{\Delta}_{k+1}$ is set as the incumbent trust-region radius, N_{k+1} is set as the current sample size of $\boldsymbol{X}_{k+1}$, and $\mathcal{Y}_k$ is set as the interpolation set for the next iteration. Note that in the next iteration the sample size of $\boldsymbol{X}_{k+1}$ is subject to change through Step 2 again.

Remark 5. *The sequence $\{\lambda_k\}$ appearing as the first argument of the "max" function in the expression for the adaptive sample size (in Step 4 of Algorithm 5 and Step 5 of Algorithm 6) is standard for all adaptive sampling contexts, e.g., [35, 36], and intended to nullify the effects of mischance without explicitly participating in the limit. It will become evident through our proofs that the probability of the first argument in the expression for the adaptive sample size being binding will decay to zero as k becomes large.*

Lastly and similar to the analysis in Chapter 3, an important observation from the Algorithms 5 and 6 is that the difference between the function estimates of two consecutive iterates can be increasing. When iteration k is unsuccessful, that is $\boldsymbol{X}_k = \boldsymbol{X}_{k+1}$, it is possible that $\bar{F}(\boldsymbol{X}_k, N_k) < \bar{F}(\boldsymbol{X}_{k+1}, N_{k+1})$. When iteration k is successful, we know from Step 6 that $\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) > \bar{F}(\boldsymbol{X}_{k+1}, N_{k+1})$ must be true but it is still possible that $\bar{F}(\boldsymbol{X}_k, N_k) < \bar{F}(\boldsymbol{X}_{k+1}, N_{k+1})$ since $\bar{F}(\boldsymbol{X}_k, N_k) \neq \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right)$. This observation will later be used in the convergence analysis.

4.3 Convergence Results

The convergence behavior of ASTRO-DF depends crucially on the behavior of three error terms: (i) *stochastic sampling error* arising due to the fact that function evaluations are through Monte Carlo simulation; (ii) *model bias* arising due to the choice of local model; and (iii) *stochastic interpolation error* arising due to the fact that model prediction at unobserved points is a combination of the model bias and the error in (i). (The analysis in the deterministic context involves only the error in (ii).) Accordingly, driving the errors in (i) and (ii) to zero sufficiently fast, while ensuring the fully linear or quadratic sufficiency of the expected model, guarantees almost sure convergence.

Driving the errors in (i) and (ii) to zero sufficiently fast is accomplished by forcing the sample sizes to increase across iterations at a sufficiently fast rate, something that we ensure by keeping the estimated standard error of all function estimates in lock step with the square of the trust-region radius. The trust-region radius is in turn also kept in lock-step with the model gradient through the model construction Algorithm 6. Such a deliberate

lock-step between the model error, trust-region radius, and the model gradient is aimed at efficiency without sacrificing consistency.

In what follows, we provide a formal proof of the with probability one convergence of ASTRO-DF's iterates. Recall that we assume that the models being constructed within Step 2 of Algorithm 5 are either linear or quadratic interpolation models. Furthermore, we focus only on convergence to a first-order critical point of the function f.

Throughout the following sections Assumption 5 holds for the ensuing theoretical proofs. Assumption 5 enforces a uniform bound on the model Hessian over the whole feasible region and can be ensured by a check that is executed each time the model is constructed or updated. The next assumption is the equivalent of Definition 8 for the SO context.

Assumption 6. *(Cauchy Reduction) The minimizer obtained in the trust-region sub-problem (Step 3 of Algorithm 5) satisfies a κ_{fcd}-Cauchy decrease with $\kappa_{fcd} > 0$, that is,*

$$M_k(\boldsymbol{X}_k) - M_k(\boldsymbol{X}_k + \boldsymbol{S}_k) \geq \frac{\kappa_{fcd}}{2} \|\nabla M_k(\boldsymbol{X}_k)\| \min\left\{ \frac{\|\nabla M_k(\boldsymbol{X}_k)\|}{\|\nabla^2 M_k(\boldsymbol{X}_k)\|}, \Delta_k \right\}.$$

Similar to Definition 8 in the deterministic TRO-DF framework, some fraction of Cauchy decrease establishes the link between trust-region radius Δ_k and [stochastic] model gradient norm $\|\nabla M(\boldsymbol{X}_k)\|$, as specified in Assumptions 6, which is crucial for the convergence guarantee.

We now make a general assumption about the behavior of the function.

Assumption 7. *(Lipschitz Continuous Gradients) The function f has Lipschitz continuous gradients, that is, there exists ν_{gL} such that $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq \nu_{gL} \|\boldsymbol{x} - \boldsymbol{y}\|$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{X} \subseteq \mathbb{R}^d$.*

Similar analysis to the one in Chapter 3 can also be performed to enforce the boundedness of the function estimates and hence the Lipschitz continuous gradients assumption can be restricted to a region instead of the entire domain but that requires changing $\{\lambda_k\}$ to $k^{3(1+\varepsilon)} = \mathcal{O}(\lambda_k)$.

Assumption 8. *(Simulation Error) The Monte Carlo oracle, when executed at $\boldsymbol{X}_k$, generates independent and identically distributed random variates $F_j(\boldsymbol{X}_k) = f(\boldsymbol{X}_k) + \xi_i \,|\, \mathcal{F}_k$, where $\xi_1, \xi_2, \ldots$ is a martingale-difference sequence adopted to $\mathcal{F}_k$ such that*

$$\sup_k \mathbb{E}[|\xi_i|^{4v} \,|\, \mathcal{F}_k] < \infty$$

for some $v \geq 2$.

Assumption 8 is arguably a mild assumption relating to the simulation oracle. The convergence analysis is also based on the standing assumptions listed in Chapter 1. So while these assumptions hold, we refuse to state them in the statement of the following lemmas and theorems. Also we specifically use the results of Lemma 11 and Lemma 12 from Chapter 3.

Remark 6. *It is our view that the minimum rate of increase on the lower bound sequence $\{\lambda_k\}$ can be reduced to a logarithmic increase instead of what has been specified in the inputs of Algorithm 5. In the notation of Theorem 7, where $\bar{X}_n = n^{-1}\sum_{k=1}^{n} X_i$ for i.i.d random variables X_i, $i = 1, 2, \cdots, n$ and N being the stopping time defined in the Theorem, this will require a large-deviation type bound on the tail probability $\mathbb{P}\{|\bar{X}_N| > t\}$ after assuming the existence of the moment-generating function of X_i's. To the best of our knowledge there currently exist no such results for fixed-width confidence interval stopping, which is the context of Theorem 7.*

4.4 Stochastic Interpolation Model Gradient Error Bounds

Before we present the main convergence results, we derive bounds on the stochastic interpolation error incurred in Step 2 of Algorithm 5 between the model gradient and the function gradient.

Lemma 14. *Let $\mathcal{Y} = \{\boldsymbol{X}, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_p\}$ be a Λ-poised set on $\mathcal{B}(\boldsymbol{X}; \Delta)$. Let $m(\boldsymbol{z})$ be an interpolation model of f on $\mathcal{B}(\boldsymbol{X}; \Delta)$; let $M(\boldsymbol{z})$ be the corresponding stochastic interpolation model of f on $\mathcal{B}(\boldsymbol{X}; \Delta)$ constructed using observations $\bar{F}\left(\boldsymbol{X}, \tilde{N}(\boldsymbol{X})\right) = f(\boldsymbol{X}) + E_1$, $\bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right) = f(\boldsymbol{Y}_i) + E_i$ for $i = 2, \ldots, p$.*

(i) For all $\boldsymbol{z} \in \mathcal{B}(\boldsymbol{Y};\Delta)$,

$$|M(\boldsymbol{z}) - m(\boldsymbol{z})| \leq \kappa_{me}(d) \max_{\boldsymbol{Y}_i \in \mathcal{Y},\ i=1,\ldots,p} \left|\bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right) - f(\boldsymbol{Y}_i)\right|,$$

where $\kappa_{me}(d) > 0$ *is a dimension dependent constant.*

(ii) If $M(\cdot)$ *and* $m(\cdot)$ *are linear, there exist positive constants* $\kappa_{egL1}, \kappa_{egL2}$ *such that*

$$\|\nabla M(\boldsymbol{X}) - \nabla f(\boldsymbol{X})\| \leq \kappa_{egL1}\Delta + \kappa_{egL2}\frac{\sqrt{\sum_{i=2}^{d+1}(E_i - E_1)^2}}{\Delta}.$$

If $M(\cdot)$ *and* $m(\cdot)$ *are quadratic, there exist positive constants* $\kappa_{egQ1}, \kappa_{egQ2}$ *such that*

$$\|\nabla M(\boldsymbol{X}) - \nabla f(\boldsymbol{X})\| \leq \kappa_{egQ1}\Delta^2 + \kappa_{egQ2}\frac{\sqrt{\sum_{i=2}^{(d+1)(d+2)/2}(E_i - E_1)^2}}{\Delta}.$$

For readability we drop the subscript k and superscript j_k representing the outer loop iteration number and the inner loop iteration number in the proof of Lemma 14.

Proof of (i). We know that for all $\boldsymbol{z} \in \mathcal{B}(\boldsymbol{Y}_1;\Delta)$

$$m(\boldsymbol{z}) = \sum_{i=1}^{p} \ell_i(\boldsymbol{z}) f(\boldsymbol{Y}_i);\ M(\boldsymbol{z}) = \sum_{i=1}^{p} \ell_i(\boldsymbol{z}) \bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right),$$

with Lagrange polynomials $\ell_j(\boldsymbol{z})$ defined in Definition 4. Notice that for univariate interpolation i.e. $d = 1$, one has

$$\ell_j(z) = \prod_{\substack{i \neq j \\ 1 \leq i \leq p}} \frac{(z - Y_i)}{(Y_j - Y_i)}.$$

Recall $|\ell_i(\boldsymbol{z})| = |\det(\mathscr{P}(\Phi, \mathcal{Y}))|^{-1} |\det(\mathscr{P}(\Phi, \mathcal{Y}_i(\boldsymbol{z})))|$, with $\mathcal{Y}_i(\boldsymbol{z}) = \mathcal{Y} \setminus \{\boldsymbol{Y}_i\} \cup \boldsymbol{z}$. From Definition 5 we know $\mathscr{P}(\Phi, \mathcal{Y})$ is nonsingular and therefore $|\det(\mathscr{P}(\Phi, \mathcal{Y}))| > 0$. For all $\boldsymbol{z} \in \mathcal{B}(\boldsymbol{X};\Delta)$ one has that $|\det(\mathscr{P}(\Phi, \mathcal{Y}_i(\boldsymbol{z})))| < \infty$ (think about the volume of the simplex of vertices of Φ after the replacement, noted in Remark 2) and hence $|\ell_i(\boldsymbol{z})| < \infty$. Let Λ_ℓ be defined as

$$\Lambda_\ell = \max_{i=1,\cdots,p} \max_{\boldsymbol{z} \in \mathcal{B}(\boldsymbol{Y}_1;\Delta)} |\ell_i(\boldsymbol{z})|.$$

It turns out Λ_ℓ is closely related to Lebesgue constant, defined as $\max_{\boldsymbol{z}\in\mathcal{B}(\boldsymbol{Y}_1;\Delta)}\sum_{i=1}^{p}|\ell_i(\boldsymbol{z})|$. Hence

$$\begin{aligned}|M(\boldsymbol{z})-m(\boldsymbol{z})| &= \left|\sum_{i=1}^{p}\ell_i(\boldsymbol{z})\left(\bar{F}\left(\boldsymbol{Y}_i,\tilde{N}(\boldsymbol{Y}_i)\right)-f(\boldsymbol{Y}_i)\right)\right| \\ &\leq \Lambda_l\sum_{i=1}^{p}\left|\bar{F}\left(\boldsymbol{Y}_i,\tilde{N}(\boldsymbol{Y}_i)\right)-f(\boldsymbol{Y}_i)\right| \\ &\leq p\Lambda_l\max_{i\in\{1,\cdots,p\}}\left|\bar{F}\left(\boldsymbol{Y}_i,\tilde{N}(\boldsymbol{Y}_i)\right)-f(\boldsymbol{Y}_i)\right|.\end{aligned}$$

In Theorem 3.14 of [44, p. 51] it has been shown that Λ_ℓ is connected to the condition number of the matrix $\mathscr{P}\left(\Phi,\hat{\mathcal{Y}}\right)$ where $\hat{\mathcal{Y}}$ is scaled by Δ and shifted with respect to $\boldsymbol{Y}_1$ version of $\mathcal{Y}$, i.e. $\hat{\mathcal{Y}}=\left\{0,(\boldsymbol{Y}_2-\boldsymbol{Y}_1)/\Delta,\cdots,(\boldsymbol{Y}_p-\boldsymbol{Y}_1)/\Delta\right\}\subset\mathcal{B}(0;1)$. Specifically it is shown that if $\left\|\mathscr{P}\left(\Phi,\hat{\mathcal{Y}}\right)^{-1}\right\|\leq\Lambda$, then $\hat{\mathcal{Y}}$ is $\sqrt{p}\Lambda$-poised in the unit ball $\mathcal{B}(0;1)$. ■

Proof of (ii). We first derive the error bound expression for the stochastic linear interpolation model, and then extend that to the quadratic interpolation model.

Let $M_L(\boldsymbol{X}+\boldsymbol{s})=M_L(\boldsymbol{X})+\boldsymbol{s}^T\nabla M_L(\boldsymbol{X})$ be a stochastic linear interpolation model of f on $\mathcal{B}(\boldsymbol{X};\Delta)$ as in Definition 10. Next let $\mathscr{L}_k:=\mathscr{L}(\mathcal{Y})$ as in Definition 6 be nonsingular, and by interpolation $M_L(\boldsymbol{X})=\bar{F}\left(\boldsymbol{X},\tilde{N}(\boldsymbol{X})\right)$ and $M_L(\boldsymbol{Y}_i)=\bar{F}\left(\boldsymbol{Y}_i,\tilde{N}(\boldsymbol{Y}_i)\right)$ for $i=2,\ldots,p$. After subtracting the first row from all other rows of $\mathscr{L}_k$, we can write

$$(\boldsymbol{Y}_i-\boldsymbol{X})^T\nabla M_L(\boldsymbol{X})=M_L(\boldsymbol{Y}_i)-M_L(\boldsymbol{X})=(f(\boldsymbol{Y}_i)-f(\boldsymbol{X}))+(E_i-E_1)$$

for $i=2,\ldots,p$. By Mean Value Theorem

$$f(\boldsymbol{Y}_i)-f(\boldsymbol{X})=\int_0^1(\boldsymbol{Y}_i-\boldsymbol{X})^T\nabla f(\boldsymbol{X}+t(\boldsymbol{Y}_i-\boldsymbol{X}))\,dt,$$

hence by Lipschitz continuity of ∇f assumed in Assumption 7, for all $i=2,\ldots,d+1$

$$(\boldsymbol{Y}_i-\boldsymbol{X})^T(\nabla f(\boldsymbol{X})-\nabla M_L(\boldsymbol{X}))\leq\frac{\nu_{gL}}{2}\|\boldsymbol{Y}_i-\boldsymbol{X}\|^2+(E_i-E_1).$$

Recall $\mathscr{L}(\mathcal{Y})$ from Definition 6. Now let us define

$$\tilde{\mathscr{L}}(\mathcal{Y})=\Delta^{-1}\begin{bmatrix}y_2^1-y_1^1 & y_2^2-y_1^2 & \cdots & y_2^d-y_1^d\\ \vdots & \vdots & \vdots & \vdots\\ y_{d+1}^1-y_1^1 & y_{d+1}^2-y_1^2 & \cdots & y_{d+1}^d-y_1^d\end{bmatrix}.$$

Then from all the inequalities we obtain

$$\Delta\tilde{\mathscr{L}}(\mathcal{Y})(\nabla M_L(\boldsymbol{X})-\nabla f(\boldsymbol{X}))\leq\frac{\nu_{gL}}{2}\begin{bmatrix}\|\boldsymbol{Y}_2-\boldsymbol{X}\|^2\\ \vdots\\ \|\boldsymbol{Y}_{d+1}-\boldsymbol{X}\|^2\end{bmatrix}+\begin{bmatrix}E_2-E_1\\ \vdots\\ E_{d+1}-E_1\end{bmatrix}.$$

By taking the norm of the above inequality, and knowing that $\tilde{L}(\mathcal{Y})$ is nonsingular from Definition 5 we arrive at

$$\|\nabla M_L(\boldsymbol{X})-\nabla f(\boldsymbol{X})\|\leq\frac{\left\|\tilde{L}^{-1}(\mathcal{Y})\right\|}{\Delta}\left(\frac{\nu_{gL}}{2}\sqrt{d}\Delta^2+\sqrt{\sum_{i=2}^{d+1}(E_i-E_1)^2}\right).$$

Hence $\kappa_{egL1}=\left\|\tilde{L}_k^{-1}(\mathcal{Y})\right\|\frac{\nu_{gL}}{2}\sqrt{d}$ and $\kappa_{egL2}=\left\|\tilde{L}_k^{-1}(\mathcal{Y})\right\|$.

Now we consider a stochastic quadratic interpolation model of f on $\mathcal{B}(\boldsymbol{X};\Delta)$ as in Definition 11

$$\begin{aligned}M_Q(\boldsymbol{Y}_i)-M_Q(\boldsymbol{X})&=(\boldsymbol{Y}_i-\boldsymbol{X})^T\nabla M_Q(\boldsymbol{X})+2^{-1}(\boldsymbol{Y}_i-\boldsymbol{X})^T\nabla^2 M_Q(\boldsymbol{X}_k)(\boldsymbol{Y}_i-\boldsymbol{X})\\&=(f(\boldsymbol{Y}_i)-f(\boldsymbol{X}))+(E_i-E_1),\end{aligned}$$

where $\nabla M_Q(\boldsymbol{X}_k)=\boldsymbol{G}_k+\hat{\mathscr{B}}_k\boldsymbol{X}_k$ and $\nabla^2 M_Q(\boldsymbol{X}_k)=\hat{\mathscr{B}}$, assuming that f has Lipschitz continuous Hessian with the Lipschitz constant ν_{HL}. Using Taylor expansion we can write

$$\begin{aligned}&(\boldsymbol{Y}_i-\boldsymbol{X})^T(\nabla M_Q(\boldsymbol{X}_k)-\nabla f(\boldsymbol{X}_k))+\frac{1}{2}(\boldsymbol{Y}_i-\boldsymbol{X})^T\left(\nabla^2 M_Q(\boldsymbol{X})-\nabla^2 f(\boldsymbol{X})\right)(\boldsymbol{Y}_i-\boldsymbol{X})\\&\leq\frac{\Delta^3}{6}+E_i-E_1\end{aligned}$$

for $i=2,\ldots,p$. Recall $\mathscr{Q}(\mathcal{Y})$ from Definition 6. Similar to $\tilde{\mathscr{L}}(\mathcal{Y})$ we define $\tilde{\mathscr{Q}}(\mathcal{Y})$ as

$$\tilde{\mathscr{Q}}(\mathcal{Y})=\begin{bmatrix}\mathscr{D}_{\Delta^{-1}} & 0\\ 0 & \mathscr{D}_{\Delta^{-2}}\end{bmatrix}\begin{bmatrix}\Delta\tilde{\mathscr{L}}(\mathcal{Y}) & \frac{\left(y_2^1-y_1^1\right)^2}{2} & \left(y_2^1-y_1^1\right)\left(y_2^2-y_1^2\right) & \cdots & \frac{\left(y_2^d-y_1^d\right)^2}{2}\\ & \vdots & \vdots & \vdots & \vdots\\ & \frac{\left(y_p^1-y_1^1\right)^2}{2} & \left(y_p^1-y_1^1\right)\left(y_p^2-y_1^2\right) & \cdots & \frac{\left(y_p^d-y_1^d\right)^2}{2}\end{bmatrix}$$

where $\mathscr{D}_{\Delta^{-1}}$ is a diagonal matrix of dimension d with Δ^{-1} diagonal entries and $\mathscr{D}_{\Delta^{-2}}$ is a diagonal matrix of dimension $\sqrt{d(d+1)/2}$ with Δ^{-2} diagonal entries. With all the

above adjustments and defending the vector W with the elements of the matrix $\nabla^2 M_Q(\boldsymbol{X}) - \nabla^2 f(\boldsymbol{X})$ correctly ordered we get

$$\left\| \begin{bmatrix} \mathscr{D}_\Delta & 0 \\ 0 & \mathscr{D}_{\Delta^2} \end{bmatrix} \tilde{\mathscr{Q}}(\mathcal{Y}) \begin{bmatrix} \nabla M_Q(\boldsymbol{X}_k) - \nabla f(\boldsymbol{X}_k) \\ W \end{bmatrix} \right\|$$
$$\leq \frac{\nu_{HL}}{6}\sqrt{\frac{d(d+1)+d}{2}}\Delta^3 + \sqrt{\sum_{i=2}^{(d+1)(d+2)/2}(E_i - E_1)^2}.$$

Each vector in the left hand side is less than or equal to the quantity on the right. Knowing that the scaled matrix $\tilde{Q}(\mathcal{Y})$ is nonsingular from Definition 5 we arrive at

$$\|\mathscr{D}_\Delta(\nabla M_Q(\boldsymbol{X}_k) - \nabla f(\boldsymbol{X}_k))\| \leq \left\|\tilde{\mathscr{Q}}^{-1}(\mathcal{Y})\right\| \left(\frac{\nu_{HL}}{6}\sqrt{\frac{d(d+1)+d}{2}}\Delta^3 + \sqrt{\sum_{i=2}^{p}(E_i - E_1)^2} \right).$$

Hence $\kappa_{egQ1} = \left\|\tilde{\mathscr{Q}}^{-1}(\mathcal{Y})\right\| \frac{\nu_{HL}}{6}\sqrt{\frac{d(d+1)+d}{2}}$ and $\kappa_{egQ2} = \left\|\tilde{\mathscr{Q}}^{-1}(\mathcal{Y})\right\|$. ■

Note that for E_i defined in the lemma above since the sample average estimators are unbiased, $\mathbb{E}[E_i] = 0$ and in addition by Assumption 8, $\mathrm{Var}(E_1) = \sigma^2(\boldsymbol{X}_k)/\tilde{N}(\boldsymbol{X}_k)$, and $\mathrm{Var}(E_i) = \sigma^2(\boldsymbol{Y}_i)/\tilde{N}(\boldsymbol{Y}_i)$ for $i = 2, \ldots, p$.

This result is crucial for almost sure convergence of the stochastic model gradient to the true gradient, that is a key to the overall algorithm convergence results.

Next, we demonstrate through the following result that the model construction algorithm (Algorithm 6) terminates with probability one, whenever the incumbent solution $\boldsymbol{X}_k$ is not a first-order critical point.

Lemma 15. *Suppose the incumbent solution $\boldsymbol{X}_k \in \mathbb{X}$ during the kth iteration is not first-order critical, that is, $\nabla f(\boldsymbol{X}_k) \neq 0$. Then Algorithm 6 terminates in a finite number of steps with probability one.*

Proof. Set $\|\nabla f(\boldsymbol{X}_k)\| = c' > 0$. We will prove the assertion through a contradiction argument.

First, we notice that the contraction loop (Steps 3-9 in Algorithm 6 is not entered if $\mu\|\nabla M_k(\boldsymbol{X}_k)\| \geq \tilde{\Delta}_k$, in which case which case Algorithm 6 terminates trivially.

Next, suppose $\mu \left\| \nabla M_k \left(\boldsymbol{X}_k \right) \right\| < \tilde{\Delta}_k$ and that the contraction loop in Steps 3-9 of Algorithm 6 is infinite. Let $\nabla M_k^{(j_k)} \left(\boldsymbol{X}_k \right)$ denote the model gradient during the j_k-th iteration of the contraction loop. Then $\mu \left\| \nabla M_k^{(j_k)} \left(\boldsymbol{X}_k \right) \right\| < \tilde{\Delta}_k w^{j_k - 1}$, $\forall j \geq 1$. This means, since $w < 1$, that $\tilde{\Delta}_k w^{j_k - 1} \to 0$ and therefore $\left\| \nabla M_k^{(j_k)} \left(\boldsymbol{X}_k \right) \right\| \xrightarrow{wp1} 0$ as $j \to \infty$. Furthermore, due to the sampling rule in (4.2) and by Theorem 1, we have that $\tilde{N} \left(\boldsymbol{Y}_i^{(j_k)} \right) \to \infty$ as $j_k \to \infty$, where $\tilde{N} \left(\boldsymbol{Y}_i^{(j_k)} \right)$ is the sample size at point $\boldsymbol{Y}_i^{(j_k)}$ after the j_k-th iteration of the contraction loop. Now, if $E_{k,i}^{(j_k)}$ denotes the error due to sampling at point $\boldsymbol{Y}_i^{(j_k)}$ after the j_k-th iteration of the contraction loop, that is, $E_{k,i}^{(j_k)} = \bar{F} \left(\boldsymbol{Y}_i^{(j_k)}, \tilde{N} \left(\boldsymbol{Y}_i^{(j_k)} \right) \right) - f \left(\boldsymbol{Y}_i^{(j_k)} \right)$, we can write for large enough k and some $\delta > 0$,

$$\begin{aligned}
\mathbb{P} \left\{ \frac{\sum_{i=2}^{p} \left| E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)} \right|}{\tilde{\Delta}_k w^{j_k - 1}} \geq c \right\} &\leq \sum_{i=2}^{p} \mathbb{E} \left[\mathbb{P} \left\{ \left| E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)} \right| \geq \frac{c \tilde{\Delta}_k w^{j_k - 1}}{(p-1)} \,|\, \mathcal{F}_k \right\} \right] \\
&\leq 2(p-1)^3 4 \left(c \tilde{\Delta}_k w^{j_k - 1} \right)^{-2} (1 + \delta) \kappa_{ias}^2 \Delta_k^4 \lambda_k^{-1} \\
&\leq 8(p-1)^3 c^{-2} (1 + \delta) \kappa_{ias}^2 \Delta_k^2 \lambda_k^{-1}, \qquad (4.3)
\end{aligned}$$

where the penultimate inequality above follows from arguments identical to those leading to (3.12) in the proof of Lemma 11 after using the adaptive sample size expression in (4.2). Since the right-hand side of (4.3) is summable, we conclude by Borel-Cantelli's first lemma [61] that

$$\left(\tilde{\Delta}_k w^{j_k - 1} \right)^{-1} \sum_{i=2}^{p} \left| E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)} \right| \xrightarrow{wp1} 0.$$

This implies, from Lemma 14 and since Algorithm 2 maintains models that are of sufficient quality, that

$$\left\| \nabla f \left(\boldsymbol{X}_k \right) - \nabla M_k^{(j_k)} \left(\boldsymbol{X}_k \right) \right\| \leq \kappa_1 \left(\tilde{\Delta}_k w^{j_k - 1} \right)^{\theta} + \kappa_2 \left(\tilde{\Delta}_k w^{j_k - 1} \right)^{-1} \sum_{i=1}^{p} \left| E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)} \right| \xrightarrow{wp1} 0$$

as $j_k \to \infty$, where θ, κ_1, and κ_2 are according to part (ii) of Lemma 14. We have arrived at a contradiction since we argued that $\left\| \nabla M_k^{(j_k)} \left(\boldsymbol{X}_k \right) \right\| \xrightarrow{wp1} 0$ but then $\| \nabla f \left(\boldsymbol{X}_k \right) \| = c > 0$ by the contrapositive assumption.

■

The next lemma shows that if a model has sufficient quality in a ball, it remains with sufficient quality in any larger concentric ball. The proof is repeated from [44, p. 200]

for completion. This result is needed to justify that the resulting model from Step 2 of Algorithm 5 has sufficient quality in $\mathcal{B}(\boldsymbol{X}_k;\Delta_k)$, given that Δ_k can be larger than $\tilde{\Delta}_k w^{j_k-1}$ from Step 11 of Algorithm 6.

Lemma 16. *Suppose $M_k(\cdot)$ is a stochastic interpolation model constructed on $\mathcal{B}\left(\boldsymbol{X}_k;\tilde{\Delta}_k\right)$ such that $\mathbb{E}[M_k(\boldsymbol{z})]$ is $\left(\kappa_{ef},\kappa_{eg}\right)$-fully linear on $\mathcal{B}\left(\boldsymbol{X}_k;\tilde{\Delta}_k\right)$. Then $\mathbb{E}[M_k(\boldsymbol{z})]$ is also $\left(\kappa_{ef},\kappa_{eg}\right)$-fully linear on $\mathcal{B}(\boldsymbol{X}_k;\Delta_k)$ for $\Delta_k \in \left[\tilde{\Delta}_k,\Delta_{max}\right]$.*

Proof. Let $\boldsymbol{s}$ be such that $\|\boldsymbol{s}\| \in \left[\tilde{\Delta}_k,\Delta_k\right]$ and let $\theta = \tilde{\Delta}_k/\|\boldsymbol{s}\|$. Then $m_k(\boldsymbol{X}_k+\theta\boldsymbol{s}) = \mathbb{E}[M_k(\boldsymbol{X}_k+\theta\boldsymbol{s})]$ is $\left(\kappa_{ef},\kappa_{eg}\right)$-fully linear on $\mathcal{B}\left(\boldsymbol{X}_k;\tilde{\Delta}_k\right)$:

$$\|\nabla m_k(\boldsymbol{X}_k+\theta\boldsymbol{s}) - \nabla f(\boldsymbol{X}_k+\theta\boldsymbol{s})\| \leq \kappa_{eg}\tilde{\Delta}_k.$$

Since we know ∇f and ∇m_k are Lipschitz continuous with Lipschitz constants ν_{gL} and ν_{gL}^m and assuming that without loss of generality $\kappa_{eg} > \nu_{gL} + \nu_{gL}^m$,

$$\begin{aligned}
&\|\nabla f(\boldsymbol{X}_k+\boldsymbol{s}) - \nabla f(\boldsymbol{X}_k+\theta\boldsymbol{s}) - \nabla m_k(\boldsymbol{X}_k+\theta\boldsymbol{s}) + \nabla m_k(\boldsymbol{X}_k+\boldsymbol{s})\| \\
&\leq \|\nabla f(\boldsymbol{X}_k+\boldsymbol{s}) - \nabla f(\boldsymbol{X}_k+\theta\boldsymbol{s})\| + \|\nabla m_k(\boldsymbol{X}_k+\boldsymbol{s}) - \nabla m_k(\boldsymbol{X}_k+\theta\boldsymbol{s})\| \\
&\leq \nu_{gL}\|\boldsymbol{s}\|(1-\theta) + \nu_{gL}^m\|\boldsymbol{s}\|(1-\theta) \\
&= \left(\nu_{gL}+\nu_{gL}^m\right)\left(\|\boldsymbol{s}\| - \tilde{\Delta}_k\right) \\
&\leq \kappa_{eg}\left(\|\boldsymbol{s}\| - \tilde{\Delta}_k\right).
\end{aligned}$$

Combining the above inequalities we reach $\|\nabla f(\boldsymbol{X}_k+\boldsymbol{s}) - \nabla m_k(\boldsymbol{X}_k+\boldsymbol{s})\| \leq \kappa_{eg}\|\boldsymbol{s}\| \leq \kappa_{eg}\Delta_k$.

Now we define $\phi(\alpha) = f(\boldsymbol{X}_k+\alpha\boldsymbol{s}) + m_k(\boldsymbol{X}_k+\alpha\boldsymbol{s})$ for $\alpha \in [0,1]$. We know that $|\phi(\theta)| \leq \kappa_{ef}\tilde{\Delta}_k^2$ since the model is $\left(\kappa_{ef},\kappa_{eg}\right)$-fully linear on $\mathcal{B}\left(\boldsymbol{X}_k;\tilde{\Delta}_k\right)$. We need to bound $|\phi(1)|$. Assuming withouth loss of generality that $\kappa_{ef} \geq 2^{-1}\kappa_{eg}$ we write

$$\begin{aligned}
|\phi(1) - \phi(\theta)| &= \left|\int_\theta^1 \phi'(\alpha)\,d\alpha\right| \\
&\leq \int_\theta^1 \|\boldsymbol{s}\|\,\|f(\boldsymbol{X}_k+\alpha\boldsymbol{s}) - m_k(\boldsymbol{X}_k+\alpha\boldsymbol{s})\|\,d\alpha \cdot \\
&\leq \int_\theta^1 \kappa_{eg}\|\boldsymbol{s}\|^2\,\alpha d\alpha = \frac{1}{2}\kappa_{eg}\left(\|\boldsymbol{s}\|^2 - \tilde{\Delta}_k^2\right).
\end{aligned}$$

Hence we finally arrive at

$$\begin{aligned}|f(\boldsymbol{X}_k+\boldsymbol{s})-m_k(\boldsymbol{X}_k+\boldsymbol{s})| &\leq |\phi(1)-\phi(\theta)|+|\phi(\theta)| \\ &\leq \kappa_{ef}\|\boldsymbol{s}\|^2 \leq \kappa_{ef}\tilde{\Delta}_k^2.\end{aligned}$$

which implies the model is $\left(\kappa_{ef},\kappa_{eg}\right)$-fully linear on $\mathcal{B}(\boldsymbol{X}_k;\Delta_k)$.

■

4.5 Main Results

We are now ready to present the main convergence results. We crucially use the result in Lemma 11 stating that when Assumption 8 holds, the sequence of function estimates observed across the iterates is almost surely bounded from below, that is, mischance cannot lead ASTRO-DF's iterates to wander in an unbounded fashion.

Next, we state a theorem that plays a crucial role in proving the overall convergence of ASTRO-DF iterates. Recall that even in deterministic TRO-DF algorithms, unlike trust-region algorithms where derivative observations are available, the trust-region radius necessarily needs to converge to zero for successful convergence. Theorem 10 states that this is indeed the case for ASTRO-DF. The proof rests on Lemma 11 and the assumed sufficient Cauchy decrease guarantee during Step 11 of Algorithm 6.

Theorem 10. *Suppose f is continuously differentiable and bounded from below. Let Assumptions 5 , 6, and 8 hold. Then $\Delta_k \xrightarrow{wp1} 0$ as $k \to \infty$.*

Proof. Note that $\bar{F}(\boldsymbol{X}_i,N_i)$ denotes the function estimate at the point $\boldsymbol{X}_i$ *before* entering the model construction step during the i-th iteration (or the function estimate at the candidate point at the end of the $(i-1)$-th iteration), and $\bar{F}\left(\boldsymbol{X}_i,\tilde{N}_i\right)$ the function estimate upon exiting the model construction step during the i-th iteration. We can then write

$$\bar{F}(\boldsymbol{X}_k,N_k) = \bar{F}(\boldsymbol{X}_1,N_1) + \sum_{i=1}^{k-1}(A_i+B_i) \tag{4.4}$$

where the summands $A_i = \bar{F}(\boldsymbol{X}_{i+1},N_{i+1}) - \bar{F}\left(\boldsymbol{X}_i,\tilde{N}_i\right)$ and $B_i = \bar{F}\left(\boldsymbol{X}_i,\tilde{N}_i\right) - \bar{F}(\boldsymbol{X}_i,N_i)$. In words A_i represents the reduction in the function estimates during the ith iteration and B_i

represents the difference between the two estimates of the function at the point $\boldsymbol{X}_i$ at the end of iteration $i-1$ and i. We now make the following observations about A_i and B_i.

(a) If i is an unsuccessful iteration, then $A_i = 0$ since $\boldsymbol{X}_i = \boldsymbol{X}_{i+1}$.

(b) If i is a successful iteration, we know by definition that $\hat{\rho}_i \geq \eta_1$. If we denote $\kappa_{efd} = (2\mu)^{-1}\eta_1\kappa_{fcd}\min\left\{(\mu\kappa_{bhm})^{-1}, 1\right\}$, then by Assumptions 5 and 6 , and by the assurance in Algorithm 6 that $\Delta_k \leq \mu\|\nabla M_k(\boldsymbol{X}_k)\|$, we have

$$\begin{aligned} A_i &\leq \eta_1\left(M_i(\boldsymbol{X}_{i+1}) - M_i(\boldsymbol{X}_i)\right) \\ &\leq -\frac{\eta_1}{2}\kappa_{fcd}\|\nabla M_i(\boldsymbol{X}_i)\|\min\left\{\frac{\|\nabla M_i(\boldsymbol{X}_i)\|}{\|\nabla^2 M_i(\boldsymbol{X}_i)\|}, \Delta_i\right\} \\ &\leq -\kappa_{efd}\Delta_i^2. \end{aligned} \tag{4.5}$$

(c) For any given $c > 0$, (3.12) in the proof of Lemma 11 ensures that $\mathbb{P}\{|B_i| > c, \text{ i.o.}\} = 0$ since

$$\begin{aligned} \mathbb{P}\left\{\left|\bar{F}\left(\boldsymbol{X}_i, \tilde{N}_i\right) - \bar{F}\left(\boldsymbol{X}_i, N_i\right)\right| > c\right\} \leq\ & \mathbb{P}\left\{\left|\bar{F}\left(\boldsymbol{X}_i, \tilde{N}_i\right) - f\left(\boldsymbol{X}_i\right)\right| > \frac{c}{2}\right\} \\ & + \mathbb{P}\left\{\left|f\left(\boldsymbol{X}_i\right) - \bar{F}\left(\boldsymbol{X}_i, N_i\right)\right| > \frac{c}{2}\right\} \end{aligned}$$

using the Boole's inequality (see Definition 9). This implies that except for a set of measure zero, $|B_i| \leq c$ for large enough i.

Now suppose $\mathcal{D} := \{\omega : \lim_{k\to\infty}\Delta_k(\omega) \neq 0\}$ denotes the set of sample-paths for which the trust-region radius does not decay to zero. For contraposition, suppose $\mathcal{D}$ has positive measure. Consider a sample-path $\omega_0 \in \mathcal{D}$. Since unsuccessful iterations are necessarily contracting iterations, we can find $\delta(\omega_0) > 0$ and a sub-sequence of successful iterations $\{k_j\}$ in the sample-path ω_0 such that $\Delta_{k_j}(\omega_0) \geq \delta(\omega_0)$. This implies from observation (b) above that

$$A_{k_j}(\omega_0) \leq -\kappa_{efd}\delta^2(\omega_0). \tag{4.6}$$

Now the iterations $k_j + 1, \ldots, k_{j+1} - 1$ are all unsuccessful iterations, implying from observation (a) above that

$$A_{k_j+\ell} = 0, \ell = 1, 2, \ldots, k_{j+1} - k_j - 1. \tag{4.7}$$

Also, by the observation (c) above, and choosing $c = \frac{1}{3}\kappa_{efd}\delta^2(\omega_0)$, we see that for large-enough i,

$$\left|\bar{F}\left(\boldsymbol{X}_i(\omega_0), \tilde{N}_i(\omega_0)\right) - \bar{F}\left(\boldsymbol{X}_i(\omega_0), N_i(\omega_0)\right)\right| \leq \frac{2}{3}\kappa_{efd}\delta^2(\omega_0). \tag{4.8}$$

We then write for large-enough j,

$$\begin{aligned}
\sum_{\ell=k_j}^{k_{j+1}-1}\left(A_\ell(\omega_0) + B_\ell(\omega_0)\right) &= A_{k_j}(\omega_0) + \sum_{\ell=k_j}^{k_{j+1}-1} B_\ell(\omega_0) \\
&= A_{k_j}(\omega_0) \\
&+ \bar{F}\left(\boldsymbol{X}_{k_{j+1}-1}(\omega_0), \tilde{N}_{k_{j+1}-1}(\omega_0)\right) - \bar{F}\left(\boldsymbol{X}_{k_j+1}(\omega_0), N_{k_j+1}(\omega_0)\right) \\
&\leq -\frac{1}{3}\kappa_{efd}\delta^2(\omega_0),
\end{aligned} \tag{4.9}$$

where the first equality follows from observation (a) above, the second equality follows from the definition of B_ℓ, and the third inequality follows from (4.6) and (4.8). The inequality in (4.9) (and the fact that there is an entire sequence $\{k_j\}$ of successful iterations) means that $\lim_{k\to\infty}\bar{F}\left(\boldsymbol{X}_k(\omega_0), N_k(\omega_0)\right) = -\infty$ thus contradicting Lemma 11. The assertion of the theorem thus holds. ■

Relying on Theorem 10, we now show that the model gradient converges to the true gradient almost surely. This, of course, does not imply that the true gradient itself converges to zero — a fact that will be established subsequently.

Lemma 17. *Suppose f is continuously differentiable and bounded from below. Let Assumptions 5, 6, and 8 hold. Then $\|\nabla M_k(\boldsymbol{X}_k) - \nabla f(\boldsymbol{X}_k)\| \xrightarrow{wp1} 0$ as $k \to \infty$.*

Proof. From Lemma 15 the stochastic model M_k constructed via Algorithm 6 terminates in finite time. In Step 2 of Algorithm 6 let $\tilde{\Delta}_k w^{j_k-1}$ denote the trust-region radius over which the model is constructed. (Note that due to Step 11 of Algorithm 6, $\tilde{\Delta}_k w^{j_k-1}$ may or may not equal the exiting trust-region radius Δ_k upon completion of k iterations of ASTRO-DF.) Then, we know from part (ii) of Lemma 14 that

$$\|\nabla M_k(\boldsymbol{X}_k) - \nabla f(\boldsymbol{X}_k)\| \leq \kappa_1\left(\tilde{\Delta}_k w^{j_k-1}\right)^\theta + \kappa_2\frac{\sqrt{\sum_{i=2}^p\left(E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right)^2}}{\left(\tilde{\Delta}_k w^{j_k-1}\right)},$$

where $E_{k,1}^{(j_k)} = \bar{F}\left(\boldsymbol{X}_k, \tilde{N}(\boldsymbol{X}_k)\right) - f(\boldsymbol{X}_k)$ is the error of the sampled function estimate at the center point of the trust-region, and $E_{k,i}^{(j_k)} = \bar{F}\left(\boldsymbol{Y}_i^{(j_k)}, \tilde{N}\left(\boldsymbol{Y}_i^{(j_k)}\right)\right) - f\left(\boldsymbol{Y}_i^{(j_k)}\right)$ for $i = 2, \ldots, p$ are the errors of the sampled function estimates at the interpolation points. (Note that $p = d+1$ and $\theta = 1$ in the linear interpolation models, and $p = (d+1)(d+2)/2$ and $\theta = 2$ in the quadratic interpolation models. For the quantities κ_1 and κ_2 refer to part (ii) of Lemma 14.) For readability we let $\boldsymbol{X}_k = \boldsymbol{Y}_1^{(j_k)}$.

We know from Theorem 10 that $\Delta_k \xrightarrow{wp1} 0$ as $k \to \infty$, and hence, $\tilde{\Delta}_k w^{j_k - 1} \xrightarrow{wp1} 0$ as $k \to \infty$. Also,

$$\sqrt{\sum_{i=2}^{p} \left(E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right)^2} \leq \sum_{i=2}^{p} \sqrt{\left(E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right)^2} = \sum_{i=2}^{p} \left|E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right|.$$

Considering these two observations, it suffices to show that as $k \to \infty$,

$$\left(\tilde{\Delta}_k w^{j_k - 1}\right)^{-1} \sum_{i=2}^{p} \left|E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right| \xrightarrow{wp1} 0. \tag{4.10}$$

Towards this, we write for $c > 0$ and large enough k and some $\delta > 0$,

$$\begin{aligned}
\mathbb{P}\left\{ \frac{\sum_{i=2}^{p} \left|E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right|}{\left(\tilde{\Delta}_k w^{j_k - 1}\right)} \geq c \right\} &\leq \sum_{i=2}^{p} \mathbb{E}\left[\mathbb{P}\left\{ \left|E_{k,i}^{(j_k)} - E_{k,1}^{(j_k)}\right| \geq \frac{c\left(\tilde{\Delta}_k w^{j_k - 1}\right)}{p-1} \,|\, \mathcal{F}_k \right\} \right] \\
&\leq \sum_{i=2}^{p} \left(\mathbb{E}\left[\mathbb{P}\left\{ \left|E_{k,i}^{(j_k)}\right| \geq \frac{c\left(\tilde{\Delta}_k w^{j_k - 1}\right)}{2(p-1)} \right\} |\, \mathcal{F}_k \right] \right. \\
&\qquad \left. + \mathbb{E}\left[\mathbb{P}\left\{ \left|E_{k,1}^{(j_k)}\right| \geq \frac{c\left(\tilde{\Delta}_k w^{j_k - 1}\right)}{2(p-1)} \right\} |\, \mathcal{F}_k \right] \right) \\
&\leq 2(p-1)^3 4 \left(c\tilde{\Delta}_k w^{j_k - 1}\right)^{-2} (1+\delta) \kappa_{ias}^2 (\tilde{\Delta}_k w^{j_k - 1})^4 \lambda_k^{-1} \\
&\leq 8(p-1)^3 c^{-2} (1+\delta) \kappa_{ias}^2 \Delta_k^2 \lambda_k^{-1},
\end{aligned} \tag{4.11}$$

where the penultimate inequality above follows from arguments identical to those leading to (3.12) in the proof of Lemma 11 after using the adaptive sample size expression in (4.2). Since the right-hand side of (4.11) is summable, we can invoke the first Borel-Cantelli lemma [61] to conclude that (4.10) holds.

■

We now show that for large enough iteration k, the steps within ASTRO-DF are always successful with probability one. This result is important in that it implies that the model

gradient and the trust-region radius will remain in lock-step for large k, almost surely. The proof proceeds by dividing the model error into three components, each of which is shown to be controlled with probability one.

Theorem 11. *Suppose f is continuously differentiable and bounded from below. Let Assumptions 5, 6, and 8 hold. Then $\mathbb{P}\{|\hat{\rho}_k - 1| \geq 1 - \eta_1, \text{i.o.}\} = 0$ for any $\eta_1 \in (0,1)$.*

Proof. At the end of Step 2 of Algorithm 6, let $m_k^{(j_k)}(\boldsymbol{z})$ be the interpolation model of f constructed on the poised set $\mathcal{Y}_k$. (Of course, we cannot construct $m_k(\cdot)$ explicitly because the true function values are unknown.) Then $m_k^{(j_k)}(\boldsymbol{z})$ is a $(\kappa_{ef}, \kappa_{eg})$-fully-linear model of f on $\mathcal{B}\left(\boldsymbol{X}_k; \tilde{\Delta}_k w^{j_k - 1}\right)$ and since $\Delta_k \geq \tilde{\Delta}_k w^{j_k-1}$, by Lemma 16 we have that $m_k(\cdot)$ is a $(\kappa_{ef}, \kappa_{eg})$-fully-linear model of f on $\mathcal{B}(\boldsymbol{X}_k; \Delta_k)$. In addition, Algorithm 6 ensures that $\Delta_k \leq \mu \|\nabla M_k(\boldsymbol{X}_k)\|$.

Assumption 6 on the Cauchy decrease in the minimization problem implies that

$$\begin{aligned} M_k(\boldsymbol{X}_k) - M_k(\boldsymbol{X}_k + \boldsymbol{S}_k) &\geq \frac{\kappa_{fcd}}{2} \|\nabla M_k(\boldsymbol{X}_k)\| \min\left\{ \frac{\|\nabla M_k(\boldsymbol{X}_k)\|}{\|\nabla^2 M_k(\boldsymbol{X}_k)\|}, \Delta_k \right\} \\ &\geq \frac{\kappa_{fcd}}{2} \|\nabla M_k(\boldsymbol{X}_k)\| \min\left\{ \frac{\Delta_k}{\mu \kappa_{bhm}}, \Delta_k \right\} \\ &\geq \kappa_{md} \Delta_k^2. \end{aligned} \tag{4.12}$$

where

$\kappa_{md} = (2\mu\kappa_{bhm})^{-1} \min(\mu \kappa_{bhm}^{-1}, 1) \kappa_{fcd}$.

Recall that

$$\hat{\rho}_k := \frac{\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right)}{M_k(\boldsymbol{X}_k) - M_k(\tilde{\boldsymbol{X}}_{k+1})}$$

and that $\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) = M_k(\boldsymbol{X}_k)$. Now using Boole's inequality (see Definition 9) and (4.12), we can write

$$\begin{aligned} &\mathbb{P}\{\hat{\rho}_k < \eta_1\} = \mathbb{P}\{|1 - \hat{\rho}_k| \geq 1 - \eta_1\} \\ &\leq \mathbb{P}\left\{ \frac{\left|\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) - M_k\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right| + \left|\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - M_k(\boldsymbol{X}_k)\right|}{\left|M_k(\boldsymbol{X}_k) - M_k\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right|} \geq 1 - \eta_1 \right\} \\ &\leq \mathbb{P}\left\{ \left|\bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right) - M_k\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right| \geq (1 - \eta_1) \kappa_{md} \Delta_k^2 \right\} \\ &\leq \mathbb{P}\left\{ Err_1 \geq \eta' \Delta_k^2 \right\} + \mathbb{P}\left\{ Err_2 \geq \eta' \Delta_k^2 \right\} + \mathbb{P}\left\{ Err_3 \geq \eta' \Delta_k^2 \right\}, \end{aligned} \tag{4.13}$$

where

$$Err_1 := \left| M_k\left(\tilde{\boldsymbol{X}}_{k+1}\right) - m_k\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right|,$$
$$Err_2 := \left| m_k\left(\tilde{\boldsymbol{X}}_{k+1}\right) - f\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right|,$$
$$Err_3 := \left| f\left(\tilde{\boldsymbol{X}}_{k+1}\right) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right)\right|,$$

and $\eta' = 3^{-1}(1-\eta_1)\kappa_{md}$. (It is useful to interpret three errors Err_1, Err_2 and Err_3 on the right-hand side of (4.13) as the *stochastic interpolation error*, the *deterministic model error*, and the *stochastic sampling error respectively*.) In what follows, we establish $\mathbb{P}\{\hat{\rho}_k < \eta_1 \text{ i.o.}\} = 0$ by demonstrating that each of the errors Err_1, Err_2 and Err_3 exceeding $\eta'\Delta_k^2$ infinitely often has probability zero.

We first analyze the stochastic interpolation error probability $\mathbb{P}\left\{Err_1 \geq \eta'\Delta_k^2\right\}$ appearing on the right-hand side of (4.13). Recall $p = d+1$ for linear interpolation. Using part (i) of Lemma 14 and relabeling $\boldsymbol{X}_k$ to $\boldsymbol{Y}_1$ for readability, we write

$$\begin{aligned}
\mathbb{P}\left\{Err_1 > \eta'\Delta_k^2\right\} &\leq \mathbb{P}\left\{ \max_{\substack{\boldsymbol{Y}_i \in \mathcal{Y}_k, \\ i=1,2,\ldots,p}} \left|\bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right) - f(\boldsymbol{Y}_i)\right| > \eta'\Delta_k^2 \right\} \\
&\leq \sum_{i=1}^{p} \mathbb{P}\left\{ \left|\bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right) - f(\boldsymbol{Y}_i)\right| > \eta'\frac{\Delta_k^2}{p} \right\} \\
&= \sum_{i=1}^{p} \mathbb{E}\left[\mathbb{P}\left\{ \left|\bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right) - f(\boldsymbol{Y}_i)\right| > \eta'\frac{\Delta_k^2}{p} \,|\, \mathcal{F}_k \right\}\right]. \qquad (4.14)
\end{aligned}$$

Now using (4.14) and arguments identical to those leading to (3.12) in the proof of Lemma 11 (and the sample size expression in Step 2 (b) of Algorithm 6), we can then say for large enough k and some $\delta > 0$ that

$$\begin{aligned}
\mathbb{P}\left\{Err_1 > \eta'\Delta_k^2\right\} &\leq p^3(\eta'\Delta_k^2)^{-2}(1+\delta)\kappa_{ias}^2\Delta_k^4\lambda_k^{-1} \\
&\leq p^3\eta'^{-2}(1+\delta)\kappa_{ias}^2\lambda_k^{-1}. \qquad (4.15)
\end{aligned}$$

Since λ_k is chosen so that $k^{1+\varepsilon} = \mathcal{O}(\lambda_k)$ for some $\varepsilon > 0$, we see that (4.15) implies that $\mathbb{P}\left\{\left(\bar{F}\left(\boldsymbol{Y}_i, \tilde{N}(\boldsymbol{Y}_i)\right) - f(\boldsymbol{Y}_i)\right) > \eta'\Delta_k^2 \text{ i.o.}\right\} = 0$ by Borel-Cantelli. This in turn implies from (4.14) that

$$\mathbb{P}\left\{\left|M_k\left(\tilde{\boldsymbol{X}}_{k+1}\right) - m_k\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right| \geq \eta'\Delta_k^2 \text{ i.o.}\right\} = 0. \qquad (4.16)$$

Next we analyze the deterministic model error probability $\mathbb{P}\left\{Err_2 \geq \eta'\Delta_k^2\right\}$ appearing on the right-hand side of (4.13). Since we know from the postulates of the theorem that

$m_k(\boldsymbol{z})$ is a $\left(\kappa_{ef}, \kappa_{eg}\right)$-fully-linear model of f on $\mathcal{B}(\boldsymbol{X}_k; \Delta_k)$, implying that if η_1 is chosen so that $\eta' = \frac{1}{3}(1-\eta_1)\kappa_{md} > \kappa_{ef}$, we have

$$\mathbb{P}\left\{\left|m_k\left(\tilde{\boldsymbol{X}}_{k+1}\right) - f\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right| \geq \eta'\Delta_k^2 \text{ i.o.}\right\} = 0. \tag{4.17}$$

Finally, we analyze the stochastic sampling error probability $\mathbb{P}\left\{Err_3 \geq \eta'\Delta_k^2\right\}$ appearing on the right-hand side of (4.13). Using arguments identical to those leading to (3.12) in the proof of Lemma 11, it is seen that

$$\mathbb{P}\left\{\left|f\left(\tilde{\boldsymbol{X}}_{k+1}\right) - \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, \tilde{N}_{k+1}\right)\right| \geq \eta'\Delta_k^2 \text{ i.o.}\right\} = 0. \tag{4.18}$$

Conclude from (4.16), (4.17), and (4.18) that each of errors Err_1, Err_2 and Err_3 exceeding $\eta'\Delta_k^2$ infinitely often has probability zero and the assertion of Theorem 11 holds. ■

Lemma 18. *For any sample path $\omega \in \Omega$ if there exists a constant $\kappa_{lbg}(\omega) > 0$, such that $\|\nabla M_k(\boldsymbol{X}_k(\omega))\| \geq \kappa_{lbg}(\omega)$ for large enough k, then there exists a constant $\kappa_{lbd}(\omega) > 0$ such that $\Delta_k(\omega) \geq \kappa_{lbd}(\omega)$ for large enough k.*

Proof. Let $K_g(\omega) > 0$ be such that $\|\nabla M_k(\boldsymbol{X}_k(\omega))\| \geq \kappa_{lbg}$ if $k > K_g(\omega)$. From Theorem 11, we let $K_s(\omega) > 0$ be such that $K_s(\omega) - 1$ is the last unsuccessful iteration, that is, k is a successful iteration if $k \geq K_s(\omega)$. Then $\tilde{\Delta}_k(\omega) > \Delta_{k-1}(\omega)$ for all $k \geq K_s(\omega)$. For $k \geq \max\left\{K_g(\omega), K_s(\omega)\right\} + 1$, consider the two cases below when Algorithm 6 starts.

CASE 1 ($\tilde{\Delta}_k(\omega) \geq \mu\|\nabla M_k(\boldsymbol{X}_k(\omega))\|$): Since $\tilde{\Delta}_k(\omega) \geq \mu\|\nabla M_k(\boldsymbol{X}_k(\omega))\|$, the inner loop of Algorithm 6 is executed, implying that

$$\Delta_k(\omega) \geq \beta\|\nabla M_k(\boldsymbol{X}_k(\omega))\| \geq \beta\kappa_{lbg}(\omega).$$

CASE 2 ($\tilde{\Delta}_k(\omega) < \mu\|\nabla M_k(\boldsymbol{X}_k(\omega))\|$): In this scenario, the inner loop of Algorithm 6 is not executed, implying that $\Delta_k(\omega) = \tilde{\Delta}_k(\omega) = \gamma_1\Delta_{k-1}(\omega)$ meaning that the trust-region radius expands from the previous iteration.

CASE 1 and CASE 2 iterations are mutually exclusive and collectively exhaustive. CASE 1 iterations imply, under the assumed postulates, that $\Delta_k(\omega) \geq \beta\kappa_{lbg}(\omega)$; CASE 2 iterations result in an expanded trust-region radius. Conclude from these assertions that $\Delta_k(\omega) \geq \min\left\{\beta\kappa_{lbg}(\omega), \Delta_{\max\left\{K_g(\omega), K_s(\omega)\right\}}\right\}$. ■

We are now fully setup to demonstrate that ASTRO-DF's iterates converge to a first-order critical point with probability one.

Theorem 12. *Suppose f is continuously differentiable and bounded from below. Let Assumptions 5, 6 hold. Then $\|\nabla f(\boldsymbol{X}_k)\| \xrightarrow{wp1} 0$ as $k \to \infty$.*

Proof. Lemma 18 and Theorem 10 together imply that $\liminf_{k\to\infty} \|\nabla M_k(\boldsymbol{X}_k)\| = 0$ almost surely. This, along with Lemma 17, implies that $\liminf_{k\to\infty} \|\nabla f(\boldsymbol{X}_k)\| = 0$ almost surely.

We now use the lim-inf convergence just established to prove the assertion of Theorem 12 through a contrapositive argument. Note that the following results hold for any given sample path $\omega \in \Omega$, but for the sake of readability we remove ω.

Suppose we have a subsequence of iterations $\{t_i\}$ such that $\|\nabla f(\boldsymbol{X}_{t_i})\| > 3\varepsilon$ for some $\varepsilon > 0$. Due to the lim-inf type convergence just established, for every t_i there exists $\ell_i = \ell(t_i)$ that is the first iteration after t_i with $\{\ell_i\}$ such that $\|\nabla f(\boldsymbol{X}_{\ell_i})\| < 2\varepsilon$. Therefore if we let $\mathcal{K}_i = \{k : t_i \le k \le \ell_i\}$, then $\|\nabla f(\boldsymbol{X}_k)\| \ge 2\varepsilon$ for all $k \in \mathcal{K}_i$. Choose i large enough so that for all $k \in \mathcal{K}_i$

(i) $\hat{\rho}_k \ge \eta_1$ (only successful iterations),

(ii) $\|\nabla M_k(\boldsymbol{X}_k)\| \ge 2\varepsilon$ (model gradient close to the function gradient),

(iii) $\Delta_k \le \kappa_{bhm}^{-1}\varepsilon$ (trust-region radius small), and

(iv) $|\bar{F}(\boldsymbol{X}_k, N_k) - f(\boldsymbol{X}_k)| \le 8^{-1}\eta_1\varepsilon\kappa_{fcd}\Delta_k$ (simulation error small).

where we use Theorem 11, Lemma 17, Theorem 10, and Lemma 11 (in which we choose $c_f = 8^{-1}\eta_1\varepsilon\kappa_{fcd}\Delta_k$) respectively.

Then we have

$$\begin{aligned}
\bar{F}(\boldsymbol{X}_{k+1}, N_{k+1}) - \bar{F}(\boldsymbol{X}_k, N_k) &\le \bar{F}(\boldsymbol{X}_{k+1}, N_{k+1}) - \bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) \\
&\quad + \left|\bar{F}\left(\boldsymbol{X}_k, \tilde{N}_k\right) - f(\boldsymbol{X}_k)\right| + |f(\boldsymbol{X}_k) - \bar{F}(\boldsymbol{X}_k, N_k)| \\
&\le -\eta_1 \frac{\kappa_{fcd}}{2}\varepsilon\Delta_k + \eta_1 \frac{\kappa_{fcd}}{4}\varepsilon\Delta_k \\
&= -\eta_1 \frac{\kappa_{fcd}}{4}\varepsilon\Delta_k, \qquad (4.19)
\end{aligned}$$

and as a result

$$\Delta_k \leq \frac{-4}{\eta_1 \kappa_{fcd} \varepsilon} \left(\bar{F}\left(\boldsymbol{X}_{k+1}, N_{k+1} \right) - \bar{F}\left(\boldsymbol{X}_k, N_k \right) \right),$$

for all $k \in \mathcal{K}_i$. It follows that

$$\begin{aligned} \| \boldsymbol{X}_{\ell_i} - \boldsymbol{X}_{t_i} \| &\leq \sum_{j \in \mathcal{K}_i} \left\| \boldsymbol{X}_{j+1} - \boldsymbol{X}_j \right\| \leq \sum_{j \in \mathcal{K}_i} \Delta_j \\ &\leq \frac{-4}{\eta_1 \kappa_{fcd} \varepsilon} \sum_{j \in \mathcal{K}_i} \bar{F}\left(\boldsymbol{X}_{j+1}, N_{j+1} \right) - \bar{F}\left(\boldsymbol{X}_j, N_j \right) \\ &\leq \frac{-4}{\eta_1 \kappa_{fcd} \varepsilon} \left(\bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i} \right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i} \right) \right), \end{aligned}$$

and hence deduce that $\| \boldsymbol{X}_{\ell_i} - \boldsymbol{X}_{t_i} \| \xrightarrow{wp1} 0$ since using (4.19)

$$\begin{aligned} \bar{F}\left(\boldsymbol{X}_{\ell_i}, N_{\ell_i} \right) - \bar{F}\left(\boldsymbol{X}_{t_i}, N_{t_i} \right) &= \sum_{j \in \mathcal{K}_i} \bar{F}\left(\boldsymbol{X}_{j+1}, N_{j+1} \right) - \bar{F}\left(\boldsymbol{X}_j, N_j \right) \\ &\leq -\eta_1 \frac{\kappa_{fcd}}{4} \varepsilon \left(\ell_i - t_i \right) \max_{j \in \mathcal{K}_i} \Delta_j, \end{aligned}$$

in which $\max_{j \in \mathcal{K}_i} \Delta_j \xrightarrow{wp1} 0$ as $i \to \infty$ by Theorem 10. We also observe similar to the argument in (3.19) that $\left| f\left(\boldsymbol{X}_{\ell_i} \right) - f\left(\boldsymbol{X}_{t_i} \right) \right| \xrightarrow{wp1} 0$.

Knowing the function gradient $\nabla f(\boldsymbol{x})$ is Lipschitz continuous, we conclude as $i \to \infty$ we must have $\| \nabla f\left(\boldsymbol{X}_{\ell_i} \right) - \nabla f\left(\boldsymbol{X}_{t_i} \right) \| \xrightarrow{wp1} 0$; but this indicates that $\| \nabla f\left(\boldsymbol{X}_{\ell_i} \right) - \nabla f\left(\boldsymbol{X}_{t_i} \right) \| < \varepsilon$ almost surely for large enough i, contradicting the definition of t_i and ℓ_i, hence proving the assertion of the Theorem. ■

5. IMPLEMENTATION HEURISTICS AND NUMERICAL EXPERIENCE WITH ASTRO-DF

ASTRO and ASTRO-DF are adaptive sampling algorithms that globally converge to a critical point with probability one. Such consistency results provide only a minimum level of guarantee in that they, unfortunately, do not assure finite-time efficiency of the proposed algorithms. Our extensive numerical experience shows that certain heuristics, particularly within ASTRO-DF, are especially important to ensure stable performance. This chapter describes such heuristics in an organized way. We also report numerical results from the implementation of ASTRO-DF on low to moderate dimensional problems. For numerical illustration, we use suite of problems similar to that used in experiments with ASTRO in Chapter 3.

5.1 Key Implementation Heuristics

As noted, notwithstanding the global convergence proofs, certain implementation heuristics appear to be important to ensure ASTRO-DF's good finite-time performance. For example, the choice of interpolation points in the model construction step, trust-region management details, the manner in which historical iterates are re-used in the model construction step, and the specific methods used for updating iterates, all affect ASTRO-DF's functioning. In what follows we detail five such aspects listed here in order of importance.

1. Choosing the set of design points $\mathcal{Y}_k^{(j_k)}$ for model construction in Algorithm 6 (Section 5.1.1).

2. Choosing the algorithm parameters to enhance practical efficiency (Section 5.1.2).

3. Pre-processing to identify the initial point $\boldsymbol{X}_0$ and the initial trust-region radius Δ_0 (Section 5.1.3).

4. Solving the trust-region sub-problem (Section 5.1.4).

5. Choosing an iterate subsequent to a successful iteration (Section 5.1.5).

The almost sure convergence results of ASTRO-DF are not affected by the choices suggested in subsections 5.1.2, 5.1.3, and 5.1.4; they are, however, affected by our proposals in subsections 5.1.1 and 5.1.5. Specifically, the convergence proofs for ASTRO-DF in [63] require the interpolation set $\mathcal{Y}_k$ to remain certifiably fully poised in every iteration. The implementation of ASTRO-DF that we discuss here relaxes this stipulation, thereby threatening convergence. We speculate that the convergence proofs in [63] could be generalized to subsume the relaxation we propose, by stipulating full linearity only on a subsequence of iterations.

5.1.1 Choosing Design Points for the Model Construction Step

The quality of models constructed within ASTRO-DF crucially affects ASTRO-DF's performance. There is, however, a natural tension between constructing accurate models and the fast convergence of ASTRO-DF. Constructing accurate models entails identifying a "well dispersed" set of design points and then sampling adequately at each of these identified points. And, the need to identify a well-dispersed set of points means that past iterates, which are usually highly correlated, can only be used sparingly, if at all. In what follows, we detail a proposal that balances the competing need for well dispersed points and the inclusion of past algorithm iterates into the design set. (What we detail here applies toward executing Step 3 in Algorithm 6.)

Our proposal to identify the $p = (d+1)(d+2)/2$ design points needed to construct a full set $\mathcal{Y}_k$ involves the following two steps.

(i) Identify a well dispersed subset, defined in a certain rigorous sense, from amongst the already observed points for inclusion into $\mathcal{Y}_k^{(j_k)}$; and

(ii) if the cardinality of the set identified in (i) is less than p, identify additional well dispersed points to complete the full set $\mathcal{Y}_k^{(j_k)}$.

The steps for choosing design points towards constructing a model are listed in Algorithm 7, requiring the TR radius and model gradient norm in the latest iteration of the model construction loop of Algorithm 6, as well as the history of all visited points.

For (i) (Steps 1–11), a convenient method for the identification of "poised" points, denoted as $\mathcal{Y}_{init}$ is through the maximization of Lagrange functions, as detailed in Algorithm 6.2 in [44, p. 95] , where the Lagrange functions are first reset to the normal basis of a quadratic interpolation model, that is,

$$\Phi(\mathbf{z}) := (\phi_1, \phi_2, \ldots, \phi_p) = \left(1, z^1, z^2, \ldots, z^d, \frac{1}{2}(z^1)^2, z^1 z^2, \ldots, \frac{1}{2}(z^2)^2, \ldots, \frac{1}{2}(z^d)^2\right),$$

and then updated according to the new design points added to the set. We use the COBYLA (Constrained Optimization BY Linear Approximation) procedure [64] for this purpose. Moreover we identify, from amongst all points visited by ASTRO-DF and lying within the current trust-region, a subset of points such that the distance between any two points included within the subset is at least $\theta\tilde{\Delta}_k w^{j_k-1}$, $\theta \in (0,1)$. We call this subset $\mathcal{Y}_{pool}$. Then, for re-using purposes, the *equivalent* points in $\mathcal{Y}_{pool}$ to those in $\mathcal{Y}_{init}$ are considered for inclusion in the sample set. An *equivalent* of a point is defined as the closest one of $\mathcal{Y}_{pool}$ with the distance of at most $\theta'\tilde{\Delta}_k w^{j_k-1}$, $\theta' < \theta$, to the point.

The current iterate (and centre of the trust-region) is always included within the sample set. In the unlikely event that the cardinality of the subset identified in (i) is equal to p, we have successfully identified the complete set $\mathcal{Y}_k^{(j_k)}$. Otherwise, as part of (ii) (Step 12–22), we search for additional points that would complete the set $\mathcal{Y}_k^{(j_k)}$ while satisfying the minimum separation $\theta\tilde{\Delta}_k w^{j_k-1}$ between all pairs of points. The criticality alert, triggered if the most recent model gradient norm is small, enforces high quality models by choosing the remainder of the sample set from new points in $\mathcal{Y}_{init}$. Albeit with no evidence of criticality, the additional points are selected from old points in $\mathcal{Y}_{pool}$.

Algorithm 7 $[\mathcal{Y}_k^{(j_k)}]$=`SampleSelection`$(\tilde{\Delta}_k w^{j_k-1}, \boldsymbol{X}_k, \left\|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\right\|, \bigcup_{\ell=0}^{k-1}\bigcup_{t=1}^{j_\ell}\mathcal{Y}_\ell^{(t)})$

Require: *Parameters from* `AdaptiveModelConstruction`: TR radius $\tilde{\Delta}_k w^{j_k-1}$, current iterate $\boldsymbol{X}_k$, current model gradient norm $\left\|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\right\|$, and previous sample sets $\bigcup_{\ell=0}^{k-1}\bigcup_{t=1}^{j_\ell}\mathcal{Y}_\ell^{(t)}$.

Parameters specific to `SampleSelection`: minimum separation constant $0<\theta<1$, equivalence constant $0<\theta'<\theta$ and criticality constant ε_g.

Find a new poised set $\mathcal{Y}_{init}=\{\boldsymbol{X}_k,\boldsymbol{Y}_2,\boldsymbol{Y}_3,\cdots,\boldsymbol{Y}_p\}$ using Lagrange polynomials. Let $\mathcal{Y}_k^{(j_k)}=\{\boldsymbol{X}_k\}$, $\mathcal{J}=\varnothing$ and $\mathcal{Y}_{pool}=\varnothing$.

for all $\boldsymbol{y}\in\bigcup_{t=1}^{j_{k-i}}\mathcal{Y}_{k-i}^{(t)}\cap\mathscr{B}\left(\boldsymbol{X}_k;\tilde{\Delta}_k w^{j_k-1}\right)$, $i=1,2,\cdots,k$ **do** {Check the visited points, starting from the most recent.}

  if $\boldsymbol{y}\notin\bigcup_{\mathbf{z}\in\mathcal{Y}_{pool}}\mathscr{B}\left(\mathbf{z};\theta\tilde{\Delta}_k w^{j_k-1}\right)$ **then** {If not within minimum separation with other points, add to the pool.}

    Set $\mathcal{Y}_{pool}=\mathcal{Y}_{pool}\cup\{\boldsymbol{y}\}$.

  end if

end for

for all $\boldsymbol{Y}_i\in\mathcal{Y}_{init}$, $i=2,3,\cdots,p$ **do**

  if $\boldsymbol{Y}_i':=\underset{\mathbf{z}\in\mathcal{Y}_{pool}\cap\mathscr{B}\left(\boldsymbol{Y}_i;\theta'\tilde{\Delta}_k w^{j_k-1}\right)}{\arg\min}\|\mathbf{z}-\boldsymbol{Y}_i\|_2$ exists, **then** {Select those points of $\mathcal{Y}_{init}$ that have equivalents in $\mathcal{Y}_{pool}$.}

    Set $\mathcal{Y}_k^{(j_k)}=\mathcal{Y}_k^{(j_k)}\cup\{\boldsymbol{Y}_i'\}$ and $\mathcal{J}=\mathcal{J}\cup\{i\}$. {Place their closest equivalent in $\mathcal{Y}_k^{(j_k)}$.}

  end if

end for

if $\left|\mathcal{Y}_k^{(j_k)}\right|<p$, **then** {If the sample set does not have p points in it choose the rest based on criticality.}

  if $\left\|\nabla M_k^{(j_k)}(\boldsymbol{X}_k)\right\|<\varepsilon_g$, **then** {Alert if the current TR is in critical region, implying poised-ness must be maintained.}

    for all $i=2$ to p and $i\notin\mathcal{J}$ **do** {Choose the remainder of the points from the new points in the poised set $\mathcal{Y}_{init}$.}

      $\mathcal{Y}_k^{(j_k)}=\mathcal{Y}_k^{(j_k)}\cap\{\boldsymbol{Y}_i\}$.

    end for

  else

    while $\left|\mathcal{Y}_k^{(j_k)}\right|\neq p$ **do** {Choose the remainder of the points from the points in $\mathcal{Y}_{pool}$.}

      Set $\boldsymbol{Y}_{best}:=\underset{\mathbf{z}\in\mathcal{Y}pool}{\arg\max}\sum_{\boldsymbol{y}\in\mathcal{Y}_k^{(j_k)}}\|\mathbf{z}-\boldsymbol{y}\|_2$. {Choose the point with largest cumulative distance to all members of $\mathcal{Y}_k^{(j_k)}$.}

      Set $\mathcal{Y}_{pool}=\mathcal{Y}_{pool}\backslash\{\boldsymbol{Y}_{best}\}$ and $\mathcal{Y}_k^{(j_k)}=\mathcal{Y}_k^{(j_k)}\cup\{\boldsymbol{Y}_{best}\}$

    end while

  end if

end if

5.1.2 Choosing Algorithm Parameters

The parameters in ASTRO-DF fall into two categories: general trust-region parameters and adaptive sampling parameters. We now discuss the choice and effect of these parameters in broad terms. It must be understood that, just as in much of algorithm design, there is a certain subjectivity in the choice of algorithm parameters. Convergence theory frequently leaves open a wide range of possibilities for algorithm parameter choice, which must then be narrowed through empirical experience. In accordance with the philosophy that a well designed algorithm implementation should not expect a user to choose algorithm parameters, we suggest default values for all parameters we discuss here. All results reported in the section on numerical results were obtained using default parameter settings.

General Trust-Region Parameters

The general parameters in the trust-region framework include $\eta_1, \gamma_1, \gamma_2, \beta, \mu$ and w. For all experiments that we report in section 5.2 we have used the following default parameter settings: $\gamma_1 = 1.2, \gamma_2 = 0.9, \beta = 0.5, \mu = 2.0$, and $w = 0.9$. In what follows, we provide some intuition on each of these parameters.

The parameter η_1 is a threshold for sufficient reduction in the function estimated value when moving from the current iterate $\boldsymbol{X}_k$ to the candidate solution $\tilde{\boldsymbol{X}}_{k+1}$. Large values of η_1 make the sufficient reduction condition more stringent, stipulating higher model accuracies; small values of η_1 make the sufficient reduction condition more lax, allowing for explorative moves. It is worth noting that the ASTRO-DF algorithm as listed in this paper includes only a sufficient decrease condition. By contrast, the deterministic TRO-DF algorithm proposed by [46] includes an additional constant η_0 that is meant to allow a *simple decrease* condition in addition to the *sufficient decrease* condition.

ASTRO-DF accepts the candidate point as the next iterate when the reduction predicted by the model exceeds the estimated reduction by a factor η_1; such acceptance then amounts to a tacit acknowledgement that the newly constructed model can perhaps adequately represent the objective function in a region with a radius that is larger than the incumbent

trust-region radius. The parameter γ_1 controls the extent of such increase in the trust-region radius post candidate acceptance. Conversely, when a candidate point is not accepted due to the predicted decrease being too small a fraction of the estimated decrease, ASTRO-DF reposes less faith in the model, leading to contraction of the trust-region radius. The extent to which such reduction happens is controlled by the parameter γ_2. The other contraction factor is w in the inner loop of Algorithm 6. Small values for both of these contraction factors can result in changes in the model as a result of changes in the sample set, and a corresponding faster consumption of the simulation budget.

The parameter β, along with the parameter μ, enforces the model gradient to be in lock-step with the trust-region radius. Algorithm 6 continues to be executed until a model of specified quality is constructed in a trust-region whose radius does not exceed the product of μ and the model gradient. A large value of μ thus allows for greater lenience, resulting in a poorer model. On the other hand, the parameter β is used to prevent the trust-region radius resulting from the execution of Algorithm 6 from becoming too small. Towards satisfying the stipulated lock-step, Algorithm 6 repeatedly shrinks the trust-region radius using the constant factor w, thereby introducing the possibility of a final trust-region with a radius that is very small. The parameter β prevents this possibility. As an example, if the parameter μ is set equal to β, the size of the trust-region that exits Algorithm 6 is strictly in lock-step with the product of β and the model gradient norm.

Furthermore the default parameter settings in the sample selection heuristic in our experiments are $\theta = 0.2, \theta' = 0.05$, and $\varepsilon_g = 10$, chosen in an ad-hoc manner.

Adaptive Sampling Parameters

Whenever the objective function needs to be estimated at a specified design point, ASTRO-DF has to make a decision on how much sampling effort needs to be exerted for estimation. One of the salient features of ASTRO-DF is that decisions on the extent of sampling are, at least to a certain degree, adaptive. Specifically, the sampling rules in expressions (4.1) and (4.2) control ASTRO-DF's sampling rate with the two parameters

κ_{oas} and κ_{ias} corresponding to the adaptive sampling constants for the outer-loop and the inner loops respectively. Small values of κ_{oas} and κ_{ias} make ASTRO-DF trajectories appear deterministic due to increased sampling leading to reduced sampling error. On the other hand, large values of κ_{oas} and κ_{ias} imply less sampling and increased variability in sample paths. The parameters κ_{oas} and κ_{ias} far more affect the convergence rate of ASTRO-DF than whether or not ASTRO-DF converges.

The other important adaptive sampling parameter is the inflation factor λ_k. This parameter implicitly sets a lower bound for the sample size during each iteration. As specified in the inputs of Algorithm 5, the sequence $\{\lambda_k\}$ should satisfy $k^{(1+\varepsilon)} = O(\lambda_k)$, that is, λ_k is roughly of the same order as the iteration number. (we use $\varepsilon = 10E-4$). Our extensive numerical experience indicates that the lower bound sample size imposed through the sequence $\{\lambda_k\}$ is rarely binding, especially as ASTRO-DF's iterates approach a stationary point. This is consistent with what has been predicted by theory in other contexts.

In all experiments described in section 5.2 we impose a large number for the inner and outer loop sampling constants ($\kappa_{oas} = \kappa_{ias} = 10^3$) to enable more exploration throughout the search.

5.1.3 Pre-processing

Like any non-linear optimization algorithm, the choice of initial values, specifically, the initial guess $\boldsymbol{X}_0$ and the initial trust-region radius Δ_0, affect ASTRO-DF's performance. Accordingly, we have found it expedient to undertake a certain pre-processing step aimed at identifying good values for the initial guess x_0 and the starting trust-region radius Δ_0. With a fixed small budget we run ASTRO-DF with a vector of random initial points and a vector of random initial trust-region radii, giving each combination of the initial point and initial trust-region radius the same share of the pre-processing simulation budget. The best combination of the initial point and trust-region radius are then selected based on the resulting relative reduction in the model gradient norm.

5.1.4 Solving the TR Subproblem

The candidate point $\tilde{\boldsymbol{X}}_{k+1} = \boldsymbol{X}_k + \boldsymbol{S}_k$ that is the potential next incumbent solution in the search process comes from a constrained optimization problem in Step 3 of Algorithm 5. To find a good candidate solution $\boldsymbol{S}_k$, one can use the Cauchy step, which is the minimizer of the one-dimensional constrained optimization problem obtained by projecting the objective function along the negative gradient and constrained to the trust-region. The resulting step satisfies the Cauchy reduction in expression (2.2) that is required for the convergence of ASTRO-DF, with $\kappa_{fed} = 2$ for linear models and $\kappa_{fed} = 1$ for quadratic models. In such a case $\boldsymbol{S}_k$ is chosen as $\boldsymbol{S}_k = t_C \nabla M_k(\boldsymbol{X}_k)$, where

$$t_C = \underset{\alpha \in [0, \Delta_k]}{\arg\min}\, M_k(\boldsymbol{X}_k - \alpha \nabla M_k(\boldsymbol{X}_k)),$$

to satisfy a $\frac{1}{2}$-Cauchy decrease. (See Section 10.1 in [44] for additional details.)

Any routine to solve the TR subproblem that provides a candidate point with a higher reduction than that obtained through the Cauchy step is obviously preferred, although the resulting computational effort needs to be weighed against the reduction in objective function value. In the experiments reported in this paper we apply the constrained optimization method COBYLA [64].

5.1.5 Updating the Next Iterate

Given that several design points (along with their function estimates) are observed during the model construction and the TR subproblem stages, an important question is which amongst these should be chosen as the subsequent iterate in the event that the the sufficient reduction step is satisfied leading to a successful iteration. An obvious choice is the candidate point $\tilde{\boldsymbol{X}}_{k+1}$ in Algorithm 5 that led to a successful sufficient reduction step. An alternative, and one that we propose, is to instead choose the best from amongst all points in the design set $\mathcal{Y}_k$ that were observed during model construction. No such step needs to be performed after unsuccessful iterations. The following steps formally list the heuristic we propose for updating an iterate after a successful step.

(a) When the iteration is successful,

- if the candidate point does not yield the best function estimate, that is,

$$\min_{\boldsymbol{Y}\in\mathcal{Y}_k} \bar{F}\left(\boldsymbol{Y}, N\left(\boldsymbol{Y}\right)\right) \leq \bar{F}\left(\tilde{\boldsymbol{X}}_{k+1}, N\left(\tilde{\boldsymbol{X}}_{k+1}\right)\right)$$

  accept $\boldsymbol{Y}_{\min} := \arg\min_{\boldsymbol{Y}\in\mathcal{Y}_k} \bar{F}\left(\boldsymbol{Y}, N\left(\boldsymbol{Y}\right)\right)$ as the new iterate, and replace an existing point in $\mathcal{Y}_k$ (one located farthest from the new iterate) with the candidate point;

- else, that is, if the candidate point provides the best (lowest) estimated function value, update the next iterate to the candidate point.

Keep $\boldsymbol{X}_k$ in the set $\mathcal{Y}_{k+1}$ if it does not provide the worst (largest) estimated function value.

(b) When the iteration in unsuccessful: choose the current iterate as the iterate that starts the next iteration.

5.2 Numerical Experience and Discussion

In this section, we report ASTRO-DF's performance on 21 nonlinear sum of squares problems included in CUTEst [56] library of problems, that are the same problems we chose to experiment with ASTRO in Chapter 3. The dimensionality of the chosen problems varies from 2 to 8. The objective function for all problems in the set takes the form described in (3.24). The "noisy" observations are obtained by adding a normal random variable $\xi_i \sim \mathrm{N}\left(0, \sigma^2\right)$ to the sum, that is, $F_i(\boldsymbol{x}) = f(\boldsymbol{x}) + \xi_i$.

Similar to ASTRO, ASTRO-DF was executed until a specified simulation budget is exhausted. Suppose the specified simulation budget for ASTRO-DF is n_{total} and let $\boldsymbol{X}^i_{k_{max}}$ denotes the solution returned by the i-th execution of ASTRO-DF on a specific problem. If ASTRO-DF is executed m times, resulting in the m returned solutions $\boldsymbol{X}^i_{k_{max}}, i = 1, 2, \ldots, m$, the estimated expectation and estimated square-root variance of the *true* optimality gap and *true* gradient norms are given in (3.25) and (3.26). It is important to note that since the convergence theory for ASTRO-DF only guarantees convergence to a stationary point,

nothing can be said about the behavior of the true optimality gap even as the budget tends to infinity.

Table 5.1 and Table 5.2 suggest that ASTRO-DF exhibits consistent and steady progress toward a stationary point across different problems. As is evident from the reported values for small budgets, ASTRO-DF's iterates rapidly approach a stationary point during the initial iterations, with the transient phase being longer for higher dimensional problems. The progress then seems to slow down in the later iterations, when the $O(1/\sqrt{n})$ Monte Carlo rate appears to become effective. Also, unlike optimality gaps expressed using function values, the optimality gaps measured in terms of the gradient norm (reported in Table 5.2) sometimes exhibit jumps. This could be due to the existence of "cliffs" in the objective function terrain that cause ASTRO-DF to suddenly encounter new stationary regions. Consistent with what is generally known to be characteristic of derivative-free trust-region algorithms in the deterministic context, the behavior of ASTRO-DF is generally stable but somewhat slow.

Given all of the parameter settings and heuristics described above we experiment ASTRO-DF on 20 sample paths for several problems in each dimension (d=2,3,4,6,8). The optimality gap results (mean and standard deviation) for $\sigma = 1$ and $n_{max} = 25,000$ are summarized in Table 5.1. The optimality gap is the difference between the true function value after the last performed iteration, $f(\boldsymbol{X}_{k_{max}})$ and the true global optimal value of the function $f(\boldsymbol{x}^*)$. Note that the optimality gap is not necessarily expected to drop to zero as ASTRO-DF can converge to a local solution. In the table each column on the right represents the respective results when the simulation budget listed in the header of the column is consumed. We record the progress of the algorithm after 500, 1K, 5K, 10K, and 20K number of oracle calls. Also, the standard deviation values are shown in the parenthesis. The reductions in the optimality gap can be compared to the initial true function value, evaluated at the initial point obtained following the pre-processing steps (see subsection 5.1.3).

In all of the instances listed in the table a clear and fast drop in the early stages and slower drop in the later stages are evident, mostly in the mean optimality gaps and sometimes in their standard deviations. Sometimes at the early iterations (within the first 5000

simulation calls) the standard deviation is 0, that is possible when all the 20 sample paths of a problem are stuck in a point. Note that the design points are generated deterministically so in the first iterations they are very similar for all sample paths. But as the search evolves the sample paths are more likely to deviate from each other across problems, though this deviation is decreasing. This is due to the fact that in the later iterations the algorithm searches for reductions in the estimated function values that are much smaller in magnitude and the sampling error can be more misleading there. Besides that, higher dimension slows the reduction of the optimality gap due to rapid consumption of the simulation budget for a large number of design points required in the model construction. Nevertheless, the route to convergence is irrefutable.

Furthermore we look at $\|\nabla f(\boldsymbol{X}_{k_{max}})\|$, the true gradient after the last iteration is performed for each specified simulation budget, in the same 20 problems. The results are shown in Table 5.2. In this table we do not see a clean decreasing trend in the values as we did in Table 5.1. This implies that though the function is consistently decreasing, the function gradient norms undergo occasional jumps that can describe a cliff like region in the function. Note that this behavior is seen more often in the higher dimension test problems.

Figure 5.1 shows the optimality gap within one standard deviation interval, for the two-dimensional Rosenbrock function, which has one global and local minimum at (1,1). The quick drop in the optimality gap during the first several 1000's of simulation calls is discernible here as well. When the optimality gap becomes small, the sufficient reduction required to update the iterate becomes small and more precision in the estimated function value is instructed to capture a correct successful iteration. This enhances the sampling rate more quickly and therefore the number of simulation calls per iteration becomes large. As a result there are not many iterations and hence movements between 10,000 and 25,000 budget. This also explains why the one standard deviation interval width stays almost unchanged after 10,000 simulation budget (mean stays at 0.16 and standard deviation roughly stays at 0.14).

The quantile graphs of the true gradient and optimality gap for a number of the sum of squares problems are shown in Figures B.9-B.16. In these figures all the plots with solid

Table 5.1.
The estimated mean and standard deviation of the true optimality gap at a (random) returned solution of ASTRO-DF, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO-DF on each problem.

d	Problem Name	Initial Function Value	Simulation Budget (n_{total})				
			500	1000	5000	10000	20000
2	CUBE	1,664,640,225.00	166.8 (12.01)	115.85 (71.15)	2.75 (0.06)	2.73 (0.07)	2.73 (0.07)
	DENSCHNB*	83,493.00	223.26 (0.04)	83.19 (34.97)	0.22 (0.18)	0.08 (0.09)	0.06 (0.05)
	DENSCHNC*	17,053,704.00	82.03 (159.96)	3.57 (3.59)	3.54 (0.09)	3.55 (0.08)	3.55 (0.09)
	DENSCHNF	6,825,024.00	63.91 (0.00)	5.53 (0.39)	0.05 (0.06)	0.03 (0.03)	0.03 (0.02)
	ROSENBR	7,398,689.00	3,228.74 (4,072.37)	14.47 (13.28)	1.52 (1.54)	0.80 (1.02)	0.54 (0.79)
	S308	589,825.00	1.2 (0.00)	0.97 (0.13)	0.87 (0.07)	0.85 (0.06)	0.85 (0.07)
	SINEVAL	265,359.79	62.54 (0.05)	62.54 (0.05)	37.78 (4.73)	33.86 (6.34)	29.53 (9.21)
3	BEALE*	4,314,111,706.20	2,689.17 (0.00)	0.60 (0.00)	0.60 (0.00)	0.60 (0.05)	0.58 (0.02)
	DENSCHND	4,880,138,240.00	965,257.37 (215,883.46)	90,575.16 (28,683.84)	5,767.2 (8,664.57)	392.77 (502.34)	54.36 (107.3)
	DENSCHNE	57,857.00	127.92 (17.80)	74.80 (49.51)	5.60 (9.46)	1.41 (1.44)	1.05 (0.04)
	ENGVAL2*	1,654,165.00	285,405.96 (98,770.26)	142,720.6 (93,007.83)	6,278.57 (7,539.4)	531.84 (850.6)	59.99 (93.88)
	YFITU	7,532.36	7,532.36 (224.99)	7,532.36 (0.00)	7,532.36 (0.00)	2,980.49 (0.00)	644.23 (251.59)
4	BROWNDEN	1,109,286,386.27	4,273,984.56 (7,396,258.7)	875,723.70 (697,421.05)	135,392.29 (159,624.17)	14,707.83 (16,466.96)	1,404.82 (1,487.65)
	HELIX	62,036.77	21.20 (1.99)	5.75 (1.04)	4.47 (2.23)	1.89 (1.68)	0.79 (0.08)
	HIMMELBF	18,223,594.79	24,919.7 (116.5)	24,919.7 (116.5)	24,919.7 (116.5)	24,768.81 (402.6)	22,230.95 (969.11)
	KOWOSB	373.13	1.95 (0.00)	1.95 (0.00)	1.95 (0.00)	1.95 (0.00)	1.83 (0.28)
6	BIGGS6	11.40	11.40 (0.00)	11.12 (0.82)	10.14 (1.39)	8.99 (0.83)	8.84 (0.62)
	PALMER5C	17,604.47	7.98 (0.31)	7.98 (0.31)	4.03 (3.45)	0.46 (0.72)	0.19 (0.26)
8	PALMER6C*	234,351,624.62	16,079.32 (1,700.96)	6,896.38 (507.6)	6,137.43 (1,530.45)	5,802.91 (1,825.29)	5,660.21 (1,925.74)
	PALMER7C*	955,015,340.28	235,849.61 (46,021.58)	58,102.21 (30,039.18)	7,730.56 (10,391.39)	6,571.74 (9,003.0)	5,434.76 (7,314.98)
	PALMER8C*	276,298,016.54	13,460.1 (2,040.14)	6,359.53 (500.1)	5,573.64 (1,526.22)	5,242.04 (1,770.13)	5,047.68 (1,975.12)

lines on the left are the optimality gaps and all the plots with dashed lines on the right are the true gradient norms of the 25%, 50%, 75%, and 90% quantiles. One remark of all these plots is that the true gradient norms $\|\nabla f(\boldsymbol{X}_{k_{max}})\|$ converges to zero within the budget of 25,000 oracle calls.

In addition to its reasonable route to convergence, we are interested in the performance of ASTRO-DF under different levels of simulation error. This comparison is illustrated in Table 5.3 that lists the performance for the two-dimensional Rosenbrock function, with all of them starting at the same initial point and initial trust-region radius. In every level of noise, ASTRO-DF is run until the budget is exhausted, with the exception of the deterministic case (noise=0) in which the algorithm stops due to smaller than allowed trust-region. As expected the higher the variability in the simulation the more sampling at every point,

Table 5.2.
The estimated mean and standard deviation of the true gradient norm at a (random) returned solution of ASTRO-DF, as a function of the total simulation budget. The statistics were computed based on 20 independent runs of ASTRO-DF on each problem.

d	Problem Name	Initial Gradient Norm	Simulation Budget (n_{total})				
			500	1000	5000	10000	20000
2	CUBE	626,688,560.79	5,641.58 (215.93)	4,148.01 (2,133.23)	55.23 (38.39)	26.35 (18.69)	23.19 (24.82)
	DENSCHNB*	13,918.26	115.42 (0.63)	25.31 (14.66)	1.0 (0.32)	0.59 (0.41)	0.51 (0.21)
	DENSCHNC*	6,327,252.19	99.51 (180.17)	9.83 (20.25)	1.13 (1.08)	1.04 (0.75)	0.97 (0.75)
	DENSCHNF	1,228,253.22	1.91 (0.05)	42.89 (1.77)	4.43 (2.82)	3.04 (1.56)	2.77 (0.27)
	ROSENBR	1,741,683.78	2,215.21 (2,450.05)	95.27 (69.98)	11.19 (10.03)	3.67 (2.65)	2.9 (2.18)
	S308	104,267.14	1.07 (0.01)	0.74 (0.23)	0.57 (0.17)	0.56 (0.19)	0.52 (0.2)
	SINEVAL	45,109.93	418.43 (0.43)	418.43 (0.43)	23.04 (16.16)	15.12 (11.84)	14.22 (12.6)
3	BEALE*	1,702,889,243.95	6154.35 (0.00)	22.34 (0.00)	22.34 (0.00)	20.13 (12.26)	11.01 (8.69)
	DENSCHND	2,284,598,497.73	671,367.94 (102,757.39)	87,701.12 (35,407.02)	3,257.27 (3,186.4)	665.65 (676.33)	90.61 (159.8)
	DENSCHNE	14,880.03	29.08 (4.72)	22.55 (12.75)	3.10 (3.48)	0.76 (1.22)	0.36 (0.23)
	ENGVAL2*	1,328,958.48	157,877.43 (86,706.09)	91,714.81 (81,874.81)	7,955.82 (12,353.36)	1,018.2 (1,486.14)	138.61 (87.25)
	YFITU	6,698.93	6,698.93 (0.00)	6,698.93 (0.00)	6,698.93 (0.00)	4,761.87 (0.00)	4,116.24 (3,901.99)
4	BROWNDEN	144,497,890.99	985,280.87 (1,489,452.47)	294,066.81 (163,202.27)	79,602.49 (85,104.94)	22,860.1 (16,104.3)	4,640.47 (3,901.12)
	HELIX	4,989.97	185.34 (10.03)	45.38 (0.90)	38.23 (13.02)	27.41 (15.79	5.14 (2.42)
	HIMMELBF	1,207,473.61	4,465.48 (1,045.14)	4,465.48 (1,045.14)	4,465.48 (1,045.14)	4,003.38 (1,391.08)	1,773.33 (965.71)
	KOWOSB	80.65	1.24 (0.00)	1.24 (0.00)	1.24 (0.00)	1.24 (0.00)	1.12 (0.29)
6	BIGGS6	2.25	2.25 (0.00)	2.08 (0.50)	1.48 (0.86)	0.74 (0.53)	0.64 (0.41)
	PALMER5C	839.76	10.16 (0.19)	10.16 (0.19)	6.87 (3.82)	2.34 (2.33)	1.61 (1.27)
8	PALMER6C*	19,027,217.80	4,362.42 (986.26)	2,968.97 (2,881.55)	3,183.22 (4,001.57)	1,165.50 (1,508.67)	2,143.23 (1,720.74)
	PALMER7C*	80,645,963.34	146,744.86 (218,010.63)	177,031.03 (135,919.20)	8,473.06 (12,862.42)	7,679.50 (11,881.42)	4,241.71 (6,463.97)
	PALMER8C*	22,347,391.94	4,049.45 (928.6)	2,314.43 (5,186.12)	2,169.98 (3,082.67)	2,721.50 (2,709.13)	1,702.07 (2076.70)

and hence the less iterations and points visited. Nevertheless the optimality gap and model gradient norm for all cases seem promising.

Recall that the adaptive sampling ensures less sampling in the early iterations and more sampling in the later iterations. Due to increasing sampling rate the standard deviation of the resulting optimality gaps become smaller for the later iterations systematically. Figure 5.1 obtained for Rosenbrock problem illustrates this result.

We now would like to see if ASTRO-DF delivers the progress in the search that would be obtained in the absence of noise in the problem, whereby we can make conclusions about its efficiency. The efficiency of ASTRO-DF can be depicted by comparing the optimality gaps after certain number of design points are observed for a problem that entails several

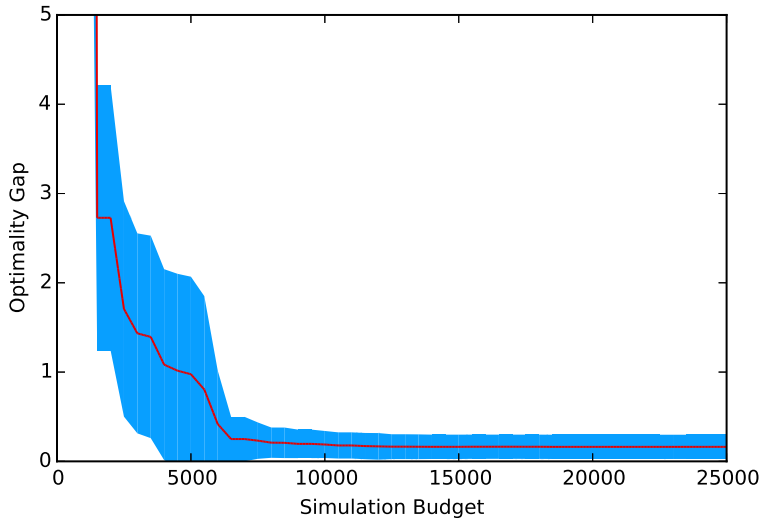


Figure 5.1. The one standard deviation interval from the mean of the optimality gap for the Rosenbrock function with $\sigma = 1$ at different levels of simulation budget. Reduction in the first 1500 simulation calls is from $7,398,689$ to 2.37 by average. After 13000 simulation calls the mean stays unchanged at 0.16.

levels of uncertainty. Note that instead of number of simulation oracle calls, here we record the progress based on the number of design points so that we can have a sound comparison with the deterministic problem. Figure 5.3 illustrates the results of this experiment. the optimality gap after certain number of points are observed. Since in the SO context much of the simulation budget is spent estimating the objective function value at each point, we will do the comparison based on the number of unique points that are used in the optimization procedure. The results are shown in Figure 5.3, that indicate high efficiency of ASTRO-DF for difference levels of noise.

With the reported numerical experience, the following discussions are noteworthy:

- In ASTRO-DF we do not define acceptable iteration as in the old deterministic TRO-DF [46], that is iterations in which the candidate point in accepted after simple decrease (typically known as $\eta_0 = 10^{-6}$). The reason is that with the simple decrease

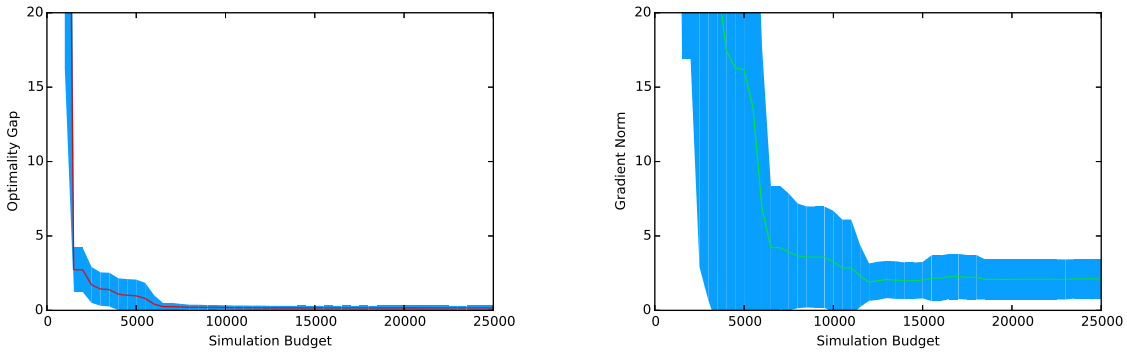


Figure 5.2. The one standard deviation interval from the mean of the optimality gap (on the left) and true function gradient norm (on the right) for the Rosenbrock function with $\sigma = 1$ at different levels of simulation budget. The variability in the function gradient is evidently more than the variability in the optimality gap.

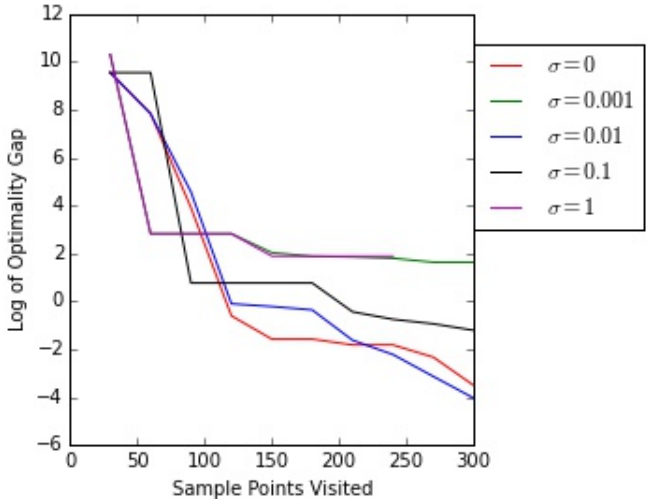


Figure 5.3. The $\log\left(\left|f\left(\boldsymbol{X}_{k_{max}}\right) - f\left(\boldsymbol{x}^*\right)\right|\right)$ after visiting several points, with the maximum simulation budget of 25,000.

the likelihood of false acceptance of a candidate point due to sampling error is much more that that in a sufficient decrease.

Table 5.3.
The number of iterations, number of points visited, final true function gradient and final optimality gap with 25,000 simulation budget, on different levels of simulation noise.

noise	iterations	points visited	$\|\nabla f(\boldsymbol{X}_{k_{max}})\|$	$\|f(\boldsymbol{X}_{k_{max}}) - f(\boldsymbol{x}^*)\|$
0	127	502	0.0131	0.000*
0.01	102	388	0.7057	0.0003
0.1	92	371	2.8821	0.1677
1	93	338	3.6573	0.1782
10	79	246	5.2054	0.0691

- ASTRO-DF suggests that the sampling rate at each point is roughly of $\mathcal{O}\left(\Delta_k^{-4}\right)$. In practice the adaptive sampling parameters can regularize this rate particularly when Δ_k is small. However we realize that when adaptive sampling parameters κ_{ias} and κ_{oas} are smaller, it results in larger replication size for each point, reduction in the stochastic error, and increase in the model accuracy. In short, with smaller adaptive sampling parameters the model gradient traces the true gradient with higher accuracy. However simulation budget is exhausted faster with fewer unique points observed. So in summary when the available simulation budget is relatively limited, the progress in the algorithm with lower κ_{ias} and κ_{oas} is not as good as when they are larger.

- Since in the theoretical convergence results of ASTRO-DF $\Delta_k \to 0$ wp1 the sampling rate starts low at the beginning of the history run and it grows larger towards the end. This is in essence what we regard as efficiency in the algorithm.

- Unlike the deterministic TRO-DF algorithm [46] we enter a model construction step in every iteration instead of optional iterations. The model gradient in the SO context is a random variable and hence we do not rely on it to the extent of neglecting the sampling requirement by optionally choosing to enter Algorithm 6. In other words

the sampling requirement must be fulfilled for each point that is going to contribute to the model construction.

- Despite the previous point, we are interested in practical aspect of so called criticality steps in the deterministic TRO-DF. In our model construction algorithm we choose poised points, collect "just enough " observations for every point, and interpolate so long as the tandem between Δ_k and $\|\nabla M_k(\boldsymbol{X}_{k_{max}})\|$ is ensured. But in the deterministic algorithm only when there is suspicion that we are near criticality, all the guarantees (full-linearity and lock-step) are required. We definitely relax full-linearity guarantees in the implemented version of ASTRO-DF by only using the fresh poised design set when the model gradient norm is alarmingly small. The general hope is that we rebuild a model that is based on history rather than new points. We might just require new observations at some of the old points to fulfill the their sampling requirement.

- In the event that the simulation budget is not limited or is not dominating the termination criteria (this almost means that the budget is unlimited), we suggest that at every iteration all fresh poised sets in $\mathcal{Y}_{init}$ be used, i.e. $q = 1$, as opposed to only using them when close to stationarity. In this case full-linearity of the model is maintained across iterations and the model accuracy is enhanced.

- The inflator λ_k that plays the role of the lower bound on the replication size namely is chosen to grow infinitesimally faster than the iteration number, i.e. $k^{(1+\varepsilon)} = \mathcal{O}(\lambda_k)$. This might seem to be a computationally burdensome requirement, but since as $k \to \infty$, λ_k becomes non-binding in the stopping rule expressions (4.1) and (4.2) it is not interfering with the efficiency. On the contrary setting $\lambda_k = 1$ can be harmful as it may lead to poorer function estimates in the later iterations and hence waste of effort by having false successful iterations.

- Often in the SO context, the simulation evaluations computationally overwhelm the standard numerical operations, such as those performed for model interpolation, constrained optimization and design set selection. So by using large adaptive sampling

parameters and also re-using the old points when forming the design set, we make substantial saving in the computation.

6. CONCLUDING REMARKS

Simulation optimization is widely used to solve many real-world problems, owing to the resilience and capacity to include details when using simulation models as opposed to simplified mathematical representations. As a result, devising SO algorithms with guarantees of global convergence to a local solution without requiring tremendous simulation budget are sought by many. The presence of sampling error in the SO context in addition to all common sources of error that exist in the deterministic context, can significantly mislead the search process. Controlling the sampling error becomes vital for the convergence and efficiency of the SO algorithms. However, in reality a major concern for an SO algorithm is the budget it consumes to reach a near optimal solution for a problem. Increased sampling, while providing accurate estimates of the objective function (and constraints), leads to increased computational burden. By contrast, reduced sampling is computationally efficient, but may harm the convergence of the SO algorithm. Therefore one intuitive proposition for SO algorithms is to adapt the Monte Carlo sample size at each iterate to the trajectory of the algorithm in that iteration. In other words larger sample size when there is evidence that the iteration is near a critical region is beneficial as making decisions about taking or rejecting a search step in such near-criticality iterations needs more care. Frugal (just enough) simulation expense for an iteration that is farther from a critical region increases the efficiency of the algorithm. Such techniques that adjust the simulation effort in an iteration to its distance from the solution are known as adaptive sampling techniques. The contribution of this thesis is to combine the adaptive sampling with one popular deterministic global optimization method called trust-region optimization that we refer to as DTRO.

Over the last decade or so, DTRO algorithms have enjoyed great attention and success in the deterministic context. They generate a sequence of iterates that converge to a first order critical point by strategically using a local model in a carefully managed neighbor-

hood of the incumbent solution and show promising performance in theory and practice. They also have advanced well into DTRO-DF in the area of derivative-free optimization, in which only function values are used to direct the search and no explicit gradient estimation is involved in the process. The common application of derivative-free optimization in the deterministic context is when the derivatives are either not available or expensive to acquire. We believe that developing analogous algorithms for the SO context, particularly due to the Monte Carlo settings for which derivative-free methods seem predominant, is quite relevant yet relatively unstudied. Therefore DTRO and DTRO-DF are worthy of further inquiry, particularly because the settings for which derivative-free trust-region methods are devised seem predominant within Monte Carlo contexts. Accordingly, we investigate theory and algorithms in which adaptive sampling rules are devised to systematically allocate the simulation budget to the iterates generated through derivative-based and derivative-free trust-region optimization for solving a range of multi-dimensional simulation optimization problems.

Consequently in this research we propose algorithms – adaptive sampling trust-region optimization algorithms (called ASTRO) and adaptive sampling trust-region optimization derivative-free algorithms (called ASTRO-DF) – that not only are convergent to a first-order critical point in rigorous probabilistic sense, but more importantly, gain practical efficiency through certain key steps related to adaptive sampling, model certification, and the careful balancing of sampling and model errors.

ASTRO, closely following the general framework of the DTRO, is provided for the settings with direct gradient observations. ASTRO involves determining the sample size with respect to the function and gradient information that is collectively observed via the Monte Carlo oracle. The almost sure convergence result follows from guaranteed Cauchy reduction in the optimization step and the sample sizes by which the likelihood of iterates wandering in an unbounded fashion due to mischance is forced to vanish. Implementation and numerical experiments of ASTRO show promising finite-time performance for a range of multi-dimensional unconstrained problems. Future research on the constrained optimization problems seems worthy. Also it is useful to explore whether ASTRO achieve the

Monte Carlo canonical convergence rate. ASTRO's theory and numerical results obtained and presented here are being prepared for publication.

ASTRO-DF, with the assumption of no readily discernible gradients, builds stochastic linear or stochastic quadratic interpolation models during the model-construction step while enforcing the model gradient, the true gradient and the trust-region radius to remain in tandem. Then the almost sure convergence follows as the sequence of the trust-region radii are ensured to converge to zero with probability one. The theoretical results of ASTRO-DF as presented here are accepted for publication in SIAM journal of Optimization (SIOPT). An online version of this paper is available at [63]. Future research focuses on a number of theoretical and practical issues within ASTRO-DF. For example other possibly more powerful model construction techniques such as regression or stochastic kriging [65] should be considered in place of interpolation models, especially alongside adaptive sampling. Ongoing research investigates this question, and it is our belief that the proof techniques that we currently present will carry over, albeit with some changes. Another interesting question involves the rate of convergence of ASTRO-DF. Convergence theory for ASTRO-DF dictates the asymptotic sampling rate to be $\mathcal{O}\left(\Delta_k^{-4}\right)$, where Δ_k is the incumbent trust-region radius. A similar requirement has been prescribed by two other recent prominent investigations [59, 66]. Is this sampling rate fundamental in any sense? Does the $\mathcal{O}\left(\Delta_k^{-4}\right)$ rate translate to the $\mathcal{O}(1/\sqrt{n})$ Monte Carlo canonical rate?

Though the adaptive sampling scheme determines stopping time for the sample sizes, questions arise regarding the interpolation model construction, initialization, updating and other details during the implementation of ASTRO-DF. We propose instructive heuristics that address these questions. This work is also accepted for publication in the proceedings of Winter Simulation Conference 2016. Having addressed ASTRO-DF's implementation and practicality, two crucial issues come to the surface:

(i) Unlike the DTRO-DF algorithm (see Algorithm 2), ASTRO-DF includes a model construction step in every iteration. It seems that such a stringent requirement can be relaxed without sacrificing convergence guarantees. In fact, the implementation of ASTRO-DF that we have used in Chapter 5 does just that by adding a critical-

ity step which stipulates that the model construction step be invoked only when the model gradient is sufficiently small. Such a simple rule improves practical efficiency; whether it preserves convergence is an open question.

(ii) Another unresolved question that is somewhat related to the remark in (i) relates to the manner of model construction. Specifically, how should the model construction step balance re-using already visited points with carefully placed new points within the trust-region? While using already visited points enhance efficiency by preserving simulation budget, they invariably result in poorer models because iterates visited by ASTRO-DF tend to be highly spatially correlated.

Finally we mention another open research question that focuses on a theoretical issue within ASTRO and ASTRO-DF. The slowly increasing sequence $\{\lambda_k\}$ ensures that the sample sizes within ASTRO-DF are forced to infinity asymptotically, and that the effects of infrequent spurious observations are limited. Our numerical experience strongly suggests that $\{\lambda_k\}$ is only rarely binding, and almost never so asymptotically. Can it be established that the probability of the lower bound sequence $\{\lambda_k\}$ being binding infinitely often is zero with probability one?

REFERENCES

REFERENCES

[1] N. B. Dimitrov and L. A. Meyers. Mathematical approaches to infectious disease prediction and control. INFORMS TutORials on Operations Research. INFORMS, 2010.

[2] S. Eubank, H. Guclu, V. S. A. Kumar, M. V. Marathe, A. Srinivasan, Z Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429:180–184, 2004.

[3] E. O. Nsoesie, R. J. Beckman, S. Shashaani, K. S. Nagaraj, and M. V. Marathe. A simulation optimization approach to epidemic forecasting. *PloS one*, 8(6):e67164, 2013.

[4] Y. T. Hou, Y. Shi, and H. D. Sherali. *Applied optimization methods for wireless networks*. Cambridge University Press, 2014.

[5] C. Osorio and M. Bierlaire. A surrogate model for traffic optimization of congested networks: an analytic queueing network approach. *Report TRANSP-OR*, 90825:1–23, 2009.

[6] O. Alagoz, A. J. Schaefer, and M. S. Roberts. Optimizing organ allocation and acceptance. In *Handbook of Optimization in Medicine*, pages 1–24. Springer, 2009.

[7] R. Pasupathy and S. G. Henderson. Simopt: a library of simulation optimization problems. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 4075–4085. IEEE, 2011.

[8] R. Pasupathy and S. G. Henderson. A testbed of simulation-optimization problems. In *Proceedings of the 38th conference on Winter simulation*, pages 255–263. Winter Simulation Conference, 2006.

[9] H. J. Kushner and G. G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, New York, NY., 2003.

[10] R. Pasupathy. On choosing parameters in retrospective-approximation algorithms for stochastic root finding and simulation optimization. *Operations Research*, 58:889–901, 2010.

[11] R. Pasupathy, P. W. Glynn, S. G. Ghosh, and F. S. Hahemi. How much to sample in simulation-based stochastic recursions? 2014. Under Review.

[12] S. Asmussen and P. W. Glynn. *Stochastic Simulation: Algorithms and Analysis*. Springer, New York, NY., 2007.

[13] S. Asmussen and P. W. Glynn. *Stochastic simulation: Algorithms and analysis*, volume 57. Springer Science & Business Media, 2007.

[14] S. Kim, R. Pasupathy, and S. G. Henderson. A guide to SAA. Frederick Hilliers OR Series. Elsevier, 2014.

[15] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466, 1952.

[16] J. Blum. Approximation methods which converge with probability one. *Annals of Mathematical Statistics*, 25(2):382–386, 1954.

[17] J. C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. *IEEE Transactions on Automatic Control*, 45:1839–1853, 2000.

[18] J. C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., Hoboken, NJ., 2003.

[19] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[20] K. Djeddour, A. Mokkadem, and M. Pelletier. On the recursive estimation of the location and of the size of the mode of a probability density. *Serdica Mathematics Journal*, 34:651–688, 2008.

[21] A. Mokkadem and M. Pelletier. A generalization of the averaging procedure: The use of two-time-scale algorithms. *SIAM Journal on Control and Optimization*, 49:1523, 2011.

[22] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[23] M. Broadie, D. M. Cicek, and A. Zeevi. General bounds and finite-time improvement for the kiefer-wolfowitz stochastic approximation algorithm. *Operations Research*, 59(5):1211–1224, 2011.

[24] R. Pasupathy and S. Ghosh. Simulation optimization: A concise overview and implementation guide. INFORMS TutORials. INFORMS, 2013.

[25] R. Y. Rubinstein and A. Shapiro. Optimization of static simulation models by the score function method. *Mathematics and Computers in Simulation*, 32:373–392, 1990.

[26] K. Healy and L. W. Schruben. Retrospective simulation response optimization. In B. L. Nelson, D. W. Kelton, and G. M. Clark, editors, *Proceedings of the 1991 Winter Simulation Conference*, pages 954–957. Institute of Electrical and Electronics Engineers: Piscataway, New Jersey, 1991.

[27] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on Stochastic Programming: Modeling and Theory*. SIAM, Philadelphia, PA, 2009.

[28] F. Bastin, C. Cirillo, and P. L. Toint. Convergence theory for nonconvex stochastic programming with an application to mixed logit. *Mathematical Programming*, 108:207–234, 2006.

[29] A. Ruszczynski and A. Shapiro, editors. *Stochastic Programming. Handbook in Operations Research and Management Science*. Elsevier, New York, NY., 2003.

[30] W. K. Mak, D. P. Morton, and R. K. Wood. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24:47–56, 1999.

[31] G. Bayraksan and D. P. Morton. Assessing solution quality in stochastic programs. *Mathematical Programming Series B*, 108:495–514, 2007.

[32] G. Bayraksan and D. P. Morton. A sequential sampling procedure for stochastic programming. *Operations Research*, 59(4):898–913, 2009.

[33] J. Royset and R. Szechtman. Optimal budget allocation for sample average approximation. *Operations Research*, 2011. Under Review.

[34] J. Luedtke and S. Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 2008. To Appear.

[35] T. S. Chow and H. Robbins. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *The Annals of Mathematical Statistics*, pages 457–462, 1965.

[36] M. Ghosh, N. Mukhopadhyay, and P. K. Sen. *Sequential Estimation*. Wiley Series in Probability and Statistics, 1997.

[37] S. G. Henderson and B. L. Nelson, editors. volume 13 of *Handbooks in Operations Research and Management Science: Simulation*. Elsevier, 2006.

[38] M. Ghosh and N. Mukhopadhyay. Sequential point estimation of the mean when the distribution is unspecified. *Communications in Statistics - Theory and Methods*, pages 637–652, 1979.

[39] M. Ghosh and N. Mukhopadhyay. Consistency and asymptotic efficiency of two stage and sequential estimation procedures. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 220–227, 1981.

[40] P. Glasserman. *Gradient Estimation Via Perturbation Analysis*. Kluwer, Netherlands, 1991.

[41] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, New York, NY., 2004.

[42] S. J. Wright and J. Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999.

[43] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region methods*, volume 1. Siam, 2000.

[44] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.

[45] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[46] A. R. Conn, K. Scheinberg, and L. N. Vicente. Global convergence of general derivative-free trust-region algorithms to first-and second-order critical points. *SIAM Journal on Optimization*, 20(1):387–415, 2009.

[47] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM Journal on Optimization*, 24(3):1238–1264, 2014.

[48] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, 2nd edition, 2006.

[49] K. Chang, L. J. Hong, and H. Wan. Stochastic trust-region response-surface method (strong)-a new response-surface framework for simulation optimization. *INFORMS Journal on Computing*, 25(2):230–243, 2013.

[50] B. D. Amos, D. R. Easterling, L. T. Watson, W. I. Thacker, B. S. Castle, and M. W. Trosset. Algorithm xxx: Qnstopquasinewton algorithm for stochastic optimization. 2014.

[51] F. Bastin, C. Cirillo, and Ph. L. Toint. An adaptive monte carlo algorithm for computing mixed logit estimators. *Computational Management Science*, 3(1):55–79, 2006.

[52] Sujin Kim and Jong-hyun Ryu. The sample average approximation method for multiobjective stochastic optimization. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 4021–4032. IEEE, 2011.

[53] G. M. Funk. *The probabilities of moderate deviations of U-statistics and excessive deviations of Kolmogorov-Smirnov and Kuiper statistics*. 1971.

[54] W. F. Grams and R. J. Serfling. Convergence rates for *u*-statistics and related statistics. *The Annals of Statistics*, 1(1):153–160, 1973.

[55] Y. S. Chow, H. Robbins, and H. Teicher. Moments of randomly stopped sums. *The Annals of Mathematical Statistics*, 36(3):789–799, 1965.

[56] N. IM. Gould, D. Orban, and P. L. Toint. Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[57] G. Deng and M. C. Ferris. Adaptation of the UOBYQA algorithm for noisy functions. In *Proceedings of the 38th conference on Winter simulation*, pages 312–319. Winter Simulation Conference, 2006.

[58] G. Deng and M. C. Ferris. Variable-number sample-path optimization. *Mathematical Programming*, 117(1-2):81–109, 2009.

[59] R. Chen, M. Menickelly, and K. Scheinberg. Stochastic optimization using a trust-region method and random models, 2015.

[60] J. Larson and S. C. Billups. Stochastic derivative-free optimization using a trust region framework. *Under review at Computational Optimization and Applications*, 2014.

[61] P. Billingsley. *Probability and Measure*. Wiley, New York, NY., 1995.

[62] S. C. Billups, J. Larson, and P. Graf. Derivative-free optimization of expensive functions with computational error using weighted regression. *SIAM Journal on Optimization*, 23(1):27–53, 2013.

[63] S. Shashaani, F. S. Hashemi, and R. Pasupathy. Astro-df: A class of adaptive sampling trust-region algorithms for derivative-free simulation optimization. *Under review at SIAM Journal of Optimization*, 2015.

[64] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*, Mathematics and Its Applications. Kluwer Academic, Dordrecht, 1994.

[65] B. Ankenman, B. L. Nelson, and J. Staum. Stochastic kriging for simulation metamodeling. *Operations research*, 58(2):371–382, 2010.

[66] J. M. Larson. *Derivative-Free Optimization of Noisy Functions*. PhD thesis, Department of Applied Mathematics, University of Colorado, Denver, CO, 2012.

APPENDICES

APPENDIX A

PYTHON CODE FOR ASTRO AND ASTRO-DF

The ASTRO main code is copied in the following:

```
from math import sqrt, ceil
import matplotlib.pyplot as plt
from numpy import asarray, outer, transpose, linalg, dot, identity, zeros, floor
from scipy.stats import norm
import ast
import julia
import os
import sys

def u16807d():
    global iseed
    u = 0.
    while round(u,6) <= 0 or u >= 1:
        iseed = (int(iseed)*16807) % 2147483647
        u = iseed / 2147483648.
    return u

def CUTEObjEval(jl, x, noise):
    global iseed, show_error
    f_dim = len(x)
    string = '['
    for i in range(f_dim):
        string += str(x[i])
        if i < f_dim-1:
            string += ';'
        else:
            string += ']'

    noise_added = 0
    if noise > 0:
        u = u16807d()
        noise_added = norm.ppf(u, loc=0, scale=sqrt(noise))
```

```
        obj = noise_added
    else:
        obj = 0
    jl.call('f = ufn(nlp,'+string+')')
    obj += jl.eval('f')
    return obj

def CUTEGradEval(jl, x, noise):
    f_dim = len(x)
    string = '['
    for i in range(f_dim):
        string += str(x[i])
        if i < f_dim-1:
            string += ';'
        else:
            string += ']'
    noise_added = []
    if noise > 0:
        for i in range(f_dim):
            u = u16807d()
            noise_added += [norm.ppf(u, loc=0, scale=sqrt(noise))]
        grad = noise_added
    else:
        grad = [0]*f_dim
    jl.call('g = ugr(nlp,'+string+')')
    grad = asarray(grad)+asarray(jl.eval('g'))
    return grad.tolist()

def AdaptiveSampling(jl, x, inflator, Delta, kappaf, kappag, gamma, noise, n_max, total_reps_maxes,
     candidate):
    global All_Y, All_Values, stop, total_reps, total_samples, iseed, total_reps_max, \
    points_rec, record_res_count, x_inc, optimum, record_res, grad_res

    current_reps_max = total_reps_maxes[record_res_count]
    if noise > 0:
        if candidate:
            n = pow(inflator/min(pow(Delta,2),pow(Delta,4)),gamma)
        else:
            n = pow(inflator/min(pow(Delta,2),pow(Delta,4)),gamma)
        n = max(2, ceil(n))
        threshold1, threshold2 = 2*[n_max - 1]

        while n <= max(threshold1, threshold2) and n <= n_max and not stop:
            ValuesUpdate(jl, x, n, noise, total_reps_maxes)
```

```
            value_index = list.index(All_Y,x)
            [f_hat, sigma2f_hat, g_hat, sigma2g_hat] = Stats(All_Values[value_index])
            threshold1 = inflator*sigma2f_hat/(pow(kappaf,2)*pow(Delta,4))
            if candidate:
                """ in practice use the same sapling rule is set 1 and 4"""
                threshold2 = inflator*sigma2g_hat/(pow(kappag,2)*pow(Delta,2))
            else:
                threshold2 = inflator*sigma2g_hat/(pow(kappag,2)*pow(Delta,2))

            if total_reps >= total_reps_max: stop = 1;
            n += 1
            if n > n_max: stop = 1;
        n -= 1
    else:
        n = 1
        """ if problem is deterministic do not use adaptive sampling, just evaluate once """
        if x not in All_Y:
            All_Y += [x]
            total_samples += 1
            All_Values += [[0., 0., 0., [0.]*len(x), [0.]*len(x)]]
            fobs_new = CUTEObjEval(jl, x, noise)
            gobs_new = CUTEGradEval(jl, x, noise)
            total_reps += 1
            if total_reps >= current_reps_max:
                if record_res_count < len(total_reps_maxes):
                    obj_determ = CUTEObjEval(jl, x_inc, 0)
                    grad_determ = CUTEGradEval(jl, x_inc, 0)
                    record_res[record_res_count] = round(abs(obj_determ - optimum),4)
                    grad_res[record_res_count] = round(linalg.norm(asarray(grad_determ)),4)
                    record_res_count += 1
                if record_res_count >= len(total_reps_maxes):
                    stop = 1
                else:
                    current_reps_max = total_reps_maxes[record_res_count]

            All_Values[len(All_Y)-1][0] += 1
            All_Values[len(All_Y)-1][1] += fobs_new
            All_Values[len(All_Y)-1][2] += pow(fobs_new, 2)
            for j in range(len(x)):
                All_Values[len(All_Y)-1][3][j] += gobs_new[j]
                All_Values[len(All_Y)-1][4][j] += pow(gobs_new[j], 2)
            f_hat = All_Values[len(All_Y)-1][1]
            g_hat = All_Values[len(All_Y)-1][3]
        else:
```

```
            f_hat = All_Values[list.index(All_Y, x)][1]
            g_hat = All_Values[list.index(All_Y, x)][3]

    return [n, f_hat, g_hat]

""" All_Y and All_Values are only updated here """
def ValuesUpdate(jl, x, n_new, noise, total_reps_maxes):

    global All_Y, All_Values, total_samples, total_reps, iseed, show_error, \
    points_rec, record_res_count, x_inc, optimum, stop, record_res, grad_res
    current_reps_max = total_reps_maxes[record_res_count]
    if x in All_Y:
        value_index = list.index(All_Y,x)
        n_old = All_Values[value_index][0]
    else:
        total_samples += 1
        value_index = len(All_Y)
        All_Y += [x]
        All_Values += [[0., 0., 0., [0.]*len(x), [0.]*len(x)]]
        n_old = 0

    if n_new > n_old:
        for i in range(int(n_old), int(n_new)):
            fobs_new = CUTEObjEval(jl, x, noise)
            gobs_new = CUTEGradEval(jl, x, noise)
            total_reps += 1
            if total_reps >= current_reps_max:
                if record_res_count < len(total_reps_maxes):
                    obj_determ = CUTEObjEval(jl, x_inc, 0)
                    grad_determ = CUTEGradEval(jl, x_inc, 0)
                    record_res[record_res_count] = round(abs(obj_determ - optimum),4)
                    grad_res[record_res_count] = round(linalg.norm(asarray(grad_determ)),4)
                    record_res_count += 1
                if record_res_count >= len(total_reps_maxes):
                    stop = 1
                else:
                    current_reps_max = total_reps_maxes[record_res_count]
            All_Values[value_index][0] += 1
            All_Values[value_index][1] += fobs_new
            All_Values[value_index][2] += pow(fobs_new, 2)
            for j in range(len(x)):
                All_Values[value_index][3][j] += gobs_new[j]
                All_Values[value_index][4][j] += pow(gobs_new[j], 2)
```

```
def Stats(all_values):
    [n, sum1_f, sum2_f, sum1_g, sum2_g] = all_values
    f_dim = len(sum1_g)
    f_hat = sum1_f/n
    sigma2f_hat = sum2_f/(n-1)-pow(f_hat,2)*n/(n-1)
    g_hat = []
    g_sigma2 = []
    for i in range(f_dim):
        gi_hat = sum1_g[i]/n
        g_hat += [gi_hat]
        g_sigma2 += [sum2_g[i]/(n-1)-pow(gi_hat,2)*n/(n-1)]
    sigma2g_hat = max(g_sigma2)#sum(g_sigma)
    return [f_hat, sigma2f_hat, g_hat, sigma2g_hat]

def BConstruction(p, g_hat, g_hat_old, B_old):
    y = asarray(g_hat) - asarray(g_hat_old)
    s = asarray(p)
    b = asarray(B_old)
    B_new = b[:]
    secant = y - dot(b,s)
    r1 = 1e-6
    r2 = 1e-3

    """ first SR1, then BFGS """
    if abs(dot(s,secant)) >= r1*linalg.norm(s)*linalg.norm(secant):
        B_new += outer(secant,secant)/dot(secant,s)
    elif linalg.norm(secant) > r2 and abs(dot(y,s)) > r2:
        B_new += -dot(dot(b,outer(s,s)),b)/dot(dot(s,b),s) + outer(y,y)/dot(y,s)
    return B_new

def SubProblem(x, g_hat, B, Delta):
    """ Nocedal's book page 72"""
    g_hat_norm = linalg.norm(g_hat)
    tau = 1
    val = dot(dot(transpose(g_hat),B),g_hat)
    if val > 0:
        tau = min(pow(g_hat_norm,3)/(Delta*val),1)
    p = (-tau*Delta/g_hat_norm)*asarray(g_hat)
    model_reduction = dot(g_hat,p)+0.5*dot(dot(p,B),p)
    return [p, model_reduction]

def ASTRO(jl, x_0, Delta_0, noise, iseeds, total_reps_maxes, show_factor, res_file, prob_name,
     test_set, gamma_1):
    global f_dim, p, m_o, All_Y, All_Values, \
```

```
    total_samples, total_reps, iseed, stop, re_used, \
    total_reps_max, show_error, points_rec, record_res_count, \
    x_inc, optimum, record_res, grad_res

    """INPUT PARAMETERS"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    f_dim = len(x_0)
    if noise == 0:
        eta_1 = .5
        eta_2 = .25
        eta_3 = .2
        gamma_1 = .5
        gamma_2 = 2
    else:
        eta_1 = .2
        eta_2 = .1
        eta_3 = .001
#        gamma_1 = .9
        gamma_2 = 1./gamma_1
    kappaf = 1; kappag = 1
    Delta_max = 10000
    Delta_min = 1e-3
    iteration_max = 1000
    g_norm_tolerance = 1e-2
    n_max = 1e6
    epsilon = 1e-3
    gamma = .5

    """COUNT VARIABLES """
    stop = 0; re_used = 0; total_samples = 0; total_reps = 0

    """OTHER SETTINGS """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    total_reps_max = total_reps_maxes[-1] #assuming it is sorted
    if os.path.exists(str(prob_name)+'.4plot.txt'): os.remove(str(prob_name)+'.4plot.txt');
    if os.path.exists(str(prob_name)+'.4plot.grad.txt'): os.remove(str(prob_name)+'.4plot.grad.txt');
    record_file = open(str(prob_name)+'.4plot.txt', 'a+')
    grad_file = open(str(prob_name)+'.4plot.grad.txt', 'a+')

    record_res = [0]*len(total_reps_maxes)
    grad_res = [0]*len(total_reps_maxes)

    total_record_res =[]
    total_grad_res = []

    if test_set:
```

```
    print 'seed\t iter\t Delta\t\t f_hat\t\t f\t\t ||g_hat||\t ||g||\t\t max_error\t total_samples
        \t total_work'

    res_file.write('\nseed\t iter\t Delta\t\t f_hat\t\t f\t\t |g_hat|\t |g|\t\t max_error\t
        total_samples\t total_work')

final_gnorm = []; final_f = []
g_norm_sum = 0
for seed_init in iseeds:
    """RESET"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    stop = 0; re_used = 0; total_samples = 0; total_reps = 0; record_res_count = 0
    iseed = seed_init
    x_init = x_0
    Delta = Delta_0
    iteration = 1
    All_Y = []
    All_Values = []
    points_rec = 0
    x_inc = x_init

    [n, f_inc, g_hat] = AdaptiveSampling(jl, x_inc, 1, Delta, kappaf, kappag, gamma, noise, n_max,
         total_reps_maxes, 0)

    obj_determ = CUTEObjEval(jl, x_inc, 0)
    grad_determ = CUTEGradEval(jl, x_inc, 0)
    g_hat_norm = linalg.norm(g_hat)
    B = identity(f_dim)
    if stop: break;

    if test_set:
        print '%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t%8d' \
        % (seed_init, iteration, round(Delta,4), round(f_inc,4), round(obj_determ,4), round(
            g_hat_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        abs(f_inc - obj_determ), total_samples, total_reps)

        res_file.write('\n%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t%8d'
            \
           % (seed_init, iteration, round(Delta,4), round(f_inc,4), round(obj_determ,4), round(
               g_hat_norm,4), \
           round(linalg.norm(asarray(grad_determ)),4),\
           abs(f_inc - obj_determ), total_samples, total_reps))
    while iteration <= iteration_max and Delta < Delta_max and Delta > Delta_min and total_reps <
        total_reps_max and g_hat_norm > g_norm_tolerance:
```

```
if noise == 0:
    inflator = 1; """ deterministic case """
else:
    inflator = pow(iteration, 3+epsilon)

[n, f_hat, g_hat] = AdaptiveSampling(jl, x_inc, inflator, Delta, kappaf, kappag, gamma,
     noise, n_max, total_reps_maxes, 0)
[p, model_reduction] = SubProblem(x_inc, g_hat, B, Delta)
x_candidate = (asarray(x_inc)+asarray(p)).tolist()
f_hat_old = f_hat
g_hat_old = g_hat[:]
if stop:
    f_inc = f_hat_old
    g_inc_norm = linalg.norm(g_hat_old)
    break

[n, f_hat, g_hat] = AdaptiveSampling(jl, x_candidate, inflator, Delta, kappaf, kappag,
     gamma, noise, n_max, total_reps_maxes, 1)

if stop:
    f_inc = f_hat_old
    g_inc_norm = linalg.norm(g_hat_old)
    break

rho_hat = (f_hat-f_hat_old)/model_reduction

""" UPDATE Delta """
if rho_hat < eta_2:
    Delta = Delta*gamma_1
elif rho_hat > eta_1 and linalg.norm(p) > .8*Delta:
    Delta = min(Delta*gamma_2,Delta_max)

""" UPDATE Iterate """
if rho_hat > eta_3:
    x_inc = x_candidate
    f_inc = f_hat
    g_inc_norm = linalg.norm(g_hat)
else:
    f_inc = f_hat_old
    g_inc_norm = linalg.norm(g_hat_old)

""" UPDATE Hessian Approximation """
if iteration == 1:
    y = (asarray(g_hat)-asarray(g_hat_old))
```

```
        B = linalg.inv(dot(y,p)/dot(y,y)*identity(f_dim))
    else:
        B = BConstruction(p, g_hat, g_hat_old, B)

    obj_determ = CUTEObjEval(jl, x_inc, 0)
    grad_determ = CUTEGradEval(jl, x_inc, 0)

    if test_set:
        if iteration % show_factor == 0:
            print '%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t%8d' \
            % (seed_init, iteration, round(Delta,4), round(f_inc,4), round(obj_determ,4), round(
                g_inc_norm,4), \
            round(linalg.norm(asarray(grad_determ)),4),\
            abs(f_inc - obj_determ), total_samples, total_reps)
        res_file.write('\n%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t
            %8d' \
        % (seed_init, iteration, round(Delta,4), round(f_inc,4), round(obj_determ,4), round(
            g_inc_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        abs(f_inc - obj_determ), total_samples, total_reps))
    iteration += 1

obj_determ = CUTEObjEval(jl, x_inc, 0)
grad_determ = CUTEGradEval(jl, x_inc, 0)
if iteration < iteration_max:
    if test_set:
        print '%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t%8d' \
        % (seed_init, iteration-1, round(Delta,4), round(f_inc,4), round(obj_determ,4), round(
            g_inc_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        abs(f_inc - obj_determ), total_samples, total_reps)
        res_file.write('\n%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t
            %8d' \
        % (seed_init, iteration, round(Delta,4), round(f_inc,4), round(obj_determ,4), round(
            g_inc_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        abs(f_inc - obj_determ), total_samples, total_reps))
res_file.write('\n\n x_inc '+str(x_inc)+'\n f_hat '+str(round(f_inc,4))+\
'\n f '+str(round(obj_determ,4))+'\n g_hat '+str(round(g_inc_norm,4))+\
'\n g '+str(round(linalg.norm(asarray(grad_determ)),4))+'\n Delta '+str(round(Delta,4)))

if iteration > iteration_max:
    if test_set:
        print 'iterations maxed.'
```

```
            res_file.write('\niterations maxed.')
        elif Delta >= Delta_max:
            if test_set:
                print 'Delta maxed.'
                res_file.write('\nDelta maxed.')
        elif Delta <= Delta_min:
            if test_set:
                print 'Delta mined.'
                res_file.write('\nDelta mined.')
        elif total_reps >= total_reps_max:
            if test_set:
                print 'replications maxed.'
                res_file.write('\nreplications maxed.')
        elif g_inc_norm <= g_norm_tolerance:
            if test_set:
                print 'g_norm mined.'
                res_file.write('\ng_norm mined.')

        """ at the end of each sample path save the true G Vector """
        final_gnorm += [round(linalg.norm(asarray(grad_determ)),4)]
        """ at the end of each sample path save the true f Value """
        final_f += [abs(round(linalg.norm(asarray(obj_determ)),4)-optimum)]
        g_norm_sum += g_inc_norm


        total_record_res += [record_res[:]]
        total_grad_res += [grad_res[:]]
    """ not the pre-processing but the actual processing """
    if test_set:
        record_file.write(str(total_record_res)+'\n')
        grad_file.write(str(total_grad_res)+'\n')
        B = asarray(total_record_res)
        G = asarray(total_grad_res)

        """ initiate """
        col1 = []
        col2 = []
        B_new = zeros((len(iseeds),len(total_reps_maxes)))
        G_new = zeros((len(iseeds),len(total_reps_maxes)))
        for i in range(len(total_reps_maxes)):
            col1 = B[:,i].tolist()
            col2 = G[:,i].tolist()
            col1.sort()
            col2.sort()
```

```
            B_new[:,i] = asarray(col1)
            G_new[:,i] = asarray(col2)
        if len(iseeds) >= 10:
            rows = [int(floor(len(iseeds)*.25)),int(floor(len(iseeds)*.5)),int(floor(len(iseeds)*.75)),
                int(floor(len(iseeds)*.9))]

            plt.figure()
            plt.plot(total_reps_maxes, B_new[rows[0],:].tolist(), 'r', total_reps_maxes, B_new[rows
                [1],:].tolist(), \
            'g',total_reps_maxes, B_new[rows[2],:].tolist(), 'b',total_reps_maxes, B_new[rows[3],:].
                tolist(), 'k')
            plt.xlabel('Simulation Budget')
            plt.ylabel('Optimality Gap')
            plt.savefig(str(prob_name)+'_og_plot_.jpg')

            plt.figure()
            plt.plot(total_reps_maxes, G_new[rows[0],:].tolist(), 'r--', total_reps_maxes, G_new[rows
                [1],:].tolist(), 'g--',\
            total_reps_maxes, G_new[rows[2],:].tolist(), 'b--',total_reps_maxes, G_new[rows[3],:].
                tolist(), 'k--')
            plt.xlabel('Simulation Budget')
            plt.ylabel('Gradient Norm')
            plt.savefig(str(prob_name)+'_grad_plot_.jpg')
    record_file.close()
    grad_file.close()
    return g_norm_sum/len(iseeds)

def main(prob_name, noises, iseeds, total_reps_maxes, show_factor):
    global f_dim, p, m_o, All_Y, All_Values, freshpoised_per, show_error
    All_Y = []; All_Values = [];
    iseeds_list = ast.literal_eval(iseeds)
    total_reps_maxes_list = ast.literal_eval(total_reps_maxes)

    if os.path.exists(str(prob_name)+'.log.txt'):
        os.remove(str(prob_name)+'.log.txt')
    res_file = open(str(prob_name)+'.log.txt', 'a+')

    jl = julia.Julia()
    jl.call('using CUTEst')
    jl.call('nlp = CUTEstModel("'+str(prob_name)+'")')
    f_dim = jl.eval('nlp.meta.nvar')

    res_file.write('\nProblem: '+str(prob_name)+'\ntotal_reps_max: '+str(total_reps_maxes_list[0]))
    res_file.write('\npre-processing\n-----------------')
```

```
print 'pre-processing\n-----------------'
""" pre-processing
Class that will first do some training to choose the best starting point and
starting trust region radius by running several times at first
(2 starting points each with 2 starting trust-region radius:
  x_0 = [16]*f_dim
  x_0 = [-16]*f_dim

  Delta_0 = 10
  Delta_0 = 5

run each for 100 iterations and pick the one combo with smallest f_hat) """
for noise in noises:
    All_Y = []; All_Values = [];
    x_0_vals = [10.0,-10.0]
    Delta_0s = [5,15]
    gamma_1s = [.9,.99]
    g_norms = []
    g_min = 1e20
    for x_0_val in x_0_vals:
        for Delta_0 in Delta_0s:
            for gamma_1 in gamma_1s:
                g_new = ASTRO(jl, [x_0_val]*f_dim, Delta_0, noise, [10], [100], show_factor,
                     res_file, prob_name, 0, gamma_1)

                g_norms += [g_new]
                if g_new < g_min:
                    g_min = g_new
                    x_0_val_best = x_0_val
                    Delta_0_best = Delta_0
                    gamma_1_best = gamma_1

    print '\ng_norms '+str(g_norms)
    print 'g_min ' + str(min(g_norms))

    res_file.write('\n\ng_norms '+str(g_norms))
    res_file.write('\ng_min ' + str(min(g_norms)))
    res_file.write('\nx_0_val_best '+str(x_0_val_best))
    res_file.write('\nDelta_0_best '+str(Delta_0_best))

    """""""""""""""""""""""""""
    print '\nfinal results for '+prob_name+'\n-----------------'
    print 'x_0_val_best '+str(x_0_val_best)+', Delta_0_best '+str(Delta_0_best)+', gamma_1_best '+
         str(gamma_1_best)
```

```
        res_file.write('\nfinal results \n-----------------')
        ASTRO(jl, [x_0_val_best]*f_dim, Delta_0_best, noise, iseeds_list, total_reps_maxes_list,
             show_factor, res_file, prob_name, 1, gamma_1_best)

    jl.call('cutest_finalize(nlp)')
    os.remove('AUTOMAT.d')
    os.remove('OUTSDIF.d')
    os.remove('lib'+str(prob_name)+'.dylib')
    res_file.close()

global f_dim, p, m_o, All_Y, All_Values, kappa_ias, kappa_oas, \
total_samples, total_reps, iseed, stop, re_used, total_reps_max, \
previous_success, x_candidate, l, model_order, p, iseed, show_error,\
record_res_count, x_inc, optimum, record_res, grad_res

All_Y = []; All_Values = [];
l = []
x_inc = []
record_res = []
grad_res = []
optimum = 0
model_order = 0; p = 0
re_used = 0
total_samples = 0; total_reps = 0
iseed = 0
stop = 0
total_reps_max = 0
record_res_count = 0

if __name__ == "__main__":
    noises = [1]
    show_factor = 10
    prob_name = sys.argv[1]#"DENSCHNC"#
    total_reps_maxes = sys.argv[2]#"[20000]"#
    iseeds = sys.argv[3]#"[30]"#
    main(prob_name, noises, iseeds, total_reps_maxes, show_factor)
```

The ASTRO-DF main code is copied in the following:

```
from math import sqrt, ceil, floor, isnan, isinf
import matplotlib.pyplot as plt
from numpy import asarray, transpose, linalg, dot, subtract, zeros, nan_to_num, matrix, add, append,
     eye, mean, log
from scipy.stats import norm
from scipy.optimize import minimize
import ast
import julia
import os
import sys

"""
Function definitions
    ValuesUpdate calculates updates f_hat, and sigma_hat with the new adaptive n
        and returns (n, f_hat, sigma_hat)
    PoisedSet finds an equi-distance poise d set from x
    AdaptiveSampling adds replications based on the sampling rule
    ModelConstruction the P matrix and FY for all the points in the interpolation set
        are used to calculate alpha in P.alpha = FY. The alpha consists of
        c, g, and A in
            m(x) = c+transpose(x)*g+0.5*transpose(x)*A*x.
        Note g and A will be at the origin. Then g(x) = g+transpose(x)*A and then
            m(x+s) = m(x)+transpose(s)*g(x)+0.5*transpose(s)*A*s.
        Returns g(x), and A.
    SubProblem solves the linear or quadratic minimization problem
"""

"""
All_Y is a (inf*d) matrix, coordinates of each point
All_Values is a (inf*3) matrix, [n, f_hat, sigma_hat] with the same index points in the All_Y
    sigm_hat is used to add more reps to the visited points in the future
"""

def u16807d():
    global iseed
    u = 0.
    while round(u,6) <= 0 or u >= 1:
        iseed = (int(iseed)*16807) % 2147483647
        u = iseed / 2147483648.
    return u

def CUTEObjEval(jl, x, noise):
```

```
    global iseed, show_error
    f_dim = len(x)
    string = '['
    for i in range(f_dim):
        string += str(x[i])
        if i < f_dim-1:
            string += ';'
        else:
            string += ']'

    noise_added = 0
    if noise > 0:
        u = u16807d()
        noise_added = norm.ppf(u, loc=0, scale=sqrt(noise))
        if noise_added > 10:
            if show_error: print 'OK the problem noise is larger than 10'
        obj = noise_added
    else:
        obj = 0
    jl.call('f = ufn(nlp,'+string+')')
    obj += jl.eval('f')
    if isnan(obj):
    return obj

def CUTEGradEval(jl, x):
    f_dim = len(x)
    string = '['
    for i in range(f_dim):
        string += str(x[i])
        if i < f_dim-1:
            string += ';'
        else:
            string += ']'

    jl.call('g = ugr(nlp,'+string+')')
    grad = jl.eval('g')
    return grad

def AdaptiveSampling(jl, x, inflator, Delta, kappa, noise, n_max, total_reps_maxes):
    global All_Y, All_Values, stop, total_reps, total_samples, iseed, total_reps_max, \
    points_rec, record_res_count, x_inc, optimum, record_res, grad_res
    if noise > 0: n = int(max(2, ceil(inflator)));
    else: n = 1;
    f_hat = 0
```

```
    current_reps_max = total_reps_maxes[record_res_count]
    if noise > 0:
        threshold = n_max - 1
        while n <= threshold and n <= n_max and not stop:
            [n, f_hat, sigma_hat] = ValuesUpdate(jl, x, n, noise, total_reps_maxes)
            if total_reps >= total_reps_max: stop = 1;
            n += 1
            if n > n_max: stop = 1;
        n -= 1
    else:
        """ if problem is deterministic do not use adaptive sampling, just evaluate once """
        if x not in All_Y:
            All_Y += [x]
            total_samples += 1
            if total_samples % 30 == 0:
                points_rec = 1
            [n, f_hat, sigma_hat] = [1, CUTEObjEval(jl, x, noise), 0]
            total_reps += 1
            if total_reps >= current_reps_max:
                if record_res_count < len(total_reps_maxes):
                    obj_determ = CUTEObjEval(jl, x_inc, 0)
                    grad_determ = CUTEGradEval(jl, x_inc)
                    record_res[record_res_count] = round(abs(obj_determ - optimum),4)
                    grad_res[record_res_count] = round(linalg.norm(asarray(grad_determ)),4)
                    record_res_count += 1
                if record_res_count >= len(total_reps_maxes):
                    stop = 1
                else:
                    current_reps_max = total_reps_maxes[record_res_count]
            All_Values += [[n, f_hat, sigma_hat]]
        else:
            f_hat = All_Values[list.index(All_Y, x)][1]
    return [n, f_hat]

""" All_Y and All_Values are only updated here """
def ValuesUpdate(jl, x, n_new, noise, total_reps_maxes):

    global All_Y, All_Values, total_samples, total_reps, iseed, show_error, \
    points_rec, record_res_count, x_inc, optimum, stop, record_res, grad_res
    [n_old, f_hat_old, sigma_hat_old] = [0, 0.0, 0.0]
    current_reps_max = total_reps_maxes[record_res_count]

    revisit_index = -1
    x_exists = 0
```

```
if x in All_Y:
    revisit_index = list.index(All_Y,x)
    [n_old, f_hat_old, sigma_hat_old] = All_Values[revisit_index]
    x_exists = 1
else:
    total_samples += 1
    if total_samples % 30 == 0:
        points_rec = 1
    All_Y += [x]

[sum1, sum2] = [0.0, 0.0]
sum1 = f_hat_old*(float(n_old)/float(n_new))
sum2 = (pow(sigma_hat_old,2)+pow(f_hat_old,2))*(float(n_old)/float(n_new))
[temp1, temp2] = [0.0, 0.0]

if n_new > n_old:
    for i in range(int(n_old), int(n_new)):
        obs_new = CUTEObjEval(jl, x, noise)
        total_reps += 1
        if total_reps >= current_reps_max:
            if record_res_count < len(total_reps_maxes):
                obj_determ = CUTEObjEval(jl, x_inc, 0)
                grad_determ = CUTEGradEval(jl, x_inc)
                record_res[record_res_count] = round(abs(obj_determ - optimum),4)
                grad_res[record_res_count] = round(linalg.norm(asarray(grad_determ)),4)
                record_res_count += 1
            if record_res_count >= len(total_reps_maxes):
                stop = 1
            else:
                current_reps_max = total_reps_maxes[record_res_count]

        temp1 +=obs_new/float(n_new)
        temp2 += pow(obs_new,2)/float(n_new)

    f_hat = sum1 + temp1
    sigma_hat = sqrt(max((sum2 + temp2) - pow(f_hat,2),0))
else:
    [n_new, f_hat, sigma_hat] = [n_old, f_hat_old, sigma_hat_old]

if x_exists:
    All_Values[revisit_index] = [n_new, f_hat, sigma_hat]
else:
    All_Values += [[n_new, f_hat, sigma_hat]]
```

```
    return [n_new, f_hat, sigma_hat]

def PoisedSet(Delta, func, x_inc, freshpoised_per):
    global All_Y, All_Values, re_used, show_factor, l, p, show_error
    min_away = Delta*0.2 # min factor of Delta distannce from each other
    Y_pool = [x_inc]
    """ Put everything in the current TR that is observed before in Y_pool"""
    for i in range(1,len(All_Y)+1):
        if All_Y[-i] not in Y_pool:
            dist = linalg.norm(subtract(asarray(All_Y[-i]),asarray(x_inc)))
            if dist <= Delta:# and dist >= Delta/2.0:
                Y_pool += [All_Y[-i]]
    """ points must be at least more than 5%Delta distance apart from each other
    Note: can start from the beginning or the end of Y_pool
    CHECK THIS !!"""
    for i in range(1,len(Y_pool)):
        for j in range(i):
            if Y_pool[j] != []:
                dist = linalg.norm(subtract(asarray(Y_pool[i]),asarray(Y_pool[j])))
                if dist <= min_away:
                    Y_pool[i] = []
                    break
    Y_pool = list(filter(None, Y_pool))
    Y_pool.remove(x_inc)

    """ Y_init finds a full-poised set from scratch """
    Y_init = [x_inc]
    """ reset lagrange functions to the nominal basis l_i(x) = phi_i(x)"""
    L = [[0.]*p]; L[0][0] = 1;
    for i in range(1, p):
        L += [[0.]*p]
        L[i][i] = 1
    l = L[0]
    """  find a poised set around the center point and update the Lagrange functions """
    """ make sure that all the points are at least 20%Delta distance away from each other """
    for i in range(1, p):
        """ 1. point selection """
        l = L[i]
        x_ini = x_inc[:]
        x_ini[-1] += Delta/1.01; """ STARTING POINT FOR THE OPT """
        good_point = 0
        [y_next, res] = LagrangeMax(Delta, func, x_ini, x_inc)

        while not good_point:
```

```
        for j in range(len(Y_init)):

            y_next_dist = linalg.norm(subtract(y_next.tolist(), Y_init[j]))
            if y_next_dist < min_away:
                """ push the point slightly to the opposite direction of the closest point """
                move_to = (min_away)*subtract(asarray(Y_init[j]), y_next)/y_next_dist
                y_next = subtract(y_next, move_to)
                good_point = 0
                break
            else:
                good_point = 1

    Y_init += [y_next.tolist()]
    """ 2. normalization """
    l = L[i][:]
    factor = func(Y_init[i], 1)
    L[i] = [round(li/factor, 3) for li in L[i]]
    """ 3. orthagonalization """
    for j in range(p):
        if j != i:
            l = L[j]
            factor = func(Y_init[i], 1)
            L[j] = [round(lj-factor*li, 3) for lj, li in zip(L[j], L[i])]

""" Now choose the closest observed points within 5%Delta to the selected \
points (if any), and rank them """

Y_closest = [[]]
Y_closest_dists = []; """ just the closest points """
Y_all_dists = [0]; """ For all the points, regardless if they have closest or no closest found (0)
    .
                    if you don't find a point within 5%Delta, set to 0"""
for i in range(1, p):
    min_dist = Delta*.05
    y_closest = []
    for j in range(len(Y_pool)):
        dist = linalg.norm(subtract(asarray(Y_pool[j]), asarray(Y_init[i])))
        if dist <= min_dist:
            min_dist = dist
            y_closest = Y_pool[j]
    Y_closest += [y_closest]
    if not y_closest == []:
        Y_pool.remove(y_closest)
        Y_closest_dists += [min_dist]
```

```
        Y_all_dists += [min_dist]
    else:
        Y_all_dists += [0]

Y_final = [x_inc]
Y_init[0] = []
""" Now based on the freshpoised_per, choose from Y_init or points with \
the least distance to the original points
if freshpoised = 1, then use all the points from Y_init or the closest to them
if freshpoised = 0, just use points from Y_all that are farthest away from the points \
                already in the Y_final and at least 5%Delta distance away from them """
freshpoised_num = int(floor(freshpoised_per*(p-1)))

""" for the freshpoised_num of the total points first choose from Y_closest \
(the closest observed points to the ones in Y_init) and remove the associated points in Y_init, \
then choose the rest from the remaining points in Y_init that are farthest \
from all the points already in Y_final but also at least 5%Delta distance away from each of them
     """
for i in range(freshpoised_num):
    if not Y_closest_dists == []:
        freshpoised_index = list.index(Y_all_dists, min(Y_closest_dists))
        Y_final += [Y_closest[freshpoised_index][:]]
        Y_closest_dists.remove(min(Y_closest_dists))
        Y_init[freshpoised_index] = []
    else:
        Y_farthest_dists = [0]*len(Y_init)
        for j in range(1, len(Y_init)):
            total_dist = 0
            too_close = 0
            for y_final in Y_final:
                if not Y_init[j] == []:
                    dist = linalg.norm(subtract(asarray(y_final), asarray(Y_init[j])))
                    if dist < min_away:
                        too_close = 1
                        break
                else:
                     dist = 0
                total_dist += dist
            if not too_close:
                Y_farthest_dists[j] = total_dist
        """ this situation happens when the new poised set (Y_init) gives too close points """
        if max(Y_farthest_dists) == 0:
            if show_error: print "Y_farthest_dists is 0"
            for j in range(1,len(Y_init)):
```

```
                if not Y_init[j] == []:
                    Y_farthest_dists[j] = min_away
                    break

        farthest_index = list.index(Y_farthest_dists, max(Y_farthest_dists))
        Y_final += [Y_init[farthest_index]]
        Y_init[farthest_index] = []

Y_init = list(filter(None, Y_init))

""" for the remaining p - freshpoised_num of the points, reuse old points that
are farthest from all the other points already selected """
for i in range(freshpoised_num+1, p):
    if len(Y_pool) > 0 and not max(Y_pool) == []:
        Y_farthest_dists = [0]*len(Y_pool)
        for j in range(1, len(Y_pool)):
            total_dist = 0
            too_close = 0
            for y_final in Y_final:
                if not Y_pool[j] == []:
                    dist = linalg.norm(subtract(asarray(y_final), asarray(Y_pool[j])))
                    if dist < min_away:
                        too_close = 1
                        break
                else:
                    dist = 0
                total_dist += dist
            if not too_close:
                Y_farthest_dists[j] = total_dist
        if max(Y_farthest_dists) == 0:
            for j in range(1,len(Y_pool)):
                if not Y_pool[j] == []:
                    Y_farthest_dists[j] = min_away
                    break
        farthest_index = list.index(Y_farthest_dists, max(Y_farthest_dists))
        Y_final += [Y_pool[farthest_index]]
        Y_pool[farthest_index] = []
    else:
        Y_farthest_dists = [0]*len(Y_init)
        for j in range(1, len(Y_init)):
            total_dist = 0
            too_close = 0
            for y_final in Y_final:
                if not Y_init[j] == []:
```

```
                        dist = linalg.norm(subtract(asarray(y_final), asarray(Y_init[j])))
                        if dist < min_away:
                            too_close = 1
                            break
                    else:
                         dist = 0
                    total_dist += dist
                if not too_close:
                    Y_farthest_dists[j] = total_dist
            """ this situation happens when the new poised set (Y_init) gives too close points """
            if max(Y_farthest_dists) == 0:
                if show_error: print "Y_farthest_dists is 0"
                for j in range(1,len(Y_init)):
                    if not Y_init[j] == []:
                        Y_farthest_dists[j] = min_away
                        break
            farthest_index = list.index(Y_farthest_dists, max(Y_farthest_dists))
            Y_final += [Y_init[farthest_index]]
            Y_init[farthest_index] = []

    Y_pool = list(filter(None, Y_pool))
    return Y_final

def ModelConstruction(Y, FY, model_order, p):
    f_dim = len(Y[0])
    P = [[1.0]+Y[i] for i in range(p)]
    if model_order == 2:
        for i in range(p):
            for j in range(f_dim):
                P[i] += [pow(Y[i][j],2)/2]
                for jj in range(j+1, f_dim):
                    P[i] += [Y[i][j]*Y[i][jj]]

    alpha = linalg.solve(P,FY)
    alpha = alpha.tolist()
    c = alpha[0]
    """ g at origin """
    g = alpha[1:f_dim+1]
    """ reading A from alpha """
    A = zeros(shape = (f_dim, f_dim), dtype = float)
    if model_order == 2:
        q_index = f_dim
        for j in range(f_dim):
            for jj in range(j, f_dim):
```

```
                q_index += 1
                A[j,jj] = alpha[q_index]
                A[jj,j] = alpha[q_index]
    """ g at x """
    g = dot(A,asarray(Y[0]))+asarray(g)
    return c, g, A

def SubProblem(x, c, g, A, Delta):
    global All_Y, All_Values
    """ METHOD 1 """
    """ M(x) = c + g'x + .5x'Ax
        dM(x) = g + Ax
        d2M(x) = A
        M(x+s) = M(x) + dM's + .5s'd2Ms
        M(x+s) - M(x) = dM's + .5s'd2Ms
        NOTE: what we call g in the code is in fact dM(x)
              To calculate M(x) we transform dM(x) to g by subtracting Ax from it"""

    g_orig = subtract(g, dot(A,asarray(x)))
    current_value = ModelEval(x, c, g_orig, A)
    x_init = x[:]
    x_init[-1] += Delta/1.01; """ starting point is chosen """
    cons = ({'type': 'ineq',\
    'fun' : lambda z: Delta/1.01 - linalg.norm(subtract(asarray(x),asarray(z))),\
    'jac' : None})
    """ constraint in the form of g_i >= 0"""
    res = minimize(ModelEval, x_init, args=(c, g_orig, A), jac=None,\
    constraints = cons, method='COBYLA', options={'disp': False})
    s = subtract(res.x, x)
    model_reduction = ModelEval(res.x, c, g_orig, A) - current_value
    """ METHOD 2 """
    if model_reduction > 0:
        g_norm = linalg.norm(g)
        if model_order == 2:
            val = pow(g_norm,3)/(dot(dot(transpose(g),A),g))
            """ convex? """
            if val > 0:
                s = -val*g/g_norm
                if linalg.norm(s) > Delta:
                    x_star = [xi+ si for xi, si in zip(x, s)]
                    x_gap = [xstari-xoldi for xstari, xoldi in zip(x_star, x)]
                    s = Delta*asarray(x_gap)/linalg.norm(asarray(x_gap))
            else:
                s = -Delta*g/g_norm
```

```
        else:
            s = -Delta*g/g_norm
        model_reduction = dot(transpose(g),s)+0.5*dot(dot(transpose(s),A),s)
    return [s, model_reduction]

""" the derivative can be derived: Grad(i,0)=i+1;
    Grad(i,i+1)=(i+1)+id+ (for iter in range (1,i-1): -= iteration);
    for iteration=i+1..d: Grad(i,iteration)=Grad(iteration,i+1)=Grad(i,i+1)+(iteration-i)
"""

def ModelEval(x, c, g, A):
     return c + dot(transpose(g),x) + 0.5*dot(dot(transpose(x),A),x)

def LagrangeEval(x, sign):
    global l
    k = 0
    f_dim = len(x)
    f = l[k]
    for i in range(f_dim):
        k += 1
        f += l[k]*x[i]
    for i in range(f_dim):
        k += 1
        f += float(l[k]*pow(x[i],2))/2
        for j in range(i+1,f_dim):
            k += 1
            f += l[k]*x[i]*x[j]
    return sign*(f)

def LagrangeMax(r, func, x_init, x_inc):
    global l, show_error
    cons = ({'type': 'ineq',\
    'fun' : lambda x: r/1.01 - linalg.norm(subtract(asarray(x_inc),asarray(x))),\
    'jac' : None})
    """ constraint in the form of g_i >= 0"""
    res = minimize(func, x_init, args=(-1.0), jac=None,\
    constraints = cons, method='COBYLA', options={'disp': False,'catol':1e-5})
    """ other solvers: for bounds(L-BFGS-B, TNC), COBYLA SLSQP"""
    new_x = res.x
    if not res.success and show_error:
        print '!!!! Optimization didnt succeed !!!! '
    return [new_x, res]
```

```
def ASTRO(jl, x_0, Delta_0, noise, iseeds, total_reps_maxes, show_factor, res_file, prob_name,
     test_set):
    global f_dim, p, m_o, All_Y, All_Values, kappa_ias, kappa_oas, \
    total_samples, total_reps, iseed, stop, re_used, \
    l, model_order, p, total_reps_max, show_error, points_rec, record_res_count, x_inc, optimum,
         record_res, grad_res

    """INPUT PARAMETERS"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    f_dim = len(x_0); model_order = 2;
    eta_1 = 0.1; epsilon = 0.005; w = 0.99; mu = 1.1; beta = 1/mu;
    gamma_1 = pow(1.1,2./float(f_dim)); gamma_2 = 1/gamma_1; Delta_max = 10000; n_max = 1e6;
    kappa_ias = 1000; kappa_oas = 1000; Delta_min = 1e-4
    g_norm_tolerance = 1e-2; g_norm_criticality_tolerance = 10; iteration_max = 10000

    """ NEW parameter used in PoisedSet method: how many observed points to re-use
    when = 1 we use every point that is closest and at most 5%Delta distance to the ideal poised set
    when = 0 we use the ideal poised set itself """

    """COUNT VARIABLES """
    stop = 0; re_used = 0; total_samples = 0; total_reps = 0
    if model_order == 1:
        p = f_dim + 1
    else:
        p = (f_dim + 1)*(f_dim +2)/2
    l = [0]*p

    """OTHER SETTINGS """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    total_reps_max = total_reps_maxes[-1] #assuming it is sorted
    if os.path.exists(str(prob_name)+'.4plot.txt'): os.remove(str(prob_name)+'.4plot.txt');
    if os.path.exists(str(prob_name)+'.4plot.grad.txt'): os.remove(str(prob_name)+'.4plot.grad.txt');
    record_file = open(str(prob_name)+'.4plot.txt', 'a+')
    grad_file = open(str(prob_name)+'.4plot.grad.txt', 'a+')

    record_res = [0]*len(total_reps_maxes)
    grad_res = [0]*len(total_reps_maxes)

    total_record_res =[]
    total_grad_res = []

    if test_set:
        print 'seed\t iter\t Delta\t\t f_hat\t\t f\t\t |g_hat|\t |g|\t\t max_error\t min_dist/Delta\t
             total_samples\t total_work'
        res_file.write('\nseed\t iter\t Delta\t\t f_hat\t\t f\t\t |g_hat|\t |g|\t\t max_error\t
             min_dist/Delta\t total_samples\t total_work')
```

```
final_gnorm = []; final_f = []
g_norm_sum = 0
for seed_init in iseeds:

    """RESET"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    stop = 0; re_used = 0; total_samples = 0; total_reps = 0; record_res_count = 0
    iseed = seed_init
    x_init = x_0
    Delta = Delta_0
    g_norm = 1e6; """ dummy g_norm """
    iteration = 1
    All_Y = []
    All_Values = []
    points_rec = 0
    Y = [[0.0]*f_dim]*p
    FY = [0.0]*p
    x_inc = x_init
    Delta_tilda = Delta

    [n, FY[0]] = AdaptiveSampling(jl, x_inc, 1, Delta, kappa_oas, noise, n_max, total_reps_maxes)
    f_inc = FY[0]
    obj_determ = CUTEObjEval(jl, x_inc, 0)
    grad_determ = CUTEGradEval(jl, x_inc)
    if stop: break;
    """ print the first iteration """

    if test_set:
        print '%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t%8d' \
        % (seed_init, iteration, round(Delta_tilda,4), round(f_inc,4), round(obj_determ,4), round(
             g_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        abs(f_inc - obj_determ), 0, total_samples, total_reps)

        res_file.write('\n%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \
             t\t%8d' \
            % (seed_init, iteration, round(Delta_tilda,4), round(f_inc,4), round(obj_determ,4),
                 round(g_norm,4), \
            round(linalg.norm(asarray(grad_determ)),4),\
            abs(f_inc - obj_determ), 0, total_samples, total_reps))
    while iteration <= iteration_max and Delta < Delta_max and Delta > Delta_min and total_reps <
         total_reps_max and g_norm > g_norm_tolerance:
        if noise == 0:
            inflator = 1; """ deterministic case """
        else:
```

```
    inflator = max(1,ceil(pow(iteration, 1+epsilon)))
if g_norm <= g_norm_criticality_tolerance:
    freshpoised_per = 1
else:
    freshpoised_per = 0

""" INNER LOOP """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
innerloop = 1
if noise > 0: improve = 1;
else: improve = 1; """ this is no not always improve the model by resampling when
     deterministic.. don't have this yet in stochastic """
Delta = Delta_tilda; """ keeping last final Delta to compare with new one """

""" CRITICALITY STEP """"""""""""""""""""""""""""""""""""""""""""""""""""""
""" if the model gradient is large, just replace the best point with the current iterate
     and replace the current iterate with the farthest point in the current pool (can also
     replace with the farthest from the new iterate) """
""" re-build the model at the current points without extra observations """
while innerloop:
    if noise == 0:
        if g_norm <= 1:
            improve = 1
    if improve:
        Y = PoisedSet(Delta_tilda, LagrangeEval, x_inc, freshpoised_per);
        for i in range(p):

            [n, FY[i]] = AdaptiveSampling(jl, Y[i][:], inflator, Delta_tilda, kappa_ias,
                 noise, n_max, total_reps_maxes)
            f_inc = FY[0]
            obj_determ = CUTEObjEval(jl, x_inc, 0)
            grad_determ = CUTEGradEval(jl, x_inc)
            if stop: break;
        if stop: break;
        [c, g, A] = ModelConstruction(Y, FY, model_order, p)
        g_norm = linalg.norm(g)
        if mu*g_norm < Delta_tilda:
            Delta_tilda *= w
        else:
            Delta = min(Delta,max(beta*g_norm, Delta_tilda))
            innerloop = 0; """ terminate inner loop """
if stop: break;
""" INNER LOOP """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
errors = []
for i in range(p):
```

```
    errors += [FY[i] - CUTEObjEval(jl, Y[i], 0) ]
if max(errors) > 10 and show_error:
    print 'FY '+str(FY)
min_dist = Delta
for i in range(len(Y)):
    for j in range(i+1, len(Y)):
        dist = linalg.norm(subtract(asarray(Y[i]),asarray(Y[j])))
        if dist < min_dist: min_dist = dist;

""" OUTER LOOP """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
[s, model_reduction] = SubProblem(x_inc, c, g, A, Delta)
if model_reduction > 0 and show_error: print 'ALARMMMMMMM';
x_candidate = (asarray(x_inc)+asarray(s)).tolist()#[xi+ si for xi, si in zip(x_inc, s)]
[n, f_hat_new] = AdaptiveSampling(jl, x_candidate, inflator, Delta, kappa_oas, noise, n_max
     , total_reps_maxes)
obj_determ = CUTEObjEval(jl, x_inc, 0)
grad_determ = CUTEGradEval(jl, x_inc)
if stop: break
rho_hat = (f_hat_new-FY[0])/model_reduction
visited_min_index = list.index(FY, min(FY))

""" UPDATE: if success, check min(FY) """
if rho_hat >= eta_1 and model_reduction < 0:
     Delta_tilda = min(Delta_max, gamma_1*Delta)

     if min(FY) < f_hat_new:
         x_inc = Y[visited_min_index][:]
         f_inc = FY[visited_min_index]
     else:
         x_inc = x_candidate[:]
         f_inc = f_hat_new
else:
    Delta_tilda = gamma_2*Delta
    x_inc = Y[0][:]
    f_inc = FY[0]

""" include the candidate point in the set if not accepted
    replace with the farthest point
    or replace with the point that has the highest objective function,
    if the candidate point has a lower objective function """
if  not x_inc == x_candidate:
    dist_new = linalg.norm(subtract(asarray(x_inc),asarray(x_candidate)))
    dist_max = 0
    for point_in_Y in range(p):
```

```
            dist_in_Y = linalg.norm(subtract(asarray(x_inc),asarray(Y[point_in_Y])))
            if dist_in_Y > dist_max: dist_max = dist_in_Y; dist_argmax = point_in_Y;
        if dist_max > dist_new:
            Y[dist_argmax] = x_candidate[:]
            FY[dist_argmax] = f_hat_new
        elif f_hat_new < max(FY):
            Y[dist_argmax] = x_candidate[:]
            FY[dist_argmax] = f_hat_new

    """ OUTER LOOP """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    obj_determ = CUTEObjEval(jl, x_inc, 0)
    grad_determ = CUTEGradEval(jl, x_inc)
    if test_set:
            print '%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t
                %8d' \
            % (seed_init, iteration, round(Delta_tilda,4), round(f_inc,4), round(obj_determ,4),
                round(g_norm,4), \
            round(linalg.norm(asarray(grad_determ)),4),\
            round(abs(max(errors, key=abs)),2), min_dist/Delta, total_samples, total_reps)
        res_file.write('\n%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t
            %4d \t\t%8d' \
        % (seed_init, iteration, round(Delta_tilda,4), round(f_inc,4), round(obj_determ,4),
            round(g_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        round(abs(max(errors, key=abs)),2), min_dist/Delta, total_samples, total_reps))
    iteration += 1

obj_determ = CUTEObjEval(jl, x_inc, 0)
grad_determ = CUTEGradEval(jl, x_inc)
if iteration < iteration_max:
    if test_set:
        print '%4d \t%4d \t%7.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%9.4f \t%4d \t\t%8d'
            \
        % (seed_init, iteration-1, round(Delta_tilda,4), round(f_inc,4), round(obj_determ,4),
            round(g_norm,4), \
        round(linalg.norm(asarray(grad_determ)),4),\
        round(abs(max(errors, key=abs)),2), min_dist/Delta, total_samples, total_reps)
if test_set:
    res_file.write('\n\n x_inc '+str(x_inc)+'\n f_hat '+str(round(f_inc,4))+\
    '\n f '+str(round(obj_determ,4))+'\n g_hat '+str(round(g_norm,4))+\
    '\n g '+str(round(linalg.norm(asarray(grad_determ)),4))+'\n Delta '+str(round(Delta_tilda
        ,4)))

if iteration > iteration_max:
```

```
            if test_set:
                print 'iterations maxed.'
                res_file.write('\niterations maxed.')
        elif Delta >= Delta_max:
            if test_set:
                print 'Delta maxed.'
                res_file.write('\nDelta maxed.')
        elif Delta <= Delta_min:
            if test_set:
                print 'Delta mined.'
                res_file.write('\nDelta mined.')
        elif total_reps >= total_reps_max:
            if test_set:
                print 'replications maxed.'
                res_file.write('\nreplications maxed.')
        elif g_norm <= g_norm_tolerance:
            if test_set:
                print 'g_norm mined.'
                res_file.write('\ng_norm mined.')

        """ at the end of each sample path save the true G Vector """
        final_gnorm += [round(linalg.norm(asarray(grad_determ)),4)]
        """ at the end of each sample path save the true f Value """
        final_f += [abs(round(linalg.norm(asarray(obj_determ)),4)-optimum)]
        g_norm_sum += g_norm

        total_record_res += [record_res[:]]
        total_grad_res += [grad_res[:]]
    """ not the pre-processing but the actual processing """
    if test_set:
        record_file.write(str(total_record_res)+'\n')
        grad_file.write(str(total_grad_res)+'\n')
        B = asarray(total_record_res)
        G = asarray(total_grad_res)

        """ initiate """
        col1 = []
        col2 = []
        B_new = zeros((len(iseeds),len(total_reps_maxes)))
        G_new = zeros((len(iseeds),len(total_reps_maxes)))
        for i in range(len(total_reps_maxes)):
            col1 = B[:,i].tolist()
            col2 = G[:,i].tolist()
            col1.sort()
```

```
            col2.sort()
            B_new[:,i] = asarray(col1)
            G_new[:,i] = asarray(col2)
        if len(iseeds) >= 10:
            rows = [int(floor(len(iseeds)*.25)),int(floor(len(iseeds)*.5)),int(floor(len(iseeds)*.75)),
                int(floor(len(iseeds)*.9))]

            plt.figure()
            plt.plot(total_reps_maxes, B_new[rows[0],:].tolist(), 'r', total_reps_maxes, B_new[rows
                [1],:].tolist(), \
            'g',total_reps_maxes, B_new[rows[2],:].tolist(), 'b',total_reps_maxes, B_new[rows[3],:].
                tolist(), 'k')
            plt.xlabel('Simulation Budget')
            plt.ylabel('Optimality Gap')
            plt.savefig(str(prob_name)+'_og_plot_.jpg')


            plt.figure()
            plt.plot(total_reps_maxes, G_new[rows[0],:].tolist(), 'r--', total_reps_maxes, G_new[rows
                [1],:].tolist(), 'g--',\
            total_reps_maxes, G_new[rows[2],:].tolist(), 'b--',total_reps_maxes, G_new[rows[3],:].
                tolist(), 'k--')
            plt.xlabel('Simulation Budget')
            plt.ylabel('Gradient Norm')
            plt.savefig(str(prob_name)+'_grad_plot_.jpg')
    record_file.close()
    grad_file.close()
    return g_norm_sum/len(iseeds)

def main(prob_name, noise, iseeds, total_reps_maxes, show_factor):
    global f_dim, p, m_o, All_Y, All_Values, freshpoised_per, show_error
    All_Y = []; All_Values = [];
    iseeds_list = ast.literal_eval(iseeds)
    total_reps_maxes_list = ast.literal_eval(total_reps_maxes)

    if os.path.exists(str(prob_name)+'.log.txt'):
        os.remove(str(prob_name)+'.log.txt')
    res_file = open(str(prob_name)+'.log.txt', 'a+')

    jl = julia.Julia()
    jl.call('using CUTEst')
    jl.call('nlp = CUTEstModel("'+str(prob_name)+'")')
    f_dim = jl.eval('nlp.meta.nvar')
```

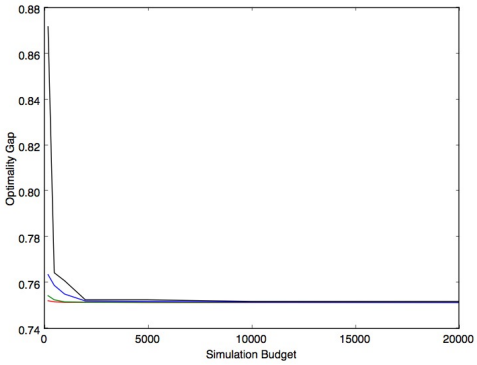
```
    res_file.write('\nProblem: '+str(prob_name)+'\ntotal_reps_max: '+str(total_reps_maxes_list[0]))
    res_file.write('\npre-processing\n-----------------')
    print 'pre-processing\n-----------------'
    """ pre-processing """
    for noise in [1]:
        All_Y = []; All_Values = [];
        print 'noise is '+str(noise)
        x_0_vals = [-16.0,16.0]
        Delta_0s = [5,15]
        g_norms = []
        g_min = 1e20
        freshpoised_per = 1
        for x_0_val in x_0_vals:
            for Delta_0 in Delta_0s:
                g_new = ASTRO(jl, [x_0_val]*f_dim, Delta_0, noise, [10], [500], show_factor, res_file,
                     prob_name, 0)
                g_norms += [g_new]
                if g_new < g_min:
                    g_min = g_new
                    x_0_val_best = x_0_val
                    Delta_0_best = Delta_0

        print '\ng_norms '+str(g_norms)
        print 'g_min ' + str(min(g_norms))
        print 'x_0_val_best '+str(x_0_val_best)
        print 'Delta_0_best '+str(Delta_0_best)
        """"""""""""""""""""""""""
        print '\nfinal results \n-----------------'
        res_file.write('\nfinal results \n-----------------')
        ASTRO(jl, [x_0_val_best]*f_dim, Delta_0_best, noise, iseeds_list, total_reps_maxes_list,
             show_factor, res_file, prob_name, 1)

    jl.call('cutest_finalize(nlp)')
    os.remove('AUTOMAT.d')
    os.remove('OUTSDIF.d')
    os.remove('lib'+str(prob_name)+'.dylib')
    res_file.close()

global f_dim, p, m_o, All_Y, All_Values, kappa_ias, kappa_oas, \
total_samples, total_reps, iseed, stop, re_used, total_reps_max, \
previous_success, x_candidate, l, model_order, p, iseed, show_error,\
record_res_count, x_inc, optimum, record_res, grad_res

show_error = 0
```

```
All_Y = []; All_Values = [];
l = []
x_inc = []
record_res = []; grad_res = []
optimum = 0
model_order = 0; p = 0
re_used = 0
total_samples = 0; total_reps = 0
iseed = 0
stop = 0
total_reps_max = 0; record_res_count = 0
```

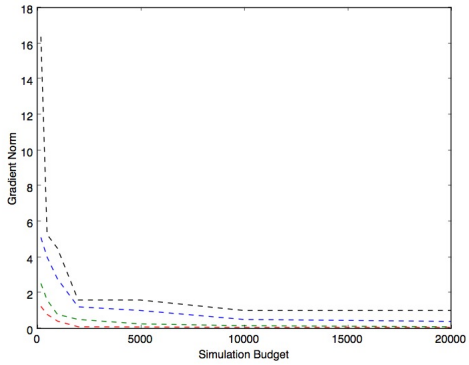
APPENDIX B

QUANTILE PLOTS OF ASTRO AND ASTRO-DF

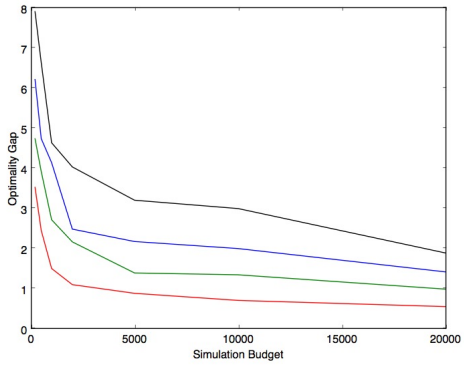
The quantile plots of ASTRO for the suite of low to moderate dimensional problems of the CUTEst framework are collected here. The red, green, blue and black plots are the 25%, 50%, 75% and 90% quantiles respectively. The solid lines (left plots) are the optimality gaps and the dashed lines (right plots) are the gradient norms.



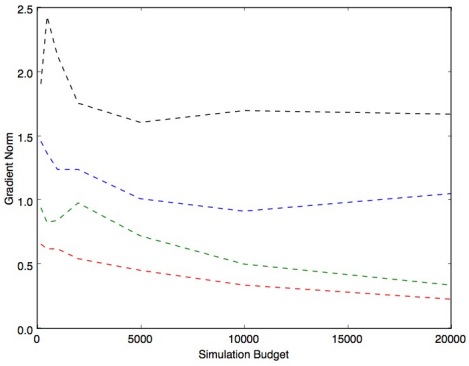
BEALE



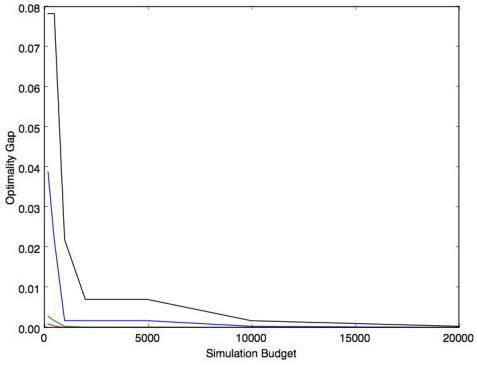
BEALE



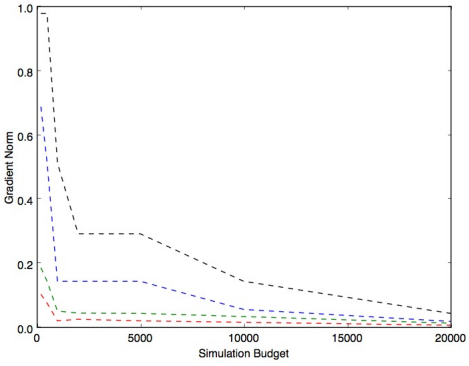
BIGGS6



BIGGS6

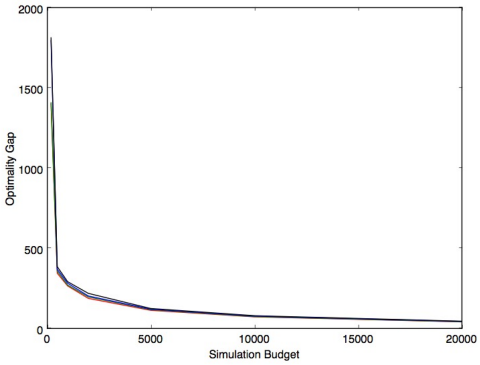


BOX3

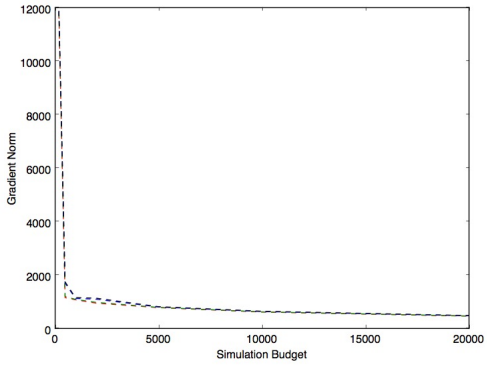


BOX3

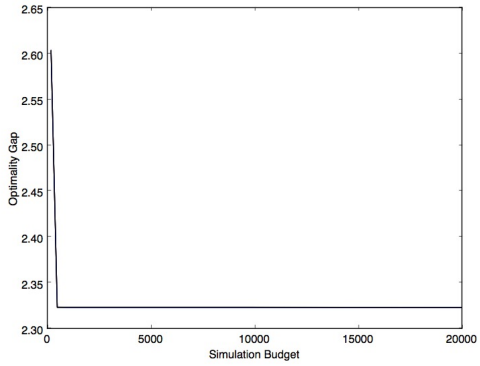
Figure B.1. Quantile plots of ASTRO for the functions BEALE, BIGGS6 and BOX3.



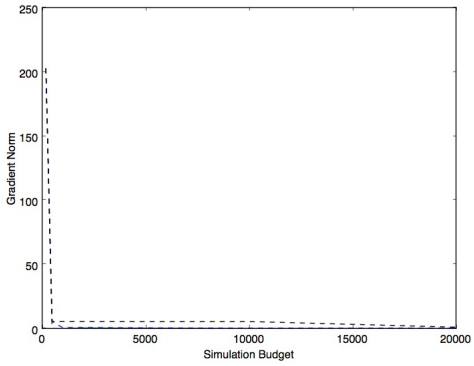
BROWNDEN



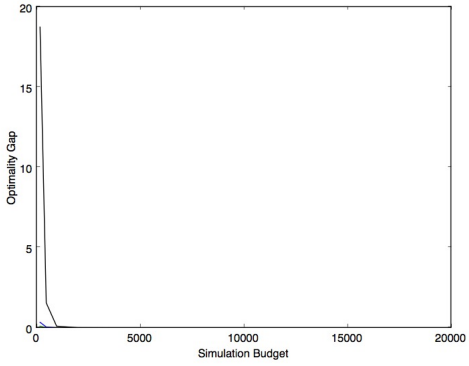
BROWNDEN



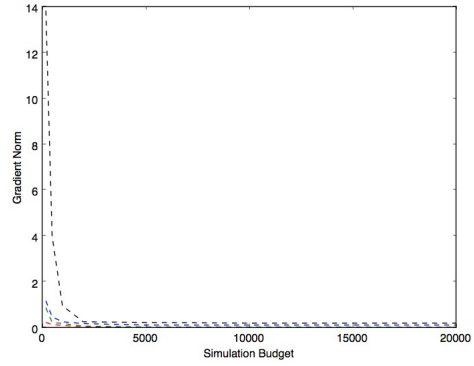
CUBE



CUBE

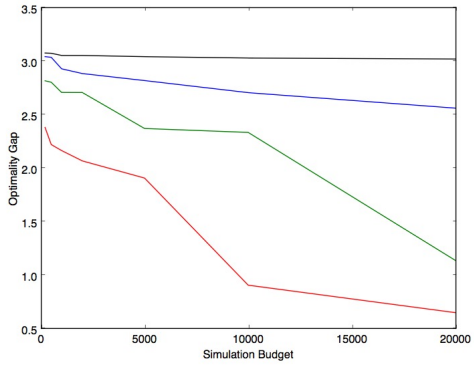


DENSCHNB

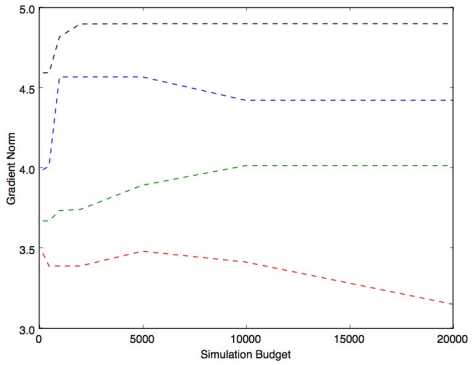


DENSCHNB

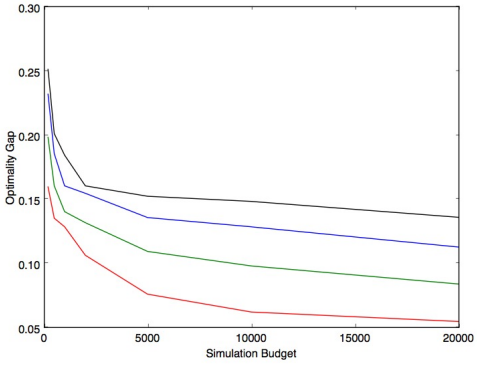
Figure B.2. Quantile plots of ASTRO for the functions BROWNDEN, CUBE and DENSCHNB.



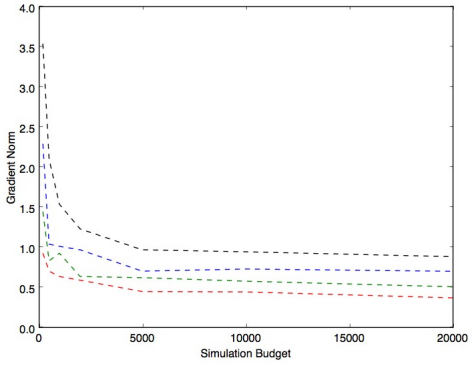
DENSCHNC



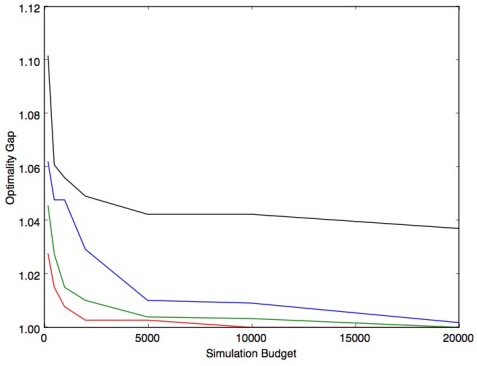
DENSCHNC



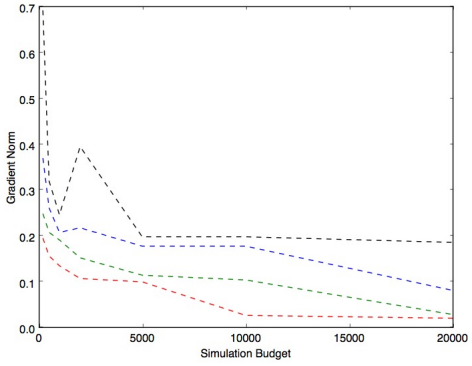
DENSCHND



DENSCHND

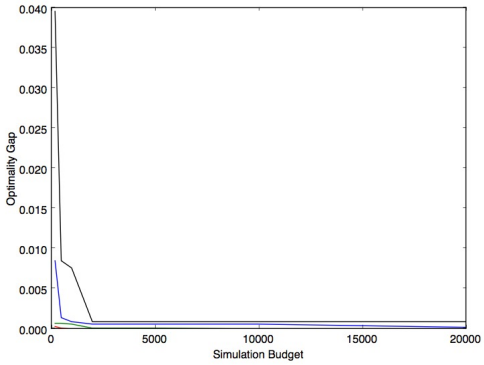


DENSCHNE

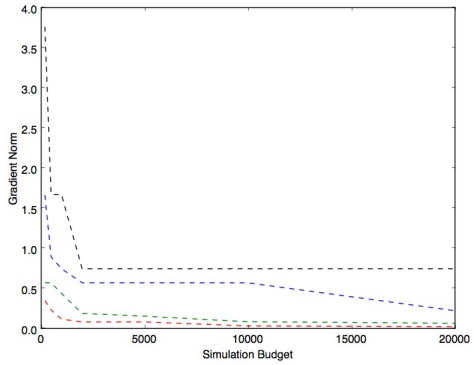


DENSCHNE

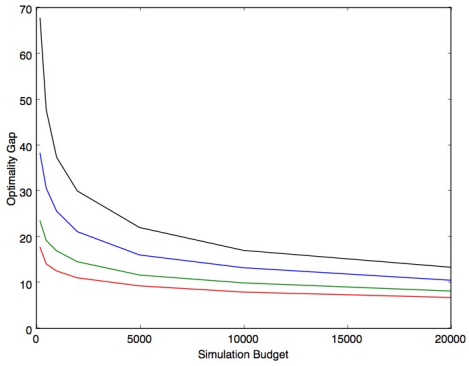
Figure B.3. Quantile plots of ASTRO for the functions DENSCHNC, DENSCHND and DENSCHNE.



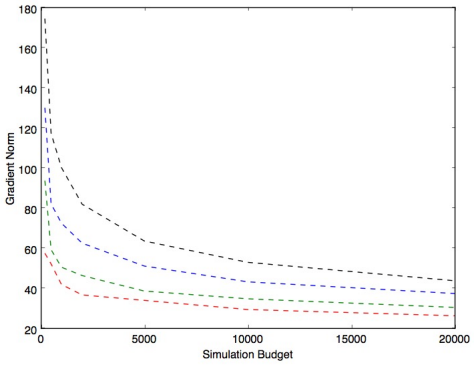
DENSCHNF



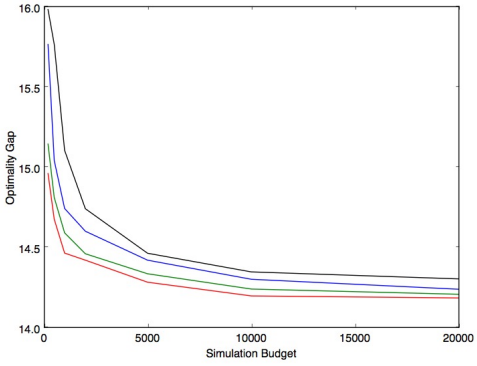
DENSCHNF



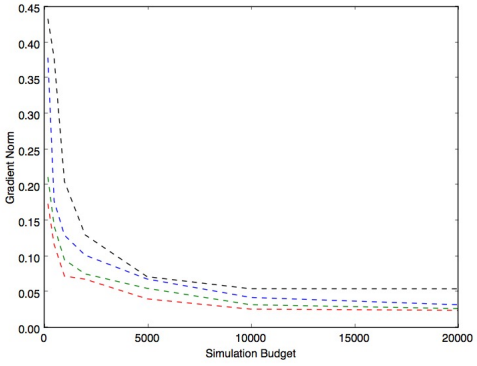
ENGVAL2



ENGVAL2

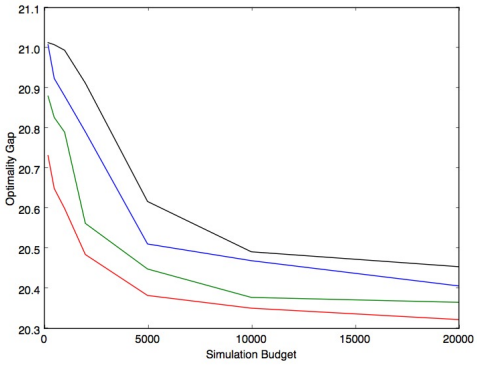


HATFLDD

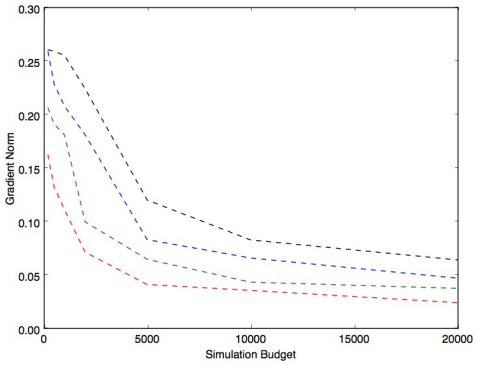


HATFLDD

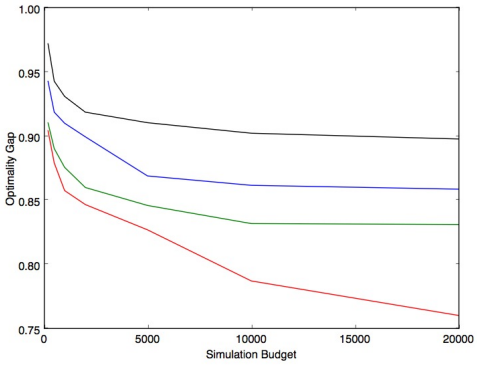
Figure B.4. Quantile plots of ASTRO for the functions DENSCHNF, ENGVAL2 and HATFLDD.



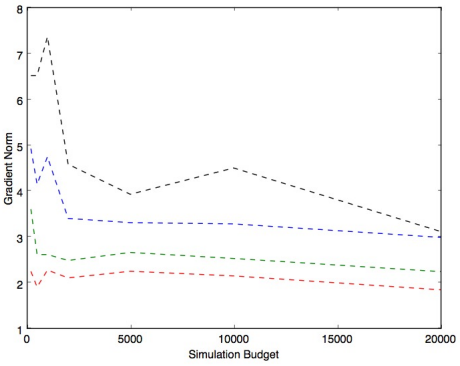
HATFLDE



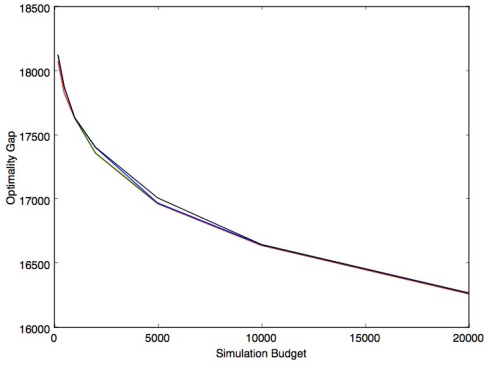
HATFLDE



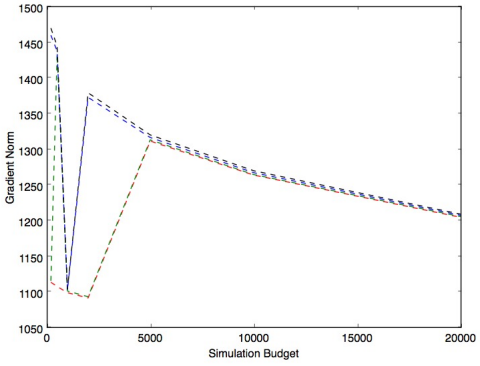
HELIX



HELIX

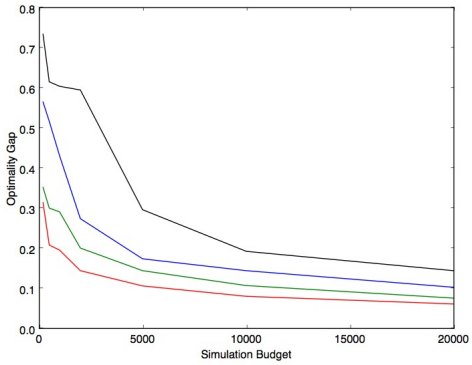


HIMMELBF

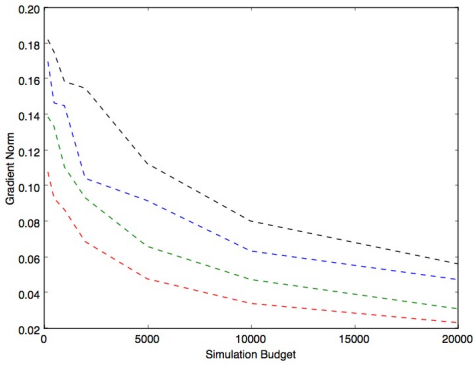


HIMMELBF

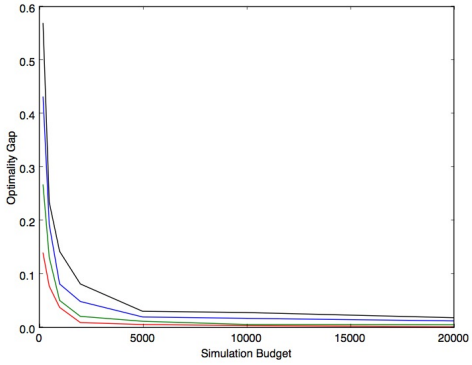
Figure B.5. Quantile plots of ASTRO for the functions HATFLDE, HELIX and HIMMELBF.



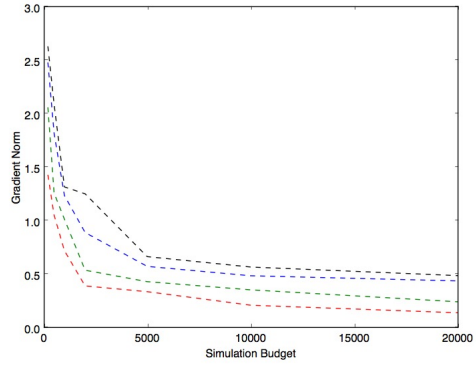
KOWOSB



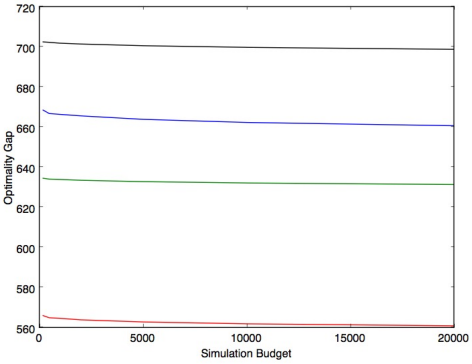
KOWOSB



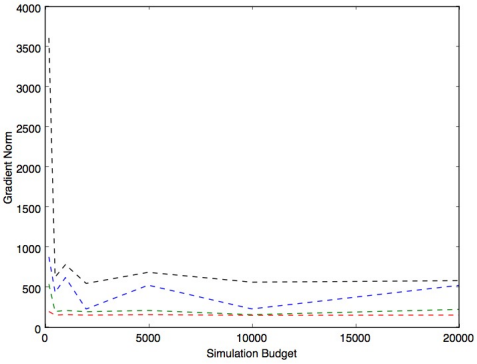
PALMER5C



PALMER5C

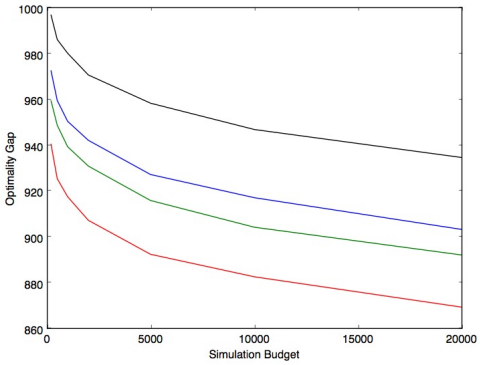


PALMER6C

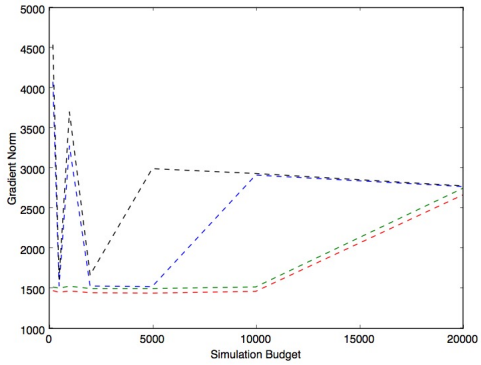


PALMER6C

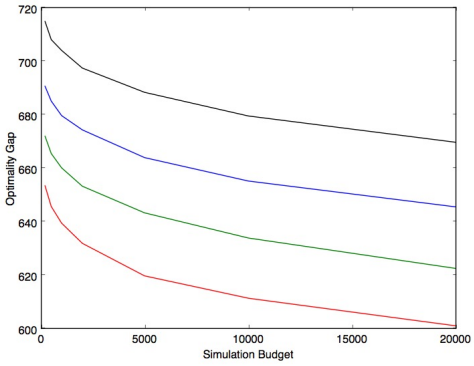
Figure B.6. Quantile plots of ASTRO for the functions KOWOSB, PALMER5C and PALMER6C.



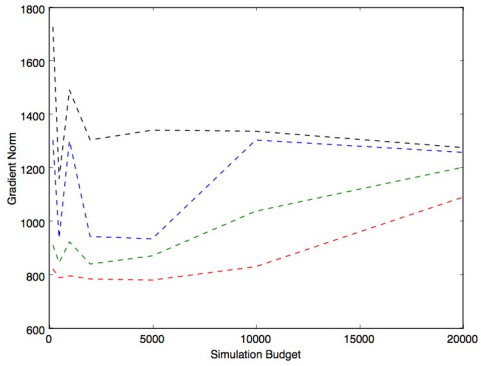
PALMER7C



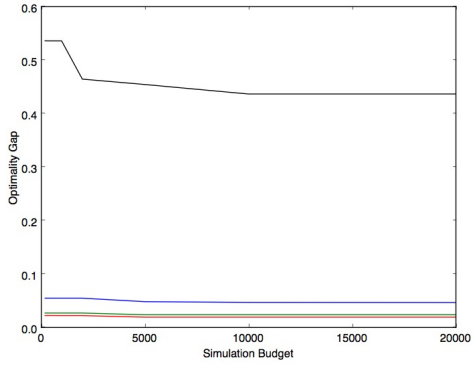
PALMER7C



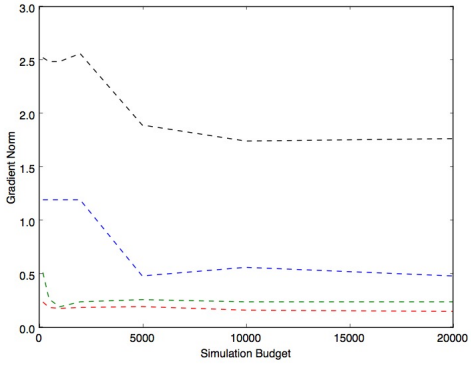
PALMER8C



PALMER8C

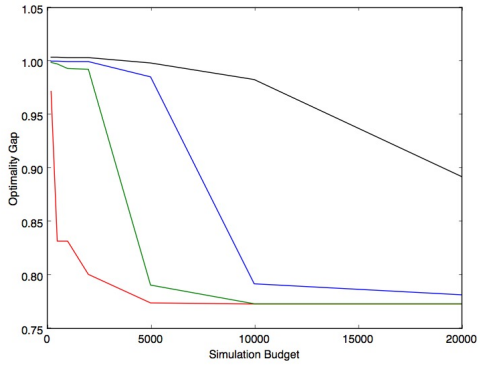


ROSENBR

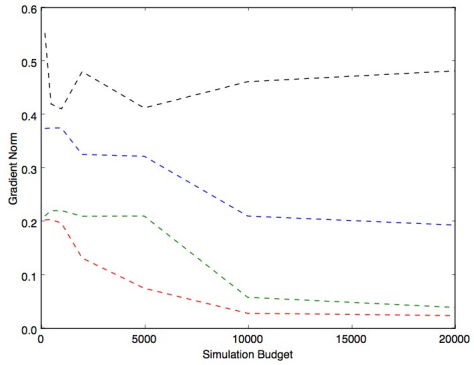


ROSENBR

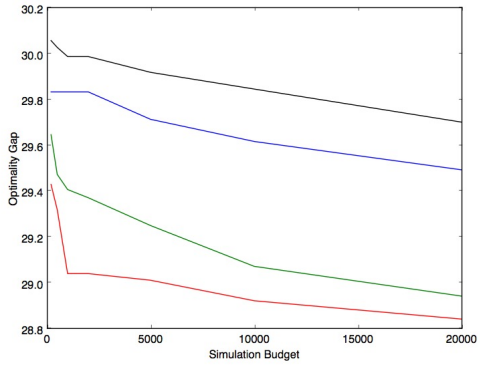
Figure B.7. Quantile plots of ASTRO for the functions PALMER7C, PALMER8C and ROSENBR.



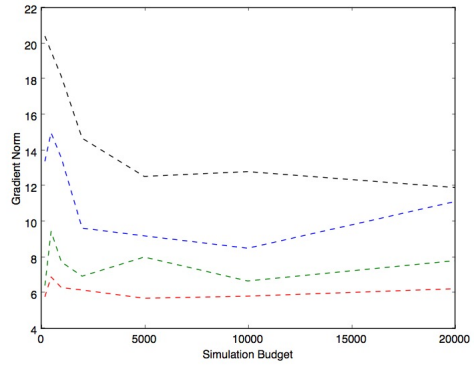
S308



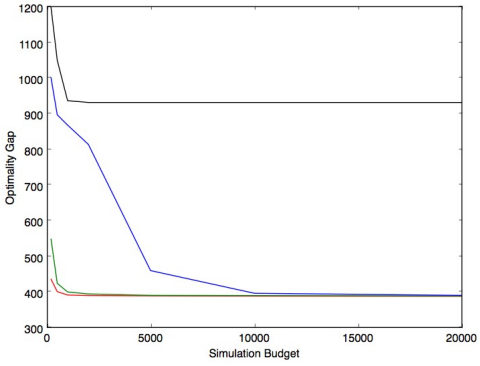
S308



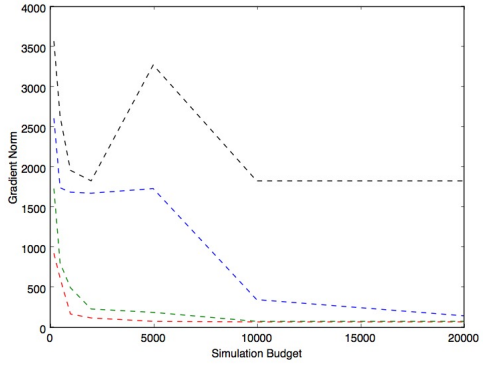
SINEVAL



SINEVAL



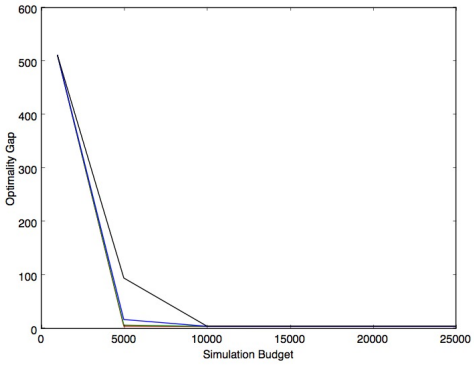
YFITU



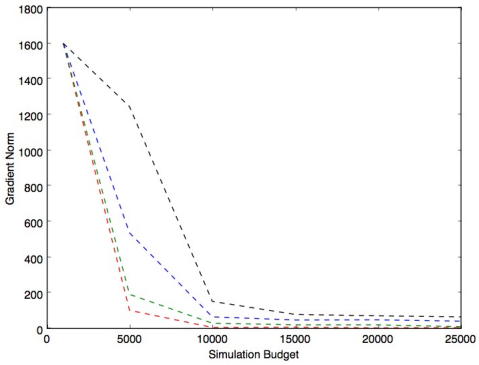
YFITU

Figure B.8. Quantile plots of ASTRO for the functions S308, SINEVAL and YFITU.

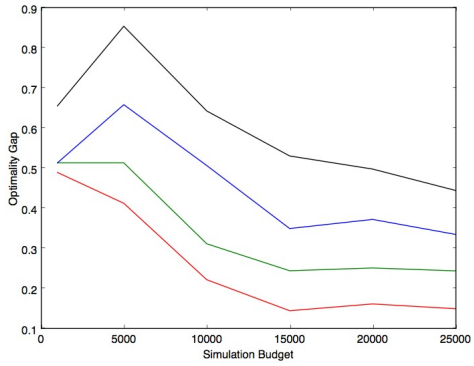
The quantile plots of ASTRO-DF are shown in the following. Again the red, green, blue and black plots are the 25%, 50%, 75% and 90% quantiles respectively. The solid lines (left plots) are the optimality gaps and the dashed lines (right plots) are the gradient norms.



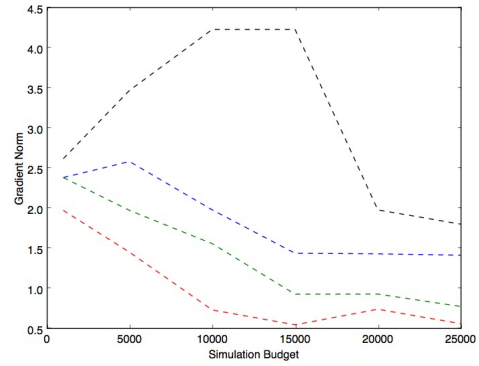
BEALE



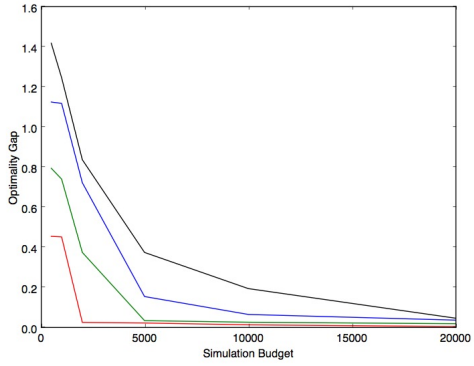
BEALE



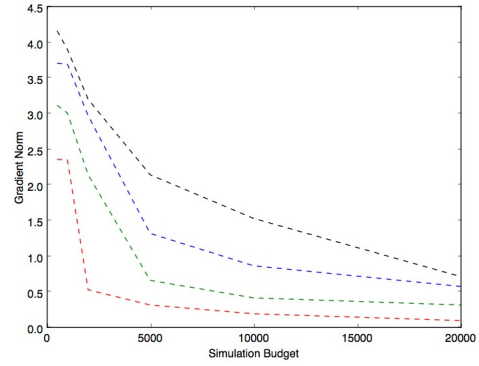
BIGGS6



BIGGS6

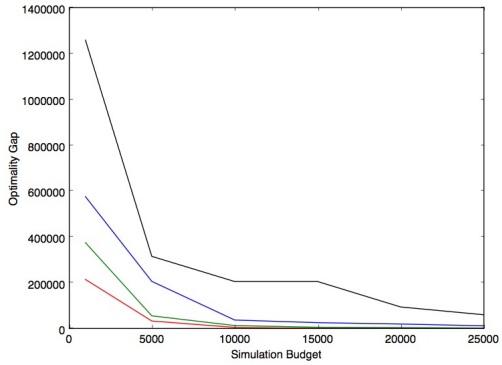


BOX3

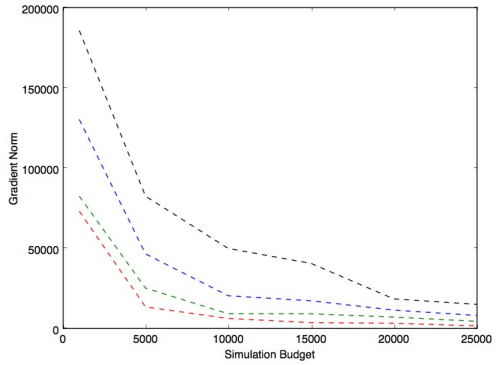


BOX3

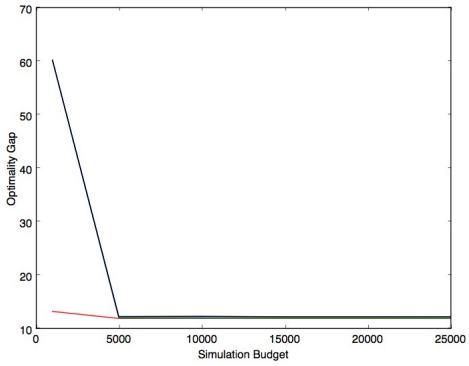
Figure B.9. Quantile plots of ASTRO-DF for the functions BEALE, BIGGS6 and BOX3.



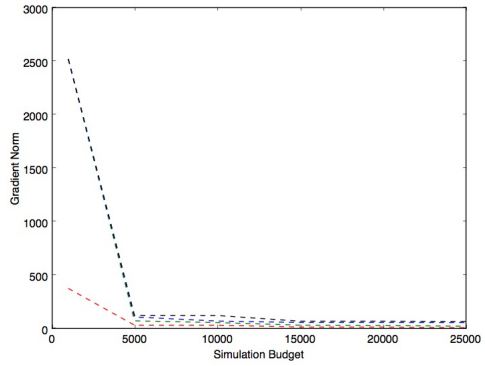
BROWNDEN



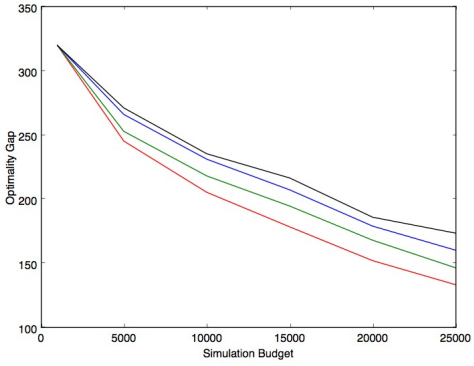
BROWNDEN



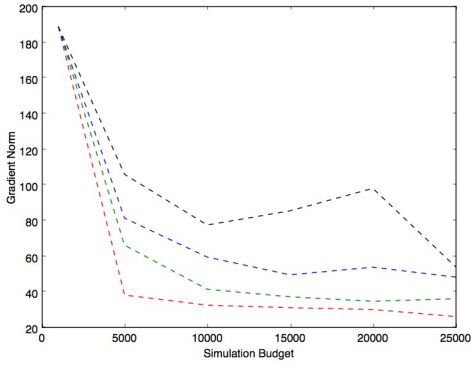
CUBE



CUBE

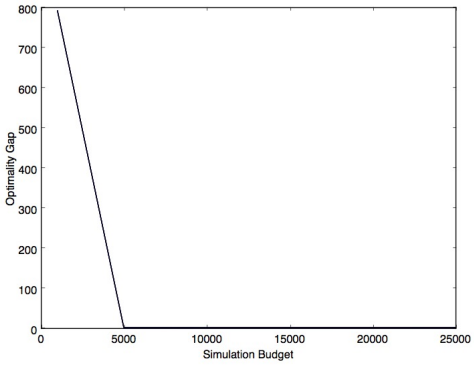


DENSCHNB

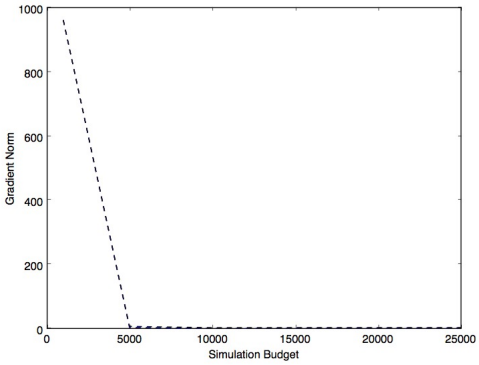


DENSCHNB

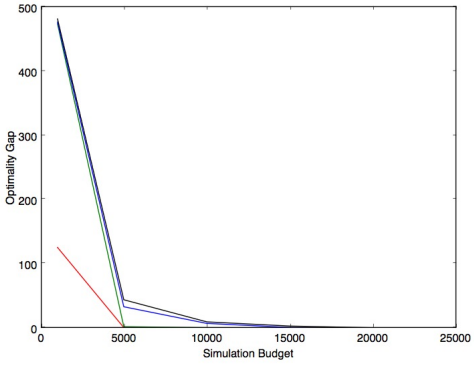
Figure B.10. Quantile plots of ASTRO-DF for the functions BROWNDEN, CUBE and DENSCHNB.



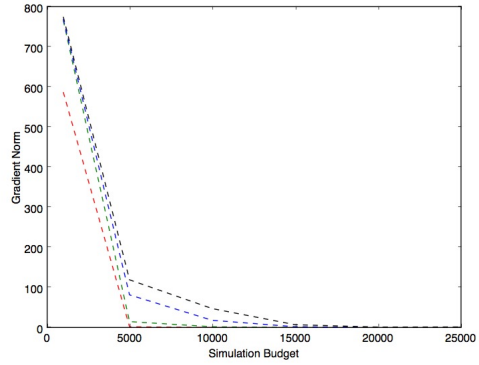
DENSCHNC



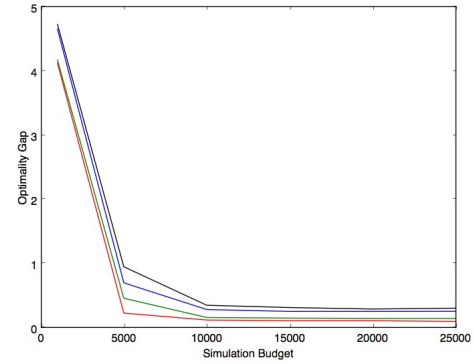
DENSCHNC



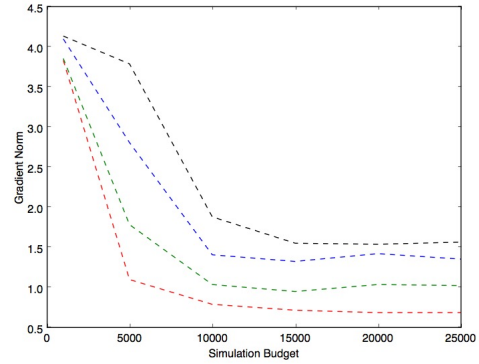
DENSCHND



DENSCHND

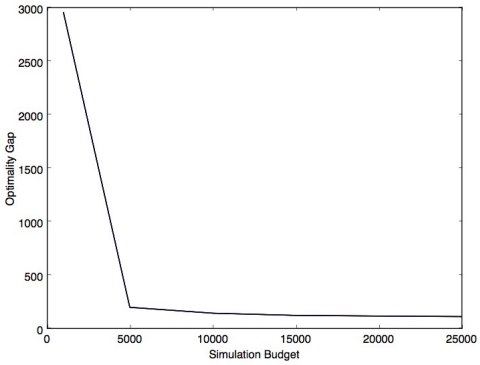


DENSCHNE

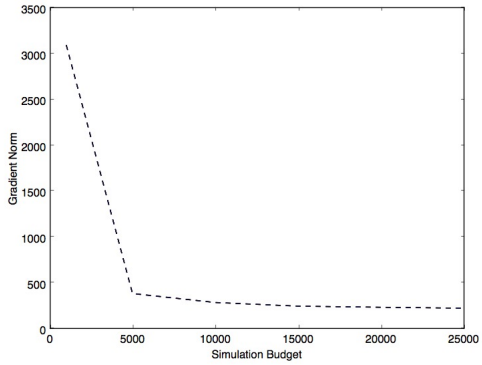


DENSCHNE

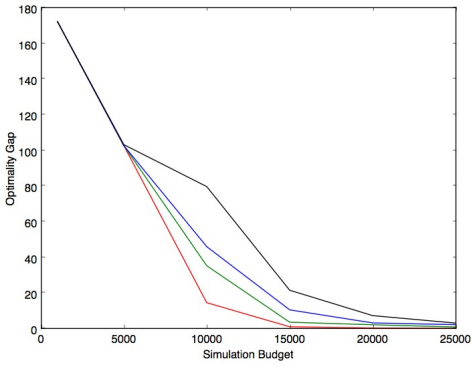
Figure B.11. Quantile plots of ASTRO-DF for the functions DENSCHNC, DENSCHND and DENSCHNE.



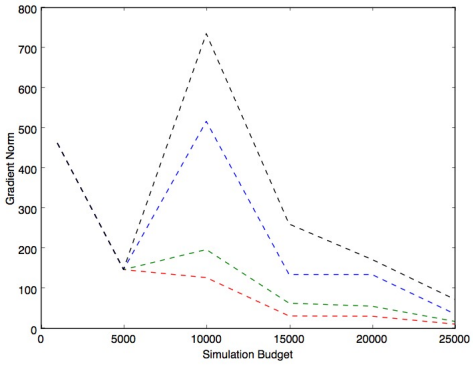
DENSCHNF



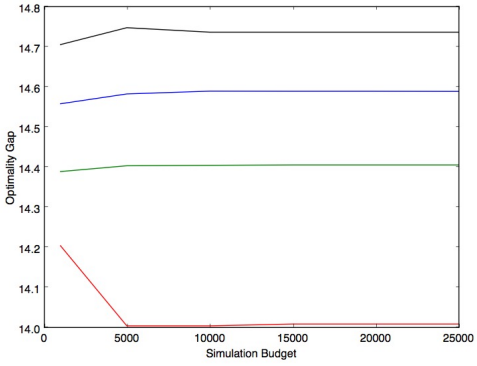
DENSCHNF



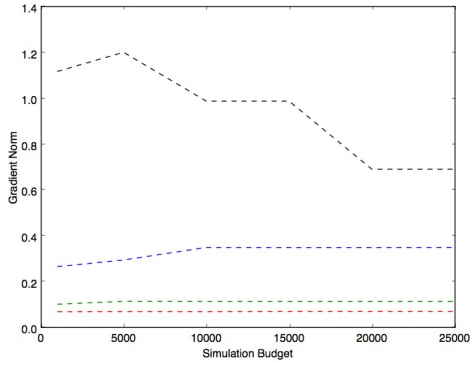
ENGVAL2



ENGVAL2

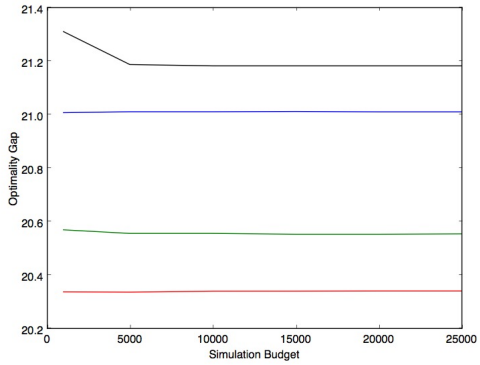


HATFLDD

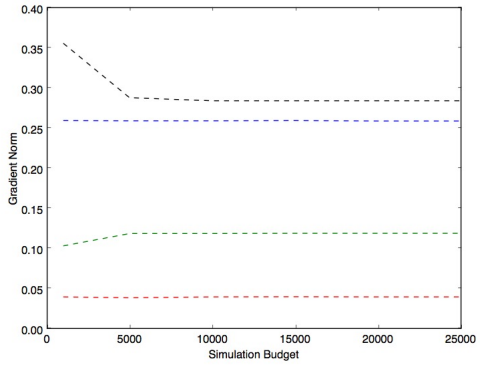


HATFLDD

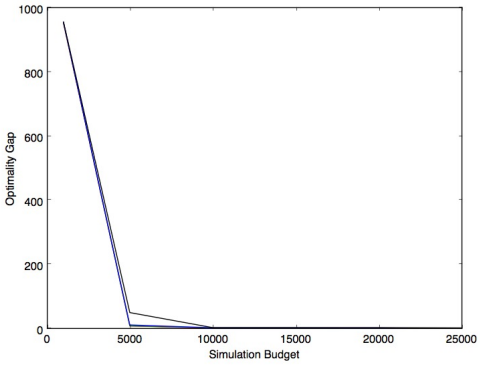
Figure B.12. Quantile plots of ASTRO-DF for the functions DENSCHNF, ENGVAL2 and HATFLDD.



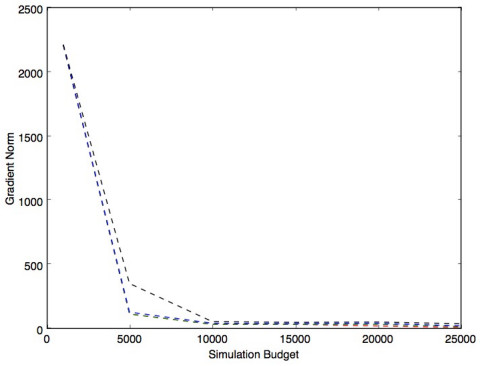
HATFLDE



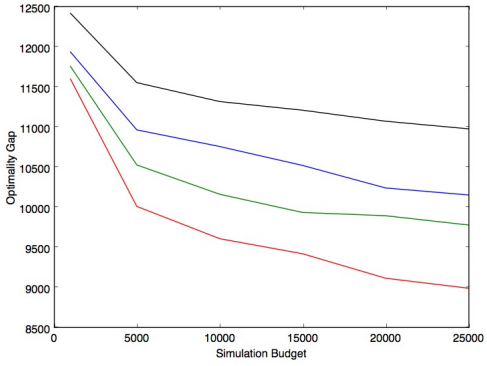
HATFLDE



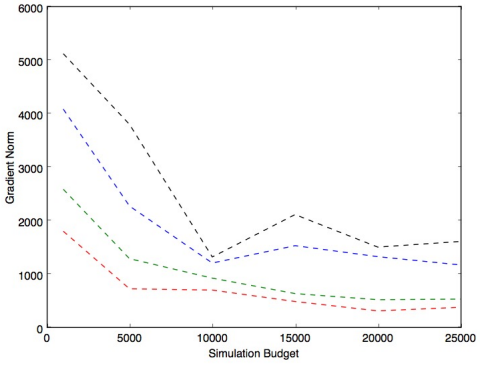
HELIX



HELIX

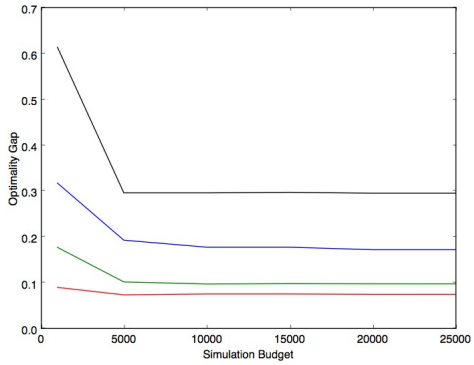


HIMMELBF

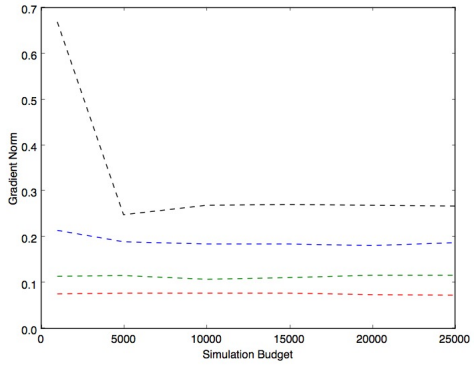


HIMMELBF

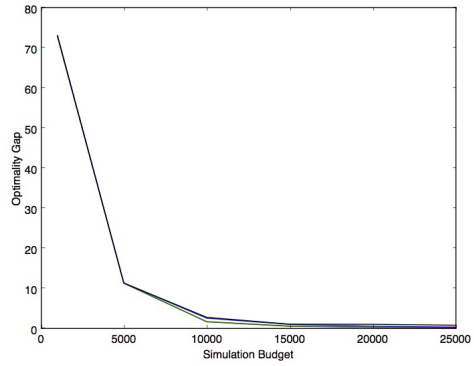
Figure B.13. Quantile plots of ASTRO-DF for the functions HATFLDE, HELIX and HIMMELBF.



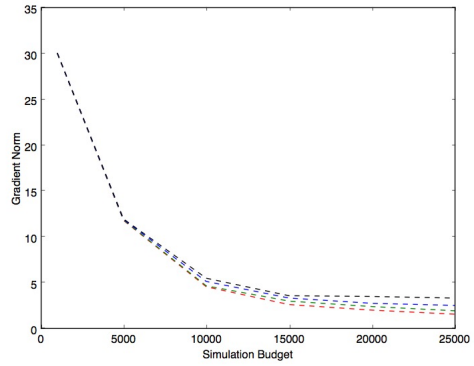
KOWOSB



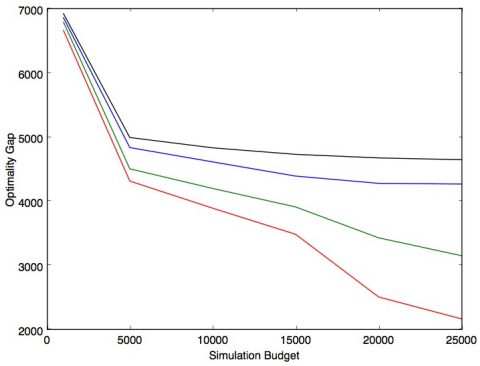
KOWOSB



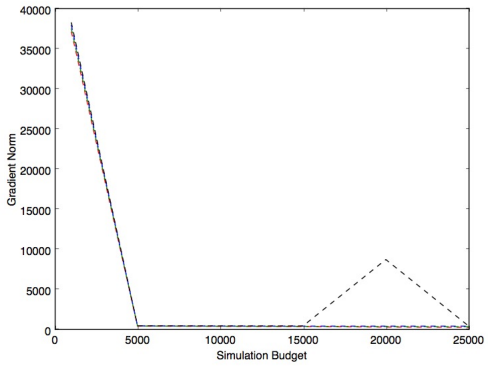
PALMER5C



PALMER5C

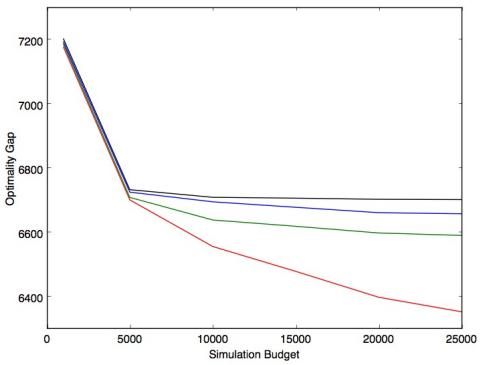


PALMER6C

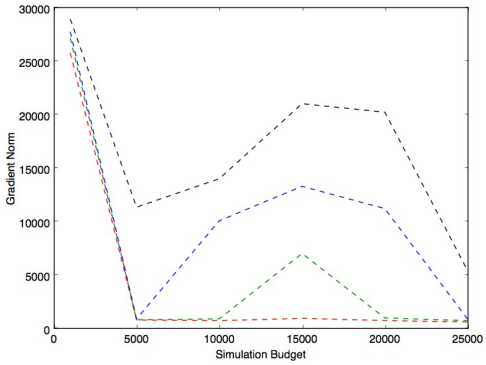


PALMER6C

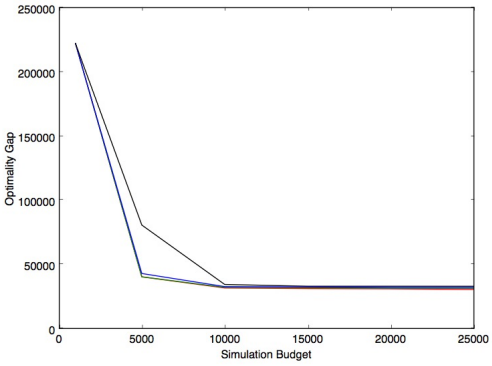
Figure B.14. Quantile plots of ASTRO-DF for the functions KOWOSB, PALMER5C and PALMER6C.



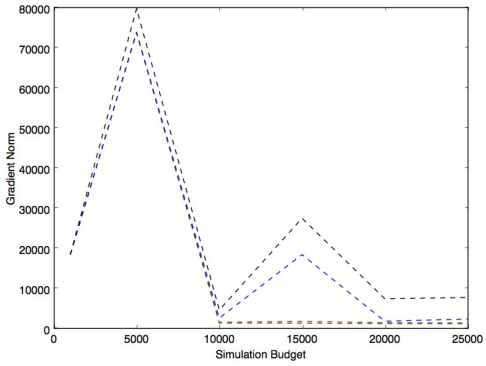
PALMER7C



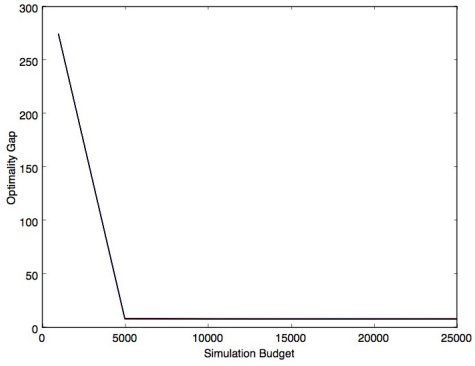
PALMER7C



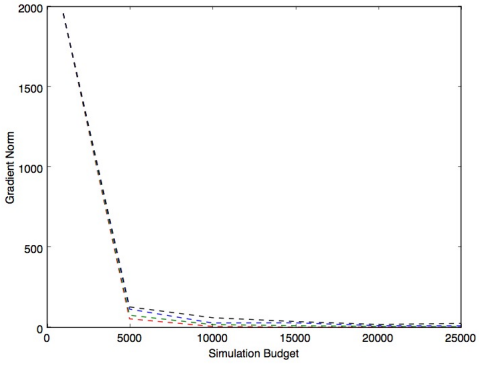
PALMER8C



PALMER8C

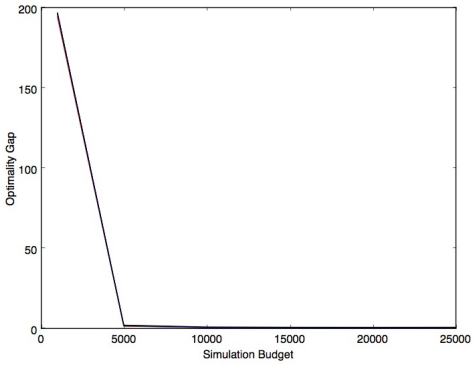


ROSENBR

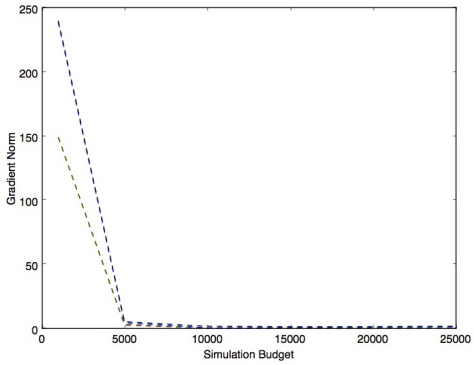


ROSENBR

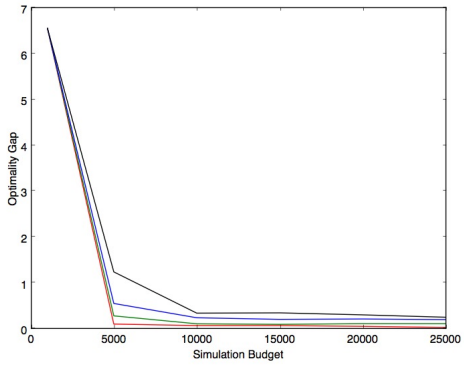
Figure B.15. Quantile plots of ASTRO-DF for the functions PALMER7C, PALMER8C and ROSENBR.



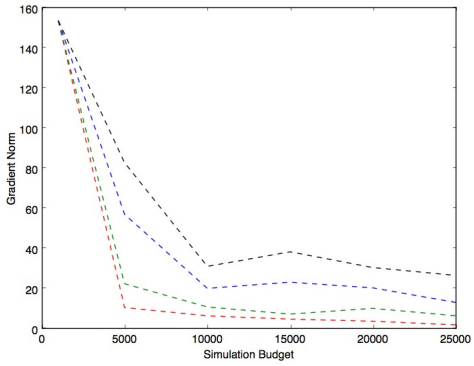
S308



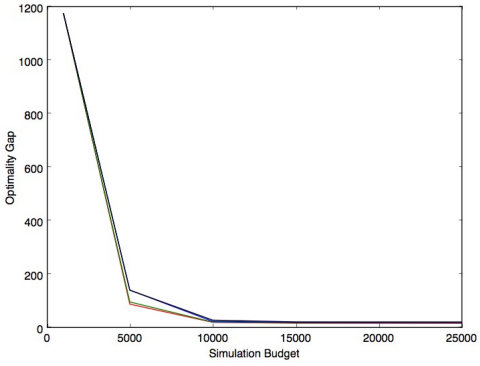
S308



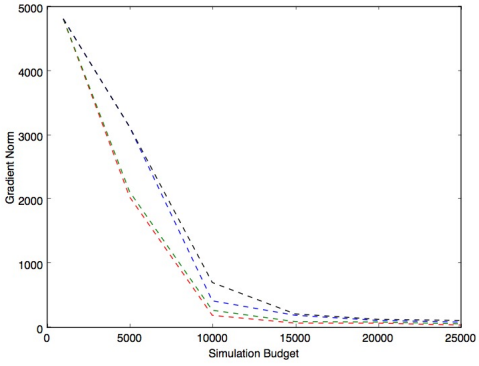
SINEVAL



SINEVAL



YFITU



YFITU

Figure B.16. Quantile plots of ASTRO-DF for the functions S308, SINEVAL and YFITU.

VITA

VITA

Sara Shashaani received her pre-university education in Tehran, Iran. She entered Iran University of Science and Technology for a Bachelor of Science degree in Industrial Engineering in 2003. Sara became involved with research on scheduling problems under the guidance of Dr. Mohammad Mahdavi. Simultaneously she enrolled in a Bachelor of Science program in Applied Computing in Southern Cross University and completed both degrees within a year in 2008 and 2009.

For graduate studies, Sara started her Master of Science program in the department of Industrial Engineering at Purdue University in 2009. Her Thesis title was "Chemotherapy Patient Scheduling and Uncertainty", in which Sara was co-advised by Dr. Mark Lawley and Dr. Hong Wan and mentored by Dr. Ayten Turkcan and Dr. Bruce Schmeiser. After completion of that program in 2011, Sara moved to Virginia Polytechnic Institute and State University to study in the area of Simulation Optimization in the department of Industrial and Systems Engineering and under the supervision of Dr. Raghu Pasupathy. Sara also worked with Dr. Madhav Marathe in Network Dynamics and Simulation Sciences Laboratory (NDSSL) in Virginia Bioinformatics Institute (VBI) to study epidemic forecast using simulation optimization. In the summer of 2012 Sara collaborated in a research project on hospital layout planning using simulation optimization, with Dr. Stefan Nickel, and Dr. Ines Arnolds in Karlsruhe Institute of Technology, and Dr. Christian Wernz in Virginia Tech. She completed another Masters degree in Operations Research from Virginia Tech and moved back to Purdue with Dr. Pasupathy to pursue her doctoral degree.

Along her academic journey, Sara served as a Teaching Assistant in a number of undergraduate and graduate level courses in science and engineering. She was been granted awards in teaching, poster presentation, travel and research both in Virginia Tech and Purdue University. Her poster presentation won the first prize at INFORMS annual meeting in 2013. She was also the recipient of the Ross Fellowship in 2014 from Purdue University.

Her previously published works include three journal papers, an approved publication in the proceedings of Winter Simulation Conference 2016, and a publications under review for SIAM journal of Optimization. Her most recent research with Dr. Pasupathy is also under preparation for publication in an esteemed academic journal.

Sara is joining the Guikema research group within the department of Industrial and Operations Engineering at the University of Michigan as a post-doctoral research fellow. She envisions to use theory and algorithms for simulation optimization problems in the interdisciplinary contexts of complex risk-prone systems.