Techniques for High-Dimensional Global Optimization and Response Surface Methodology

Megan Louise Scammell
School of Mathematics
Cardiff University

2022

A thesis submitted for the degree of Doctor of Philosophy

Summary

This thesis aims to improve the efficiency and accuracy of optimization algorithms. High-dimensional optimization problems are frequently encountered in many practical situations due to the advancements in technology and availability of big data. Also, the analytical form of a black-box objective function is unknown, adding to the challenging nature of high-dimensional optimization problems.

Multistart is a celebrated global optimization algorithm that involves sampling points at random from the feasible domain and applying a local optimization algorithm to find the corresponding local minimizer. The main drawback of multistart is the low efficiency since the same local minimizers may be found repeatedly. A vital research contribution in this thesis improves the efficiency of multistart for high-dimensional optimization problems by reducing the number of local searches to the same local minimizers.

Ensuring local optimization methods are reliable and accurate when only objective function values containing errors are available is an important area of research. Specifically, the central focus is on the first phase of the Box-Wilson (BW) algorithm, a response surface methodology (RSM) strategy. The first phase of BW consists of performing a succession of moves toward a subregion of the minimizer. A significant research contribution in this thesis enhances the accuracy of the first phase of BW and RSM, in general, for high-dimensional optimization problems by employing a different choice of search direction.

Producing high-quality software is vital to ensure accurate research investigations and to allow other researchers to apply the software, which is an additional research contribution demonstrated in this thesis. Furthermore, increasingly complex high-dimensional optimization problems are encountered in various areas of machine learning. Therefore, the development of advanced optimization methods is essential to the progression of many machine learning algorithms. Consequently, the final research contribution in this thesis outlines the potential enhancements to optimization methods applied within various areas of machine learning.

Acknowledgements

I would like to thank my supervisors, Dr. Jonathan Gillard and Prof. Anatoly Zhigljavsky, for their excellent guidance and support throughout my studies. You have always provided me with invaluable advice and feedback, which has been instrumental in the completion of this work. I would also like to extend my thanks to Dr. Vince Knight who kindly offered advice for using and developing software, which helped me tremendously with carrying out research.

I would like to express my special thanks and gratitude to my parents, who have always supported and helped me whilst studying. This would not have been possible without you both and I will always be extremely grateful for all that you have done. I would also like to extend my special thanks to my fiancé Matthew for his constant encouragement, patience and support.

I would also like to thank the friends I have met during the course of my studies who have made undergraduate and postgraduate education a special and memorable experience. Finally, I would like to thank the Engineering and Physical Sciences Research Council (EPSRC) for providing funding during my postgraduate studies which enabled me to carry out this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation of thesis

Optimization is frequently applied in many practical situations to find the maximum or minimum of some objective function defined on a feasible domain. Throughout this thesis, the minimization problem will be considered.

There have been significant advancements in technology and storage capabilities in recent times, and consequently, substantial amounts of data are widely available. Therefore, high-dimensional optimization problems are often encountered and are considered in this thesis. The analytical form of the objective function may also be unknown in many practical situations, which further enhances the difficulty of optimization problems.

The type of optimization algorithm employed depends on the underlying structure and behaviour of the objective function. Specifically, many local minima may exist if the objective function is assumed to be multimodal. A local minimum is the smallest function value observed on a subregion of the feasible domain. If a local minimum is the smallest function value observed on the entire feasible domain, then the local minimum is the global minimum. On the other hand, if the objective function is unimodal then a single local minimum exists, which is also the global minimum. Global optimization algorithms aim to find the global minimum of an objective function and often consist of a global and local phase. The global phase involves exploring the feasible domain by sampling points, and the local phase employs a local optimization algorithm to make local improvements to the sampled points. The choice of local optimization algorithm often depends on whether the objective function is deterministic or stochastic. The objective function is deterministic if the same output is repeatedly produced when evaluated with a particular input. Conversely, the objective function is stochastic if different outputs are produced when evaluated with a particular input.

The research contributions in this thesis aim to improve the efficiency and accuracy of algorithms for high-dimensional optimization problems. In addition, the enhanced algorithms also have the potential to be applied with black-box objective functions. Suppose the objective function is deterministic, multimodal and applying a local optimization algorithm to many sampled points from the feasible domain results in repeatedly finding the same local minima. In that case, it is desirable to apply the local optimization algorithm to promising points only, which may lead to the discovery of a new local or global minimum. As a result, the efficiency of a global optimization algorithm can be significantly improved, which is a key research area explored in this thesis. On the other hand, suppose the objective function is stochastic for high-dimensional optimization problems. Ensuring local optimization methods are reliable and accurate when only function values containing errors are available is another key area of research investigated in this thesis.

Improved computational resources and the availability of big data in recent times have presented many opportunities to enhance progression in various fields. Specifically, the ability to efficiently and effectively extract knowledge and trends from a substantial amount of data is a crucial area of research. Machine learning involves learning a model from observed data to predict an output or trend for unseen data. The progression of many machine learning algorithms involves solving increasingly complex high-dimensional optimization problems, and as a result, a great deal of interest is devoted to optimization methods. The development of advanced optimization methods has influenced the significant progress made in machine learning. Therefore, a discussion on how each research contribution in this thesis may be applied to enhance optimization methods used in various areas of machine learning will be presented.

1.2 Outline of thesis

The structure of this thesis is summarized as follows.

- Chapter 1 motivates and illustrates the key research contributions presented in this thesis.
- Chapter 2 provides a literature review of global and local optimization algorithms, properties of high-dimensional objects and distributions, and the connection between machine learning and optimization.
- Chapter 3 outlines the multistart with early termination of descents (METOD) algorithm, which has been published in [144]. METOD aims to reduce the

number of repeated local searches to the same local minimizer. A variety of enhancements to the original METOD algorithm in [144] have been implemented in Chapter 3 to improve the efficiency and accuracy of identifying new local minima. In addition, clear recommendations for the METOD algorithm parameters are provided, along with numerical investigations. Furthermore, the implementation of the METOD algorithm in Python is discussed, which allows other researchers to apply the software for high-dimensional global optimization problems.

- Chapter 4 presents a different choice of search direction in high-dimensional response surface optimization, where the objective function to be optimized is stochastic. Hence, an important area of research is to ensure that descent directions produced in stochastic local optimization algorithms are accurate and reliable. Specifically, the key focus of Chapter 4 is the Box-Wilson (BW) algorithm, which is a response surface methodology (RSM) strategy and consists of two phases. The first phase of BW consists of performing a succession of moves toward a subregion of the minimizer. The second phase of BW involves fitting a locally quadratic model of the response function to estimate the location of the minimizer. Chapter 4 investigates a different choice of descent direction for the iterative updates within the first phase of BW for high-dimensional stochastic optimization problems.
- Chapter 5 illustrates the best practices adopted for the development of software for research contributions presented in Chapters 3 and 4. The purpose of this is to ensure that the software and outputs produced from the software are reliable and accurate.
- In Chapter 6, all work outlined in the previous chapters is discussed and further areas of research are identified.

1.3 Research contributions

The key research contributions of this thesis are,

1. To improve the efficiency of multistart for high-dimensional optimization problems by reducing the number of local searches to the same local minimizers.
2. To enhance the accuracy of high-dimensional response surface optimization algorithms by employing a different choice of descent direction.
3. To produce high-quality software for the research contributions in Chapters 3 and 4, which ensures numerical results are accurate and other researchers are able to apply the software.

4. To outline the potential enhancements to optimization methods applied within various areas of machine learning.

1.4 Notation

All notation used is outlined within each chapter. Furthermore, notation is specific to each chapter and can vary between different chapters. In general, if a single point is generated and applied with an optimization algorithm, the point is denoted as $x = (x_1, \dots, x_d)^T$, where d is the dimension. Alternatively, if many points are generated and applied with an optimization algorithm, the collection of points are denoted as $x_n = (x_{n_1}, \dots, x_{n_d})^T$, where $n = 1, \dots, N$ is the point index and N is the total number of points.

Chapter 2

Literature review

This chapter is summarized as follows.

- Section 2.1 outlines the purpose of the literature review.
- Section 2.2 provides an overview of the global optimization problem and the approaches adopted by many global optimization algorithms to solve the corresponding problem. In addition, the motivation for research contributions in Chapter 3 is presented, which is to improve the efficiency of multistart, a celebrated global optimization algorithm.
- In Section 2.3, an overview of deterministic local optimization methods is presented, where exact information on objective function values and derivatives is assumed to be available.
- Section 2.4 summarises stochastic local optimization methods, namely, stochastic approximation (SA) and response surface methodology (RSM), where noisy objective function values are utilized. The focus of this section is to demonstrate how the application of RSM can be expanded to high-dimensional optimization problems, which is the main research contribution in Chapter 4.
- In Section 2.5, the counter-intuitive properties of high-dimensional objects and distributions are presented.
- Section 2.6 portrays the immense value and significance of optimization in various areas of machine learning, which illustrates the great importance and potential applications of research contributions in Chapters 3 and 4.

2.1 Introduction

This chapter aims to motivate and provide context for the research contributions presented in Chapters 3 and 4. That is, enhanced algorithms to efficiently and accurately solve high-dimensional black-box optimization problems, where the underlying objective function is either assumed to be deterministic or stochastic. If the objective function is deterministic, exact information on objective function values and derivatives is available. On the other hand, the objective function is stochastic if objective function values contain errors.

Section 2.2 discusses the problem of global optimization and the approaches adopted by many global optimization algorithms to solve the corresponding problem. Many global optimization algorithms utilize local optimization algorithms to make local improvements. Hence, Sections 2.3 and 2.4 present various deterministic and stochastic local optimization algorithms, which are applied depending on whether the underlying objective function is assumed to be deterministic or stochastic. Sections 2.2 - 2.4 also demonstrate the challenges often encountered with high-dimensional black-box optimization problems. In addition, Section 2.5 presents the counter-intuitive properties of high-dimensional objects and distributions, further illustrating the challenging aspects of high-dimensional black-box optimization. Section 2.6 portrays the importance of optimization in various areas of machine learning.

2.2 Global optimization

2.2.1 Overview

Global optimization is an extensive and fast-growing area of research. Many procedures are associated with global optimization, such as examining and investigating optimization problems, developing various algorithms to perform optimization, and developing software to allow automation of algorithms and analysis. Global optimization is frequently utilized in various fields, including engineering, natural sciences, medicine and machine learning (see [1, 45, 96, 99, 104]). Developments in technology and science involve solving increasingly difficult global optimization problems with a large number of variables, where the analytical form of the objective function to be optimized is often unknown. These problems are called high-dimensional black-box global optimization problems since the objective function to be optimized is black-box. This section outlines the approach of many global optimization algorithms and is based on the monographs [139, 140, 141, 143].

2.2.2 Formulation of global optimization problem

Consider the following global minimization problem,

$$f^* = \min_{x \in \mathfrak{X}} f(x), \tag{2.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is an objective function that is continuous on the feasible domain $\mathfrak{X}$. In some cases, objective function values may contain errors, which is often the case when the objective function f is a regression function [141]. A global minimizer is any point $x^* \in \mathfrak{X}$ which satisfies,

$$x^* = \operatorname*{argmin}_{x \in \mathfrak{X}} f(x). \tag{2.2}$$

By [139, 141, 143], a global optimization algorithm constructs a sequence of points $x_n \in \mathfrak{X}$ $(n = 1, \dots, N)$ such that,

$$y_N = \min_{n=1,\dots,N} f(x_n)$$

approaches f^* as N increases. The structure of the feasible domain $\mathfrak{X}$ and the objective function f may influence the ability of solving (2.1). Throughout, it is assumed that $\mathfrak{X} \in \mathbb{R}^d$ is continuous and high-dimensional. The search space of the feasible domain $\mathfrak{X}$ increases exponentially as the dimension d grows. For example, suppose $\mathfrak{X} = [0, 1]^d$, and a grid of equidistant points to cover $\mathfrak{X}$ is obtained. If $d = 1$, then 10 points placed at intervals of 0.1 cover $\mathfrak{X}$ evenly. Hence, the number of points in the grid to cover $\mathfrak{X}$ for d dimensions is 10^d and thus, the number of points required increases exponentially as d grows. In addition, Section 2.5 provides an overview of high-dimensional geometry to illustrate the contradictory behaviour of sampling points from a high-dimensional unit cube and Gaussian distribution compared to the known 2-dimensional behaviour.

A local minimizer x_l^* $(l = 1, 2, \dots)$ exists if

$$f(x_l^*) < f(x), \tag{2.3}$$

for all $x \in A_l \subset \mathfrak{X}$, where A_l is a region of attraction of the local minimizer x_l^* and $f(x_l^*)$ is the local minimum. If the objective function f has just one local minimizer in the feasible domain $\mathfrak{X}$, the objective function is unimodal and the local minimizer is the global minimizer. Otherwise, if the objective function has many local minimizers in the feasible domain $\mathfrak{X}$, then the objective function is multimodal. The local minimizer x_l^* is also the global minimizer x^* if (2.3) holds for all $x \in \mathfrak{X}$. Each time local search is applied with $x \in A_l$, the local minimizer x_l^* associated with region of attraction A_l will be found. Figures 2.1 and 2.2 portray an example of a unimodal and multimodal objective function defined on the feasible domain $\mathfrak{X}$.

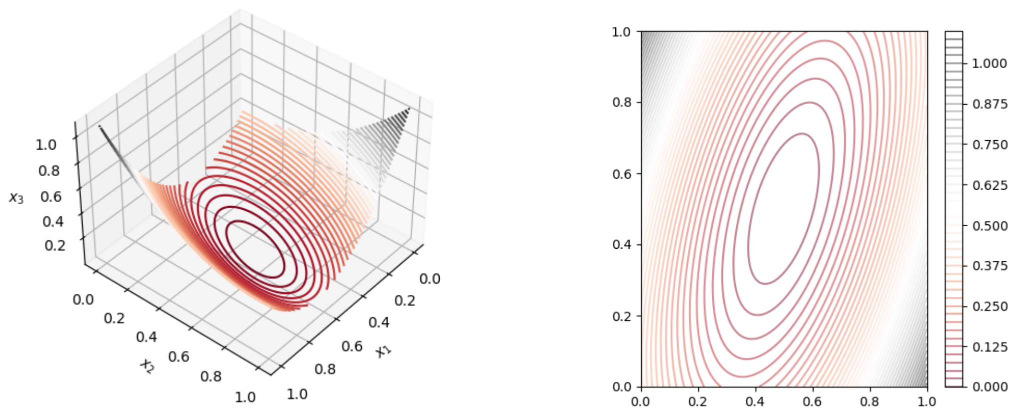


Figure 2.1: Plot of a unimodal function (left) and corresponding contour plot (right), where $x^* = (0.5, 0.5)^T$ and $\mathfrak{X} = [0, 1]^2$.

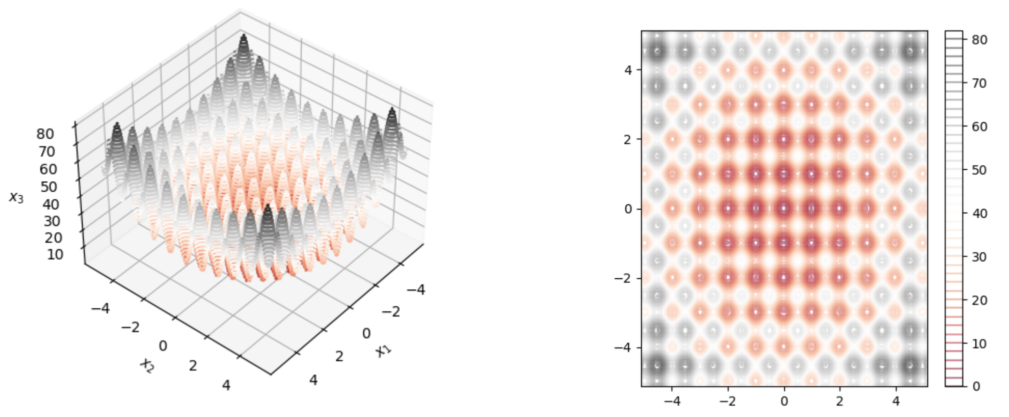


Figure 2.2: Plot of a multimodal function (left) and corresponding contour plot (right), where $x^* = (0, 0)^T$, $\mathfrak{X} = [-5, 5]^2$ and the objective function f is called Rastrigin's function.

2.2.3 Properties of global optimization algorithms

Suppose x is a point sampled from $\mathfrak{X}$. The objective function value $f(x)$ can be improved by applying several iterations of a local search method to obtain a sequence of points $x^{(k)}$ $(k = 0, 1, \dots)$ that approach a local minimizer x_l^*. The sequence $x^{(k)}$ can be constructed as follows,

$$x^{(k+1)} = x^{(k)} - \gamma^{(k)} s^{(k)}, \tag{2.4}$$

where $x^{(k)} \in \mathbb{R}^d$ is a point at the k-th iteration of a local search method, $s^{(k)}$ is the search direction and $\gamma^{(k)}$ is the step length. If exact information on objective function values and derivatives is available, deterministic local optimization procedures in Section 2.3 are utilized to obtain $s^{(k)}$ and $\gamma^{(k)}$. Otherwise, if the objective function values contain errors, stochastic local optimization procedures discussed in Section 2.4 are used.

Many global optimization algorithms incorporate global and local search to improve the likelihood of finding the global minimum. The global search involves exploring and sampling points scattered over the feasible domain $\mathfrak{X}$. The local search involves making local improvements to the sampled points by employing some local optimization method. The efficiency of global optimization algorithms often depends on the trade-off between global and local search. A suitable balance between global and local search depends on the complexity of $\mathfrak{X}$, information of f (obtained before and during the search), and also the computational cost of evaluating the gradient of f [139, 140, 143].

Global optimization algorithms can be mainly categorized as deterministic or stochastic. The focus throughout will be on stochastic global optimization (see [139, 143]). Stochastic global optimization methods are described in [143] as methods for solving global optimization problems where the problem itself is stochastic (i.e. objective function values contain errors), the algorithm contains stochastic elements or a combination of the both.

Global random search (GRS) is a class of stochastic global optimization algorithms. Suppose x_n $(n = 1, \dots, N)$ is a collection of points, where x_n has some probability distribution P_n. The GRS class consists of algorithms that generate x_n $(n = 1, \dots, N)$, where at least one P_n is non-degenerate [139, 140, 143]. In addition, [139, 140, 143] states that the probability distributions P_n with $n \geq 2$ may depend on previous starting points $x_1, \dots, x_{n-1}$ and on corresponding objective function evaluations $f(x_1), \dots, f(x_{n-1})$.

Many GRS algorithms employ a selection of the principles outlined in [139, 140, 143], which are provided in Table 2.1. Popular GRS algorithms are stated in [140, 143] and include pure random search, markovian global search, pure adaptive search, random multistart and population based search. Advantages of GRS algorithms are outlined in [139, 140, 143] and include, ease of implementation and insensitivity to the structure of the feasible domain, volatility of the objective function, and size of dimension. However, the main drawback of GRS algorithms is the slow convergence rate [139, 140, 143]. Hence, improving the efficiency of GRS algorithms is a key area of research in stochastic global optimization.

Low efficiency is often encountered when applying random multistart (also known

Principle	**Description**
P1	*Sample points at random and evaluate f.*
P2	*Random coverage of the feasible domain.*
P3	*Use of local optimization techniques.*
P4	*Apply heuristics to avoid a collection of points around certain local minima.*
P5	*Selection of new points in the vicinity of good previous points.*
P6	*Use of statistical inference.*
P7	*Decrease of randomness in the selection rules for new points.*

Table 2.1: A variety of principles from [139, 140, 143] which are utilized within a broad range of GRS algorithms.

as multistart) since local search is applied to each sampled point generated within the global phase. Hence, the same local minima may be found repeatedly, and as a result, computational efforts are not used resourcefully to locate the global minimum. Consequently, the main focus of Chapter 3 is to illustrate an efficient version of multistart, which aims to reduce the number of local searches to the same local minimizers. As a result, the efficiency of multistart can be significantly improved, especially when the dimension of the problem is very large.

2.3 Deterministic local optimization

2.3.1 Overview

Deterministic local optimization algorithms assume exact information on objective function values and derivatives is available, and are often employed within global optimization algorithms. The focus of this section is on deterministic local optimization procedures that obtain $x^{(k)}$ $(k = 1, 2, \ldots)$ according to the iterative procedure (2.4), where various methods to compute the search directions $s^{(k)}$ and step lengths $\gamma^{(k)}$ are discussed.

2.3.2 Step length $\gamma^{(k)}$

Suppose $s^{(k)}$ is a search direction for the iterative update (2.4) and satisfies,

$$-\nabla f(x^{(k)})^T s^{(k)} < 0.$$

Then $\gamma^{(k)}$ can be selected by minimizing the following one-dimensional optimization problem,

$$\gamma^{(k)} = \operatorname*{argmin}_{\gamma>0} f(x^{(k)} - \gamma s^{(k)}). \tag{2.5}$$

It can be computationally expensive to identify an exact value of $\gamma^{(k)}$ that satisfies (2.5). Therefore, a solution of (2.5) is often approximated. Typically, line search algorithms test a variety of guesses γ within (2.5), and the most suitable γ is selected as $\gamma^{(k)}$ when some stopping criterion is met. Specifically, [94, Chpt. 3.1] states that line search algorithms consist of two phases. The first phase brackets an interval of suitable values of γ to test, and the second phase applies bisection or interpolation to select an appropriate step length $\gamma^{(k)}$ within the bracketed interval. An extensive summary of line search methods is provided in [20, Chpt. 7], [77, Chpt. 8.1] and [94, Chpt. 3], and the following contains a brief overview of the line search methods discussed.

Golden section search

Suppose that (2.5) is unimodal on a bracketed interval $[0, b]$. The interval can be obtained by applying bracketing methods, which are discussed in [20, Chpt. 7.7]. The golden section search method was first introduced in [60] to approximate the minimizer of a one-dimensional unimodal function $f : [a, b] \to \mathbb{R}$ without the use of derivatives (see [20, Chpt. 7.2] and [77, Chpt. 8.1] for more details). Consequently, golden section search can be used to determine $\gamma^{(k)}$. Consider the points

$$x_1 = b + \frac{a-b}{\phi} \tag{2.6}$$

$$x_2 = a + \frac{b-a}{\phi} \tag{2.7}$$

where $\phi = 1.618$ is the golden ratio. By construction, the length of interval $[a, x_2]$ is the same as interval $[x_1, b]$.

Consider the following updates performed at each iteration of golden section search. If $f(x_1) < f(x_2)$, then $b \leftarrow x_2$, $x_2 \leftarrow x_1$ and x_1 is recomputed using (2.6). Otherwise, if $f(x_1) \geq f(x_2)$, then $a \leftarrow x_1$, $x_1 \leftarrow x_2$ and x_2 is recomputed using (2.7). At each iteration, the length of the interval reduces by $1/\phi$ and the algorithm terminates when the length of the interval is less than some tolerance. By [14, Chpt 5.4], golden section search has guaranteed linear convergence.

Brent's minimization method

Brent's minimization method [14, Chpt 5.4] uses both golden section search and successive parabolic interpolation to approximate a minimizer of a one-dimensional unimodal function $f : [a, b] \to \mathbb{R}$. Successive parabolic interpolation involves fitting a quadratic model from three points $(x_i, f(x_i))$, where $i = 0, 1, 2$. The point x_{i+1} is found by observing where the derivative of the fitted quadratic model is zero. A point x_i is replaced with x_{i+1} and the process is repeated until some stopping criterion is met.

According to [14, Chpt 5.1], a drawback of successive parabolic interpolation is that it can diverge or converge to a maximum or inflection point. Therefore, applying a combination of golden section search and successive parabolic interpolation ensures that the advantages of both methods are retained (see [14, Chpt 5.4] for more details).

Both golden section search and Brent's minimization method have been implemented within the SciPy library [134] in Python, and can be applied by using `scipy.optimize.minimize_scalar` with the appropriate method.

Algorithms for inexact line search and termination conditions

In practice, it is common to sacrifice some accuracy when computing $\gamma^{(k)}$ within line search methods to improve the overall efficiency of the local optimization procedure. Hence, the Armijo, Wolfe and Strong Wolfe conditions are proposed to allow termination of a line search method when a sufficient reduction in the objective function value is achieved. The following description of the Armijo, Wolfe and Strong Wolfe conditions is based on [94, Chpt. 3.1].

Consider the following Armijo condition

$$f(x^{(k)} - \gamma^{(k)} s^{(k)}) \leq f(x^{(k)}) - c_1 \gamma^{(k)} \nabla f(x^{(k)})^T s^{(k)}, \tag{2.8}$$

where $c \in (0, 1)$. The Armijo condition (2.8) ensures that a sufficient decrease in the objective function f is achieved. Furthermore, the Armijo condition can be used in conjunction with the following curvature condition to eliminate unacceptable small step sizes $\gamma^{(k)}$,

$$\nabla f(x^{(k)} - \gamma^{(k)} s^{(k)})^T s^{(k)} \leq c_2 \nabla f(x^{(k)})^T s^{(k)}, \tag{2.9}$$

where $c_2 \in (c_1, 1)$. Application of the Armijo and curvature conditions are known as the Wolfe conditions. It may be possible that a step length $\gamma^{(k)}$ which satisfies the Wolfe conditions is not close to a minimizer of (2.5). Hence, the curvature condition can be altered as follows to ensure that the selected $\gamma^{(k)}$ is in the vicinity of a local minimizer or stationary point of (2.5)

$$|\nabla f(x^{(k)} - \gamma s^{(k)})^T s^{(k)}| \leq c_2 |\nabla f(x^{(k)})^T s^{(k)}|. \tag{2.10}$$

The conditions (2.8) and (2.10) are known as the Strong Wolfe conditions.

The Armijo backtracking line search algorithm (see [94, Alg. 3.1] for more details) is a popular method to determine $\gamma^{(k)}$, and in general, involves repeatedly reducing the initial guess γ until the Armijo condition (2.8) is satisfied. The curvature condition (2.9) is omitted since the trial step lengths γ are suitably chosen by the backtracking method. A line search algorithm applying the Strong Wolfe conditions is provided in [94, Chpt. 3.5], which consists of a bracketing and selection phase. Furthermore, the corresponding algorithm in [94, Chpt. 3.5] has been implemented within the SciPy library [134] in Python and can be applied by using `scipy.optimize.line_search`.

2.3.3 Search direction $s^{(k)}$

The computation of the search direction $s^{(k)}$ will be discussed according to first-order, second-order and direct search methods. The motivation for various first and second-order methods to compute the search direction $s^{(k)}$ originates from Taylors theorem presented in Theorem 1 (based on [94, Thm. 2.1]). Direct search methods incorporate objective function values to determine the search direction $s^{(k)}$. Typically, direct search methods are employed if the gradient is unknown or computationally expensive to compute.

Theorem 1. *Assume $f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable and that $p \in \mathbb{R}^d$. Then*

$$f(x^{(k)} + p) = f(x^{(k)}) + p^T \nabla f(x^{(k)} + tp), \tag{2.11}$$

where $0 < t < 1$. In addition, if f is twice continuously differentiable then

$$f(x^{(k)} + p) = f(x^{(k)}) + p^T \nabla f(x^{(k)}) + \frac{1}{2} p^T \nabla^2 f(x^{(k)} + tp)p, \tag{2.12}$$

where $0 < t < 1$.

First order

Recall that Taylor's first-order approximation around $x^{(k)}$ involves constructing a linear approximation of the objective function f. The gradient $\nabla f(x^{(k)})$ provides the direction of the most significant change in the objective function f at point $x^{(k)}$. Hence, first-order search directions repeatedly utilize the gradient of the objective function $\nabla f(x^{(k)})$ to locate a local minimizer of an objective function f. Consider the following search direction for the recursive procedure (2.4),

$$s^{(k)} = \nabla f(x^{(k)}). \tag{2.13}$$

Steepest descent is an iterative method which involves constructing search directions $s^{(k)}$ of the form (2.13) and computing step lengths $\gamma^{(k)} > 0$ according to (2.5). Steepest descent ensures that a reduction in the objective function value is achieved at each iteration since

$$f(x^{(k)} - \gamma^{(k)} s^{(k)}) = f(x^{(k)}) - \gamma^{(k)} \nabla f(x^{(k)})^T \nabla f(x^{(k)}) < f(x^{(k)}),$$

where $\nabla f(x^{(k)})^T \nabla f(x^{(k)}) > 0$ and $\gamma^{(k)} > 0$ is selected according to (2.5). The magnitude of p in (2.11) must be small to employ Taylor's first-order approximation around $x^{(k)}$. Hence, the magnitude of $\gamma^{(k)} \nabla f(x^{(k)})$ must be small, which can be controlled by the step length $\gamma^{(k)}$. The advantage of using $s^{(k)} = \nabla f(x^{(k)})$ is that only first-order derivatives are required compared to higher order derivatives, which are more expensive to compute. However, convergence to a local minimizer can be very slow due to the zig-zag nature of steepest descent iterations.

There have been several variations of first-order methods to compute the search direction $s^{(k)}$, such as steepest descent with momentum [102] known as the heavy ball method, and Nesterov's accelerated gradient method [92]. The conjugate gradient method is a special case of the heavy ball method (see [101, Chpt. 3.2]). Consider the following non-linear conjugate gradient method to compute a search direction for the recursive procedure (2.4),

$$s^{(k)} = \nabla f(x^{(k)}) - \beta^{(k)} s^{(k-1)}, \tag{2.14}$$

where $\beta^{(k)}$ ensures that search directions $s^{(k)}$ and $s^{(k-1)}$ are conjugate. Initially, conjugate methods were used to solve linear systems of equations, which was expanded to non-linear functions in [33]. An overview of linear and non-linear conjugate gradient methods is presented in [46] and [94, Chpt. 5], along with proposed variations of $\beta^{(k)}$. The computational cost of performing non-linear conjugate gradient methods and steepest descent is similar since first-order derivatives are computed. Despite this, [94, Chpt. 2.2] states that non-linear conjugate gradient methods are more effective than steepest descent and improve efficiency. Non-linear conjugate gradient methods are not as efficient as second-order methods. However, they require far less storage capacity since matrices do not need to be stored for non-linear conjugate gradient methods [94, Chpt. 2.2].

Second order

Second-order methods utilize curvature information of the objective function f to determine a search direction. Consider the following second-order Taylor series,

$$f(x^{(k)} + p) \approx f(x^{(k)}) + p^T \nabla f(x^{(k)}) + \frac{1}{2} p^T \nabla^2 f(x^{(k)}) p, \tag{2.15}$$

where $p \in \mathbb{R}^d$. If $\nabla^2 f(x^{(k)})$ is positive definite, then (2.15) can be differentiated with respect to p and set to zero in order to obtain the Newton direction. That is, consider the following search direction for the iterative procedure (2.4),

$$s^{(k)} = \nabla f(x^{(k)})(\nabla^2 f(x^{(k)}))^{-1}. \tag{2.16}$$

Note that (2.12) is different to (2.15) since $\nabla^2 f(x^{(k)} + tp)$ is replaced with $\nabla^2 f(x^{(k)})$. By [94, Chpt. 2.2], if $\nabla^2 f(x^{(k)})$ is smooth enough and the magnitude $\|p\|$ is small, then (2.15) approximates (2.12) reasonably well. A natural step length of $\gamma^{(k)} = 1$ is typically used when $s^{(k)}$ is of the form (2.16). Although, the step length $\gamma^{(k)}$ may be updated to reduce the objective function value further.

By [94, Chpt. 2.2], the main advantage of the Newton direction is that convergence to a local minimizer can be very fast. However, computation of the Hessian $\nabla^2 f(x^{(k)})$ at each iteration k can be expensive and challenging. Consequently, Quasi-Newton search directions are often employed, which approximate $\nabla^2 f(x^{(k)})$ by a matrix $B^{(k)}$ that utilizes changes in the gradient to obtain knowledge on second-order derivatives [94, Chpt. 2.2]. Examples of methods to update $B^{(k)}$ are BFGS [15, 32, 43, 116] and Symmetric Rank One (SR1) [94, Sect. 6.2].

Direct search

Direct search methods can compute search directions $s^{(k)}$ based on objective function values. Many methods to compute $s^{(k)}$ utilize the objective function values at the current point $x^{(k)}$ with the unit vectors $\pm e_i$ $(i = 1, \dots, d)$, where e_i contains a 1 in the i-th position and zeros in all other positions.

The approach of finite difference methods is to evaluate the objective function with minor changes in the coordinates of $x^{(k)}$ and is discussed in [94, Chpt. 8.1]. Consider the following one-sided finite difference method

$$\frac{df}{dx} \approx \begin{bmatrix} \dfrac{f(x^{(k)} + c^{(k)} e_1) - f(x^{(k)})}{c^{(k)}} \\ \vdots \\ \dfrac{f(x^{(k)} + c^{(k)} e_d) - f(x^{(k)})}{c^{(k)}} \end{bmatrix} \tag{2.17}$$

where $c^{(k)}$ is some constant (see [94, Chpt. 8.1] for a discussion on the choice of $c^{(k)}$). Consider the following central finite difference method which can be applied to obtain a more accurate derivative estimate,

$$\frac{df}{dx} \approx \begin{bmatrix} \frac{f(x^{(k)} + c^{(k)}e_1) - f(x^{(k)} - c^{(k)}e_1)}{2c^{(k)}} \\ \vdots \\ \frac{f(x^{(k)} + c^{(k)}e_d) - f(x^{(k)} - c^{(k)}e_d)}{2c^{(k)}} \end{bmatrix}. \tag{2.18}$$

Even though the central finite difference method produces a more accurate gradient approximation, it is more computationally expensive than the one-sided forward difference method since $2d$ function evaluations are required instead of $d + 1$. The search direction $s^{(k)}$ can be set as (2.17) or (2.18) to approximate the gradient.

On the other hand, the unit vectors e_i $(i = 1, \ldots, d)$ can be used directly as the search direction $s^{(k)}$. In particular, coordinate search (see [94, Chpt. 9.3]) attempts to reduce the objective function value by searching along $s^{(k)} = \pm e_i$ $(i = 1, \ldots, d)$ at each coordinate $x_i^{(k)}$. Enhancements to the selection order of coordinates $i = 1, ..., d$ are discussed in [94, Chpt. 9.3].

Pattern search is described as a generalization of coordinate search in [94, Chpt. 9.3]. Furthermore, [94, Chpt. 9.3] states that pattern search involves selecting a set of search directions for $x^{(k)}$ and evaluating the objective function f at a predefined step length $\gamma^{(k)}$ along each search direction. If a significant reduction in the objective function value is achieved, then $x^{(k+1)}$ is obtained by moving along the corresponding search direction $s^{(k)}$ at the selected step length $\gamma^{(k)}$. Subsequently, the set of search directions and choice of step length may be updated for $x^{(k+1)}$, and the process is repeated.

2.4 Stochastic local optimization

2.4.1 Overview

Stochastic (simulation) local optimization algorithms are employed when the objective function values are stochastic or when there is some randomness in selecting the search direction [121]. Consider the following objective function

$$f(x) = \eta(x) + \epsilon, \tag{2.19}$$

where $\mathfrak{X}$ is some feasible domain, $x \in \mathfrak{X} \subset \mathbb{R}^d$, $\eta(x)$ is an unknown objective function and ϵ is the error. If $\eta(x)$ is assumed to be unimodal, then one local minimizer exists in the feasible domain $\mathfrak{X}$, which is also the global minimizer.

In [121], randomness in the search direction can reduce sensitivity to noisy objective function values and aid exploration in different regions of $\mathfrak{X}$. A vast amount of literature has been devoted to stochastic optimization [2, 19, 35, 120, 121, 129]. It is assumed that only noisy objective function values (2.19) are available to compute $\gamma^{(k)}$ and $s^{(k)}$ in (2.4) within this section. In [129], stochastic approximation and response surface methodology are categorized as stochastic local optimization methods which use objective function values (2.19) and will be discussed in this section. In particular, special attention will be given to the choice of search direction $s^{(k)}$ used for the iterative update (2.4) within each method.

2.4.2 Stochastic approximation (SA)

SA was introduced in [61, 108] to iteratively reduce the stochastic objective function (2.19) through estimation of the gradient $\widehat{\nabla}\eta(x^{(k)})$. Analogous to steepest descent, the search direction in (2.4) is

$$s^{(k)} = \widehat{\nabla}\eta(x^{(k)}).$$

In [108], unbiased stochastic gradient estimates are used to construct $\widehat{\nabla}\eta(x^{(k)})$. Alternatively, finite difference estimates are applied in [61] to compute $\widehat{\nabla}\eta(x^{(k)})$. Discussions on SA methods which utilize unbiased stochastic gradient estimates to approximate the gradient are omitted in this subsection, but will be discussed in Subsection 2.6.4 (see [120, Chpt. 4-5] for overview and [19, Sect. 3] for recent developments of SA methods that use unbiased stochastic gradient estimates).

Suppose that only noisy objective function values are available. The following SA methods are proposed to estimate $\widehat{\nabla}\eta(x^{(k)})$. Finite difference methods can be applied to estimate the gradient of an objective function at some point $x^{(k)}$, and have been outlined in (2.17) and (2.18) for computing the search direction $s^{(k)}$ within (2.4) for deterministic local optimization methods. Forward difference stochastic approximation (FDSA) is discussed in [120, Chpt. 6] and [121], along with the selection of step length $\gamma^{(k)}$ and parameter $c^{(k)}$ in (2.17) and (2.18). The computation of (2.18) requires $2d$ function evaluations and consequently, FDSA will be computationally expensive to compute for large d. Consider the following simultaneous perturbation stochastic approximation (SPSA) method outlined in [120, Chpt. 7]

$$\widehat{\nabla}\eta(x^{(k)}) = \begin{bmatrix} \dfrac{f(x^{(k)} + c^{(k)}\xi^{(k)}) - f(x^{(k)} - c^{(k)}\xi^{(k)})}{2c^{(k)}\xi_1^{(k)}} \\ \vdots \\ \dfrac{f(x^{(k)} + c^{(k)}\xi^{(k)}) - f(x^{(k)} - c^{(k)}\xi^{(k)})}{2c^{(k)}\xi_d^{(k)}} \end{bmatrix} \tag{2.20}$$

where $\xi^{(k)} = (\xi_1^{(k)}, \ldots, \xi_d^{(k)})$ is a mean zero random vector. The selection of $\gamma^{(k)}$, $c^{(k)}$ and $\xi^{(k)}$ are discussed in detail within [120, Chpt. 7] and [121]. It can be observed in (2.20) that $\xi^{(k)}$ incorporates additional randomness in the estimate of the gradient, allowing the opportunity to explore different regions of the feasible domain $\mathfrak{X}$. Furthermore, the computation of SPSA in (2.20) requires just two objective function evaluations at each iteration k, irrespective of the size of dimension d. According to [120, Chpt. 7] and [121], the statistical accuracy is similar for SPSA and FDSA with a specified amount of iterations, although only $1/d$ function evaluations required by FDSA are used by SPSA.

2.4.3 Response surface methodology (RSM)

RSM is a collection of methods for approximating a minimizer of a regression function using a series of observations containing errors. The most often used and cited RSM strategy is the Box-Wilson (BW) algorithm (see [13] and [51]). The BW algorithm consists of two phases. The first phase of BW consists of iteratively applying the recursion (2.4) until a subregion of the optimum is reached. At each iteration, a first-order model whose coefficients are estimated using least-squares are used to determine the search direction $s^{(k)}$ in (2.4). Different options are available for choosing the step-length $\gamma^{(k)}$ in (2.4) (see [34, 83, 86, 89]). The second phase of BW involves fitting a locally quadratic model of the response function for estimating the location of the minimizer.

Informative overviews of the BW algorithm and RSM, in general, can be found in [84, Chapter 11] and [89]. In addition, surveys outlining the development and progress of RSM are provided in [59, 87, 88].

Different modifications of the BW algorithm are distinguished by using different designs on Phase I and different rules for choosing the step length $\gamma^{(k)}$ of the descents. The choice of search direction $s^{(k)}$ for the first phase of BW has not been challenged in the RSM-related literature, and is routinely suggested as the least-squares estimator of the gradient of the response function at a current point. However, this can restrict the use of BW since a significant computational cost is associated with performing the least-squares estimator when the number of observations and dimension of a design matrix is large. In addition, [38, 39, 141] shows that the least-squares estimator does not provide the optimal direction. Consequently, Chapter 4 presents an alternative search direction $s^{(k)}$, which can significantly improve the first phase of BW and RSM in general for high-dimensional problems.

2.5 High-dimensional geometry

2.5.1 Overview

This section is based on the monographs [10, 140] and aims to illustrate the counter-intuitive properties of high-dimensional geometry. Specifically, the behaviour of many high-dimensional objects and distributions contradicts the known behaviour in 2-dimensions. This section considers the behaviour of sampled points drawn uniformly from a high-dimensional unit cube and the high-dimensional Gaussian distribution. The purpose of discussing the behaviour of points drawn uniformly from a high-dimensional unit cube and Gaussian distribution is to portray the challenges encountered by global optimization problems when the dimension is large. Particularly with the exploration of the feasible domain during the global phase of optimization algorithms.

2.5.2 Properties of a high-dimensional unit cube

The results in this subsection are from [10, Sect. 2.3] and [140, Sect. 1.1].

Volume of the unit cube

Suppose $\mathfrak{X} \in \mathbb{R}^d$ is a high-dimensional object. The following quantity is obtained by shrinking $\mathfrak{X}$ by a small constant $\epsilon > 0$,

$$(1-\epsilon)\mathfrak{X} = \{(1-\epsilon)\mathfrak{X} | x \in \mathfrak{X}\}.$$

Consider the following proportion [10, Sect. 2.3],

$$\frac{\text{Vol}((1-\epsilon)\mathfrak{X})}{\text{Vol}(\mathfrak{X})} = (1-\epsilon)^d \leq e^{-\epsilon d}. \tag{2.21}$$

If ϵ is fixed, then (2.21) approaches zero as d tends to infinity. Consequently, almost all the volume of $\mathfrak{X}$ is contained in $\mathfrak{X} \setminus (1-\epsilon)\mathfrak{X}$. Hence, the volume of a high-dimensional object (e.g. high-dimensional unit cube) is near the boundary.

Mass in the unit cube

Consider the following result in [140, Sect. 1.1.2], derived using the Hoeffding inequality, where $\epsilon > 0$, variables $x_1, \dots, x_d$ are sampled uniformly at random from $[0, 1]$, and $(x_1 - 1/2)^2, \dots, (x_d - 1/2)^2$ are independent random variables on $[0, 1/4]$ with mean $1/12$,

$$\Pr\left\{\left|(x_1 - 1/2)^2 + ... + (x_d - 1/2)^2 - \frac{d}{12}\right| \geq \epsilon d\right\} \leq 2e^{-8d\epsilon^2}.$$

The result suggests that the annulus surrounding the sphere, with radius $\sqrt{d/12}$ and centre point $(1/2, \dots, 1/2)^T$, contains almost all of the mass of a high-dimensional cube $[0,1]^d$. Consequently, hardly any mass will be near the 2^d vertices of the cube [140, Sect. 1.1.2].

Distance between two random points drawn uniformly from the unit cube

Suppose $x = (x_1, \dots, x_d)^T$ and $y = (y_1, \dots, y_d)^T$ are two points sampled uniformly at random from $[0,1]^d$. The squared distance between x and y is,

$$\|x - y\|^2 = \sum_{i=1}^{d} (x_i - y_i)^2. \tag{2.22}$$

Furthermore, $\mathbb{E}\,\|x - y\|^2 = d/6$ and $\text{Var}\|x - y\|^2 = 7d/180$. Hence, the distance between points drawn uniformly at random from $[0,1]^d$ grows with increasing d, suggesting that points will be far away from one another in high dimensions.

Numerical investigation of properties

Suppose points $x_n = (x_{n_1}, \dots, x_{n_d})^T$ and $y_n = (y_{n_1}, \dots, y_{n_d})^T$ are sampled uniformly at random from $[0,1]^d$ for different dimensions d, where $n = 1, \dots, 100000$. Points x_n and y_n will be used to illustrate some of the discussed properties of a high-dimensional unit cube. Figure 2.3 shows boxplots of the distances $\|x_n - y_n\|$ for different d. It can be observed in Figure 2.3 that the distances $\|x_n - y_n\| \approx \sqrt{d/6}$. Therefore, points x_n and y_n will be far away from one another when the dimension is large.

A point x_n is classified as being near an edge or boundary of the cube $[0,1]^d$ if at least one coordinate x_{n_i} $(i = 1, \dots, d)$ is greater than 0.95 or less than 0.05. Furthermore, a point x_n is classified as being near a vertex if all coordinates are either greater than 0.95 or less than 0.05. This is also the case for each point y_n. For example, $x_1 = (0.5, 0.6, 0.97, 0.3)^T$ is a point near a boundary of $[0,1]^4$ and $x_2 = (0.98, 0.01, 0.02, 0.99)^T$ is a point near a vertex of $[0,1]^4$. Table 2.2 shows the total percentage of points x_n and y_n that are near a boundary or near a vertex of the cube $[0,1]^d$ for various dimensions d.

Table 2.2 shows the number of points near a boundary of the cube $[0,1]^d$ grows as a function of d. In fact, all sampled points x_n and y_n are near a boundary of the cube $[0,1]^d$ when $d \geq 200$. However, Table 2.2 illustrates that no sampled points x_n and y_n are near any of the 2^d vertices of the cube $[0,1]^d$ when $d \geq 10$.

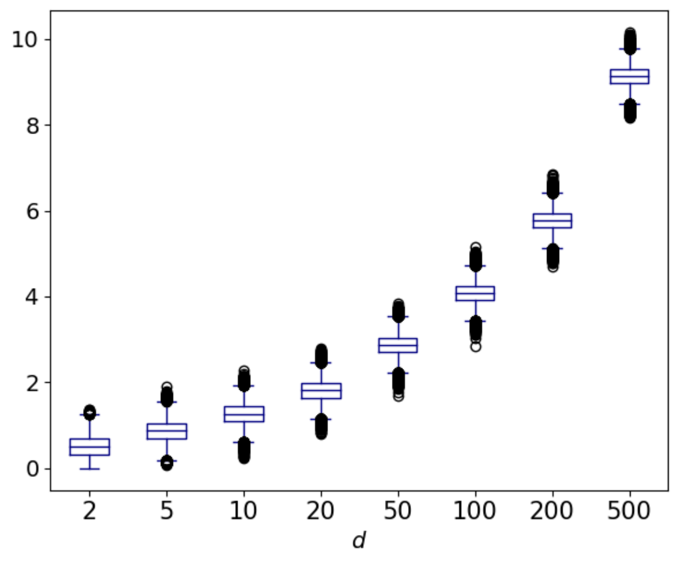


Figure 2.3: Boxplots of the distances $\|x_n - y_n\|$ with $n = 1, \dots, 100000$, where $x_n = (x_{n_1}, \dots, x_{n_d})^T$ and $y_n = (y_{n_1}, \dots, y_{n_d})^T$ are sampled uniformly at random from $\mathfrak{X} = [0,1]^d$.

	d							
	2	5	10	20	50	100	200	500
Near a boundary	18.932	40.955	65.159	87.79	99.451	99.998	100	100
Near a vertex	1.008	0.001	0	0	0	0	0	0

Table 2.2: Total percentage (%) of uniformly sampled points $x_n = (x_{n_1}, \dots, x_{n_d})^T$ and $y_n = (y_{n_1}, \dots, y_{n_d})^T$ from $[0,1]^d$ that are near a boundary or near a vertex of the cube $[0,1]^d$ for various d and $n = 1, \dots, 100000$.

2.5.3 Properties of the Gaussian distribution in high dimensions

Suppose a point $x = (x_1, \dots, x_d)^T$ is sampled from $N(0, I_d)$. Consider,

$$\mathbb{E}\,\|x\|^2 = d\,\mathbb{E}\,x_1^2 = d\mathrm{Var}(x_1^2) = d.$$

Therefore, $\|x\| \approx \sqrt{d}$ by the Central Limit Theorem when d is large. Furthermore, suppose another point $y = (y_1, \dots, y_d)^T$ is sampled from $N(0, I_d)$. In a similar fashion to the above, consider,

$$\mathbb{E}\,\|x - y\| = d\,\mathbb{E}(x_1 - y_1)^2 = d(\mathbb{E}\,x_1^2 + \mathbb{E}\,y_1^2 - 2\,\mathbb{E}\,x_1 y_1) = d(\mathrm{Var}(x_1) + \mathrm{Var}(y_1)) = 2d.$$

Suppose d is large, then points x and y are approximately orthogonal by Pythagoras' theorem since $\|x\|^2 \approx d$, $\|y\|^2 \approx d$ and $\|x - y\|^2 \approx 2d$ [10, Sect. 2.2]. The Gaussian

Annulus Theorem in [10, Sect. 2.6] shows that for any $\beta \leq \sqrt{d}$ and positive constant c, at least $1 - 3e^{-c\beta^2}$ of the probability mass lies within an annulus of $\sqrt{d} - \beta \leq \|x\| \leq \sqrt{d} + \beta$. Hence, mostly all of the probability is concentrated within a thin annulus around the sphere of radius $\sqrt{d}$, which contradicts the known behaviour of a 2-dimensional Gaussian distribution.

2.6 Machine learning and optimization

2.6.1 Overview

Machine learning consists of learning a model from observed input data to predict an output or trend for unseen data. Machine learning algorithms are widely used for speech and image recognition, recommendation systems, text classification, market research and data visualization. The recent success of many machine learning algorithms is due to the improved computational resources to solve increasingly challenging problems and the availability of big data.

Most machine learning algorithms can be categorized as either supervised or unsupervised learning [44, 48]. Supervised learning consists of learning a prediction model to approximate the relationship between observed inputs and outputs. In addition, the learnt prediction model must also be able to predict accurate outputs for unseen inputs. Supervised learning can be categorized as a regression or classification problem depending on whether the observed output is continuous or discrete. Algorithms applied to solve supervised learning problems include logistic regression, neural networks and support vector machines (SVM). For unsupervised learning, the values or labels of output variables are unknown, and the objective consists of understanding the attributes and features of input variables to determine trends. Clustering and dimension reduction are examples of unsupervised learning problems and can be solved by K-means clustering and principal component analysis (PCA), respectively. Typically, a loss function is used to check the accuracy of the predictions in supervised learning. Since outputs are unknown within unsupervised learning, it can be challenging to determine whether the predicted trends are an accurate representation.

This section focuses on the formulation of optimization problems within machine learning and the methods applied to solve the optimization problems. Similar to Section 2.3, gradient-based optimization methods utilized within machine learning are of interest. Therefore, the remainder of this section will focus on supervised learning models with continuous variables since the derivative of the loss function may be computed.

2.6.2 Relationship between optimization and machine learning

Optimization is a vital part of many machine learning algorithms, and a great deal of recent literature is devoted to the relationship between optimization and machine learning [6, 11, 25, 37, 119, 124].

The progression of many machine learning algorithms involves solving increasingly challenging optimization problems, which has inspired a great deal of interest in various optimization methods. The development of advanced optimization methods has influenced the advancements made in machine learning. On the other hand, machine learning algorithms have also been utilized to aid optimization. For example, [5] provides a survey of applications that use machine learning to solve combinatorial optimization problems. Also, [16] uses machine learning to determine when to apply local search within a global optimization algorithm in order to improve efficiency. In addition, [135] provides a review of machine learning applications for the optimization of product quality and production processes.

The application of optimization within machine learning will be explored throughout this section. Areas of machine learning where optimization is utilized are outlined in [119, Sect. 3] and are summarized in Table 2.3.

2.6.3 Formulation of the model training optimization problem for supervised learning

A fundamental part of machine learning is model training [119]. Typically, the goal of optimization is to minimize the objective function's value with respect to the objective function's parameters. However, the aims of optimization in model training are twofold. Firstly, the prediction model should have a low training error, and secondly, be capable of generalizing to unseen data. Hence, optimality may be reduced to improve generalization. The model training optimization problem is formulated for many machine learning algorithms in [37, 124]. The types of optimization problems within machine learning vary from convex optimization problems (e.g. logistic regression and SVM) to non-convex and non-linear optimization problems (e.g. deep neural networks) [11].

Consider the following discussion on the general model training optimization problem for supervised learning, which is based on [11, Sect. 3.1]. Suppose that $x \in \mathbb{R}^d$ is some input variable and $y \in \mathbb{R}$ is the corresponding output variable. It is desirable to generate a prediction model to predict the correct output y given an

Area	**Description**
Data preprocessing [119, Sect. 3.1]	Various data preprocessing methods are utilized within machine learning to ensure data is suitable for model learning. For example, methods such as dimension reduction, instance reduction and data balancing may be applied (see [119, Sect. 3.1] for a description of methods), which can be formulated as optimization problems.
Algorithm selection [119, Sect. 3.2]	Selecting a suitable machine learning algorithm for a specified problem can be difficult. Hence, optimization techniques can be applied to determine the most suitable machine learning algorithm from a candidate set of algorithms for a specified problem.
Hyper-parameter tuning [119, Sect. 3.3]	Hyper-parameters are properties related to a machine learning algorithm that must be tuned based on the specified problem to improve model training. Typically, hyper-parameters are selected by an expert. Consequently, the most promising set of hyper-parameters for a specified problem may not be chosen. The selection of hyper-parameters can be formulated as a black-box optimization problem. The generalization error of the prediction model with each set of hyper-parameters is approximated using the validation dataset, which is a subset of the training dataset.
Model training [119, Sect. 3.4]	For many machine learning algorithms, the parameters for the prediction model can be optimized by minimizing a loss function based on the learning problem. Furthermore, the prediction model with the selected parameters must generalize to unseen data.

Table 2.3: Areas of machine learning where optimization is utilized, as stated in [119, Sect. 3].

input x. The prediction model can be derived from a family of functions

$$\mathcal{H} = \{h(.,w)\}, \tag{2.23}$$

where $w \in \mathbb{R}^d$ is the parameter vector that is optimized during the training process. In order to measure the accuracy of each proposed prediction model, the loss function

$l(h(x,w),y)$ is computed, where $h(x,w)$ is the predicted output and y is the true output. Suppose that $P(x,y)$ is the joint probability distribution that represents the relationship between all possible inputs x and outputs y. Then the following optimization problem is formulated

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \mathbb{E}[l(h(x,w),y)], \tag{2.24}$$

which is known as the expected risk. In general, the probability distribution $P(x,y)$ is not generally known. However, a training set with N independent and identically distributed samples (x_n, y_n) with $n = 1, ..., N$ can be selected. Therefore, the following empirical risk can be minimized with respect to w

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \frac{1}{N}\sum_{n=1}^{N} l(h(x_n,w),y_n). \tag{2.25}$$

It is desirable to train the prediction model, which minimizes both (2.24) and (2.25). Furthermore, it is also desirable for the prediction model to accurately predict outputs for unseen inputs, known as generalization. This can be checked by applying the selected prediction model on a test dataset that consists of unseen inputs and outputs.

If the training error is minimal and the testing error is relatively large, then it is likely that the prediction model over-fits the training dataset. That is, the prediction model memorizes the training dataset in order to make predictions. On the other hand, the prediction model under-fits the training dataset if the training and testing error are both relatively large. By [44, Chpt. 5.2], a machine learning algorithm performs best when the prediction model is appropriate for the true complexity of the problem and the amount of training data provided. The balance between over-fitting and under-fitting can be controlled by amending the models capacity [44, Chpt. 5.2]. Methods to control the capacity include choosing an appropriate family of functions (2.23) and applying regularization in (2.25) to simplify the model.

The most common form of regularization involves adding a penalty term in (2.25) during optimization. That is,

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \frac{1}{N}\sum_{n=1}^{N} l(h(x_n,w),y_n) + \lambda g(w),$$

where λ is some regularization constant and g is some convex regularization function. Possibilities for g include the L^1 norm $g(w) = \|w\|_1$ or L^2 norm $g(w) = \|w\|_2$. Including a regularization function ensures that entries of the parameter vector w do not include extreme values. If $g(w) = \|w\|_1$ is applied, entries w_i $(i = 1, ..., d)$ which are not influential in predicting an output may be set to zero, which can be viewed as feature selection.

Suppose a selection of samples are removed from the training dataset to construct a validation dataset. Another form of regularization is early stopping of model training (see [44, Sect. 7.8]), where the focus is on the validation error produced by the prediction model as opposed to the training error. Specifically, the validation error is evaluated repeatedly with updated parameters of the prediction model, obtained after applying a particular number of iterations of a training algorithm. If the validation error is reduced, parameters of the prediction model are stored since they provide the lowest validation error observed so far. Otherwise, if the validation error is not reduced, the model parameters are not stored. The process is repeated until no further improvement in the validation error is observed with updated parameters of the prediction model.

Hyper-parameters are required by a family of functions and also for regularization. Consequently, hyper-parameters must also be optimized in addition to the parameters of the prediction model. As discussed in Table 2.3, a selection of samples can be removed from the training dataset to construct a validation dataset. Then for each set of chosen hyper-parameters, a prediction model is learnt using the training dataset, and the generalization error can be computed using the validation dataset. The prediction model that produces the smallest generalization error on the validation dataset, among all sets of chosen hyper-parameters, is selected as the prediction model.

2.6.4 Optimization methods

During model training, optimization methods are applied to minimize the loss function with respect to the parameters of the prediction model. The following list of desirable properties of an optimization method in machine learning are outlined in [6].

- *Good generalization,*
- *Scalability to large problems,*
- *Reasonable execution times and memory requirements,*
- *Algorithm is simple to implement,*
- *Exploit problem structure,*
- *Fast convergence to an approximate solution,*
- *Numerical stability for class of machine learning models selected,*
- *Theoretical properties of convergence and complexity.*

Optimization methods applied for model training are outlined in [11, 25, 110, 124]. An overview of first-order optimization methods will be discussed in this subsection, namely, batch gradient descent (BGD) and stochastic gradient descent (SGD). Steepest descent is presented in Section 2.3, which is an instance of gradient descent since the step length $\gamma^{(k)}$ is chosen to produce the smallest objective function value along the search direction $-\nabla f(x^{(k)})$. Optimization methods for machine learning use a learning rate $\eta^{(k)}$ that can be fixed or adapted at each iteration.

Throughout this subsection, the focus will be on the optimization problem (2.25). The notation is simplified as follows to discuss various optimization methods. Suppose $f_n(w) = l(h(x_n, w), y_n)$, then (2.25) reduces to

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} f_n(w).$$

Hence, the k-th iteration of BGD is as follows

$$\begin{aligned} w^{(k+1)} &= w^{(k)} - \frac{\eta^{(k)}}{N} \sum_{n=1}^{N} \nabla f_n(w^{(k)}) \\ &= w^{(k)} - \eta^{(k)} \nabla f(w^{(k)}). \end{aligned} \tag{2.26}$$

BGD is intuitive and straightforward to implement. However, it can be observed in (2.26) that computation of the gradient uses all samples of the training dataset. Consequently, deriving the gradient is computationally expensive and results in slow convergence, particularly when N and d are large. By [124], the computational complexity of performing an iteration of BGD is $\mathcal{O}(Nd)$. Therefore, SGD was proposed in [108] to reduce the computational cost associated with BGD.

Iterative methods that use direct noisy gradients have been investigated in [108]. As a result, the field of stochastic approximation was introduced (see [120, Chpt. 4-7]). SGD computes an approximation of the true gradient in (2.26) by computing the gradient for one sample. Specifically, the k-th iteration of SGD is outlined as follows,

$$w^{(k+1)} = w^{(k)} - \eta^{(k)} \nabla f_n(w^{(k)}), \tag{2.27}$$

where $n \in \{1, ..., N\}$ is chosen uniformly at random. SGD is widely used to solve optimization problems in machine learning, especially when the number of samples N is extremely large. The approximate gradient $\nabla f_n(w^{(k)})$ in (2.27) is an unbiased estimator of the true gradient $\nabla f(w^{(k)})$ in (2.26). The computational cost of SGD at each iteration is significantly less than BGD since a single sample is required to compute $\nabla f_n(w^{(k)})$. In fact, the computational complexity of performing an iteration of SGD is $\mathcal{O}(d)$ [124]. However, many more iterations of SGD may need to be performed compared to BGD since an approximation of the gradient is derived from a single sample. Even with additional SGD iterations, the computational cost of SGD is still significantly less than BGD.

The variance of gradients produced in SGD is large since they may not necessarily be descent directions. Hence, the objective function value at $w^{(k)}$ may fluctuate rather than monotonically decrease with increasing k. To reduce the variance of the gradients in SGD, several samples may be randomly selected to approximate the gradient. This approach is known as mini-batch stochastic gradient descent. That is,

$$w^{(k+1)} = w^{(k)} - \frac{\eta^{(k)}}{|B^{(k)}|} \sum_{n \in B^{(k)}} \nabla f_n(w^{(k)}), \tag{2.28}$$

where $B^{(k)} \subset \{1, 2, ..., N\}$ is a subset of samples selected uniformly at random at the k-th iteration. From herein, mini-batch stochastic gradient descent will be referred to as SGD. The fluctuation of objective function values at $w^{(k)}$ for increasing k will lessen if the gradient is approximated with more samples. However, some level of fluctuation can still exist. This is beneficial since the literature [110, 124] states that the fluctuations caused by the approximated gradients in SGD can encourage exploration of the feasible domain, which can aid the discovery of better local minima. This is not the case for BGD since repeatedly applying (2.26) will find one local minimum. Nevertheless, fluctuations in SGD can slow down the convergence of finding a local minimum [124].

Challenges are often encountered when applying SGD, which are discussed in [110] and summarized as follows. Firstly, selecting the learning rate $\eta^{(k)}$ can be challenging. Specifically, it is difficult to determine how to adjust the learning rate during the training process and apply it with sparse data. Secondly, it may be possible to be trapped within a suboptimal local minimum, which is a difficulty often faced when optimizing a highly non-convex objective function (i.e. deep neural networks). Even finding local minima of a non-convex function may be difficult in some cases since saddle points may be present. Consequently, SGD may find a saddle point instead of a local minimizer [26].

The following enhancements are proposed to improve the performance of SGD (see [11, 25, 110, 124] for reviews of enhancements). Methods such as momentum [102] and Nesterov's accelerated gradient [92] with appropriate random initialization and a slowly increasing schedule for the momentum parameter have been applied with SGD in [126] to improve the performance. In addition, adaptive learning rate algorithms have been proposed to address the challenge of selecting the learning rate in SGD. AdaGrad is proposed in [28], which adjusts the learning rate for each parameter $w_i^{(k)}$ $(i = 1, ..., d)$ based on previous gradient information. A key drawback of AdaGrad is that the computation of the learning rate involves diving by the square root of the sum of squares of the previous gradients with respect to $w_i^{(k)}$. Therefore, the denominator will rapidly increase if there are many iterations of SGD, and as a result, the learning rate may tend to zero. Consequently, AdaDelta [138] and RMSprop (unpublished method presented in [52]), were proposed to address the problem of the learning rate tending to zero after many iterations. Another adaptive learning algorithm called Adam was proposed in [62], which incorporates momentum methods. It can be observed in [123] that adaptive learning rate algorithms can escape saddle points faster than SGD, which leads to faster convergence. Another proposed enhancement of SGD is to reduce the variance of approximated gradients since large variation can result in slow convergence. Consequently, several variance reduction methods, including SAG [109], SAGA [27] and SVRG [56] are proposed to improve the convergence of SGD.

2.7 Summary

A vast amount of literature is presented in this chapter to provide context for the research contributions presented in Chapters 3 and 4. That is, enhanced algorithms to efficiently and accurately solve high-dimensional black-box optimization problems, where the underlying objective function is either assumed to be deterministic or stochastic.

The global optimization problem is presented in this chapter, along with the approaches adopted by many global optimization algorithms to solve the corresponding problem. Many global optimization algorithms consist of a global and local search. The efficiency of global optimization algorithms often depends on the trade-off between global and local search.

The global search involves exploring and sampling points scattered over the feasible domain. Recall from Section 2.2 that the GRS class of stochastic global optimization algorithms incorporate random decisions to select points from the feasible domain within the global search. However, the behaviour of many high-dimensional objects and distributions contradicts the known behaviour in 2-dimensions. Consequently, the behaviour of sampling points from the feasible domain during the global search of optimization algorithms is counter-intuitive.

The local search involves making iterative local improvements (2.4) to the sampled points, where the step length $\gamma^{(k)}$ and search direction $s^{(k)}$ are updated according to some local optimization method. If exact information on objective function values and gradients is available, deterministic local optimization methods are applied for local search. Otherwise, stochastic local optimization methods are applied if objective function values contain errors.

Suppose that the black-box objective function is deterministic, and applying a local optimization method to find a local minimizer is computationally expensive since many iterations are required. In that case, it is desirable to apply the local optimization method with only promising points, which may lead to the discovery of a new local or global minimum. This is because applying the local optimization method to all sampled points will repeatedly find the same local minima, which will reduce efficiency. Techniques to identify promising points may include exploiting the information already gained and applying very few iterations of a local optimization method to each sampled point. As a result, the efficiency of many global optimization algorithms can be significantly improved, which is the focus of Chapter 3. Specifically, Chapter 3 presents an efficient version of multistart, which is part of the GRS class of stochastic global optimization algorithms discussed in Section 2.2.

On the other hand, suppose the black-box objective function is stochastic for high-dimensional optimization problems. Thus, ensuring local optimization methods are reliable and accurate when only function values containing errors are available to determine search directions $s^{(k)}$ is a key area of research, and the focus of Chapter 4. In Section 2.4, RSM is a stochastic local optimization algorithm and the most often used RSM strategy is the BW algorithm, which consists of two phases. The first phase of BW consists of iteratively applying the recursion (2.4) until a subregion of the optimum is reached. The second phase of BW involves fitting a locally quadratic model of the response function to estimate the location of the minimizer. Chapter 4 demonstrates that a different choice of search direction $s^{(k)}$ for the iterative update (2.4) within the first phase of BW can expand the application of BW and RSM, in general, for high-dimensional stochastic optimization problems.

Optimization is a vital part of machine learning. Specifically, the progression of many machine learning algorithms involves solving increasingly challenging optimization problems, which has inspired a great deal of interest in various optimization methods. The development of advanced optimization methods has influenced the progress made in machine learning. Hence, the research contributions presented in Chapters 3 and 4 may be applied to enhance optimization methods utilized in various areas of machine learning.

Chapter 3

Multistart with Early Termination of Descents

This chapter is summarized as follows.

- Section 3.1 provides an overview of the multistart algorithm with a variety of applications. In addition, a brief overview of multistart with early termination of descents (METOD) is provided, which can significantly improve the efficiency of multistart.
- Section 3.2 outlines variations of multistart within the literature that reduce the number of local searches which lead to the same local minimizer.
- Section 3.3 details the various objective functions that are applied with the METOD algorithm for analysis and numerical examples.
- In Section 3.4, a detailed overview of the METOD algorithm is provided.
- Section 3.5 provides a numerical study on the main inequality used by METOD to terminate local searches early for objective functions in Section 3.3.
- Section 3.6 presents the investigation of a key METOD algorithm parameter, along with recommended values for the parameter.
- Section 3.7 provides results of numerical examples applying the METOD algorithm with objective functions in Section 3.3.
- In Section 3.8, the accuracy and efficiency of METOD is investigated for high dimensions.
- Section 3.9 details the implementation of the METOD algorithm in Python. In addition, a summary of input and output parameters of the METOD algorithm are discussed, along with an example of application.

Chapter 5 illustrates the measures taken to ensure that the software developed for the METOD algorithm is accurate and reliable. Furthermore, the METOD algorithm has been implemented in Python and made publicly available on GitHub. Hence, all source code, tests, analysis results, and examples applying the METOD algorithm are readily available. In Sections 3.5 - 3.8, analysis of the METOD algorithm is conducted and all results can be found at https://github.com/Megscammell/METOD-Algorithm/tree/master/Numerical_Experiments. On the other hand, Section 3.9 focuses on the selection of METOD algorithm parameters to enhance adaptability and performance for a range of global optimization problems.

3.1 Introduction

Suppose a solution to the following unconstrained global optimization problem is required

$$\min_{x \in \mathfrak{X}} f(x), \tag{3.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a multimodal, continuous and differentiable objective function defined on the feasible domain $\mathfrak{X}$. Since f is multimodal, many local minimizers x_l^* ($l = 1, 2, ...$) may exist. A local minimizer x_l^* exists if

$$f(x_l^*) < f(x), \tag{3.2}$$

for all $x \in A_l \subset \mathfrak{X}$, where A_l is a region of attraction of the local minimizer x_l^*. Each time local search is applied with $x \in A_l$, the local minimizer x_l^* associated with region of attraction A_l will be found. The local minimizer x_l^* is also the global minimizer x^* if (3.2) holds for all $x \in \mathfrak{X}$.

Multistart is a celebrated stochastic global optimization method for solving global optimization problems of the form (3.1). Specifically, multistart is part of the global random search (GRS) class of stochastic global optimization methods, discussed within Section 2.2. Multistart consists of a global and local phase. The global phase involves sampling points from the feasible domain $\mathfrak{X}$ (e.g. uniformly at random) to allow diverse exploration, which may lead to new regions of attraction being discovered. A local search procedure is applied to each sampled point to find a local minimizer x_l^*.

Multistart has been applied to a variety of different global optimization problems. In [81], a survey of multistart methods for combinatorial optimization is provided. The survey focuses on memory and memoryless based multistart methods. The memory based methods exploit previous information for successive solutions, and memoryless based methods generate new solutions without knowledge of prior solutions. For both categories, an extensive review of various algorithms and applications has been provided. A randomized greedy multistart algorithm is proposed in [76] for the minimum common integer partition problem, which has several applications in molecular biology. That is, randomness is injected into the algorithm for several iterations to allow diversification and exploration of possible solutions in the early stages. After a certain number of iterations of the algorithm, a deterministic approach is applied to search within promising regions identified at the early stages of the algorithm. A neural network to classify items for a multi-criteria ABC inventory classification problem is proposed in [75]. A minimum number of neurons should be included in the hidden layer to reduce the risk of overfitting to the training set and, consequently, reducing the capability to generalize to new training samples. Therefore, it is proposed in [75] to use a randomized greedy multistart algorithm to add neurons to the hidden layer to minimize the size of the neural network. Further applications include a multistart algorithm for signal adaptive subband systems [128] and also multistart methods for quantum approximate optimization [117].

Although multistart is intuitive and simple to implement, the efficiency of multistart can be very poor since the same local minimizers may be found after applying local search to each sampled point. Many algorithms attempt to reduce the number of local searches executed within multistart to improve efficiency and to avoid a large concentration of sample points at discovered local minimizers.

The multistart with early termination of descents (METOD) algorithm, introduced in [144], aims to reduce the number of local searches to the same local minimizers. For objective functions with locally quadratic behaviour close to the neighbourhoods of local minimizers, the METOD algorithm can reduce the number of repeated local searches to the same local minimizer by applying a particular inequality. As a result, the efficiency of multistart can be significantly improved upon, especially when the dimension of the problem is very large (i.e. $d = 100$). The application of METOD is suitable for global optimization problems where the objective function f is considered black-box. In addition, the following assumptions of f are discussed in [144, Sect. 1]. Firstly, the objective function f has the form (3.1), and computation of function evaluations and derivatives are not expensive. Secondly, the feasible domain $\mathfrak{X}$ has a simple structure, and the total number of local minimizers of f is not large. Finally, the volume of the region of attraction of the global minimizer is not small.

Consider the following notation that will be used throughout this chapter.

- $\mathfrak{X} \subset \mathbb{R}^d$, the feasible domain;

- N, the total number of starting points;
- $X = \{x_1, x_2, \ldots\}$, sequence of points in $\mathfrak{X}$;
- $x_n^{(0)} = x_n = (x_{n_1}, x_{n_2}, ..., x_{n_d})^T \in X$ $(n = 1, 2, \ldots)$, point chosen from set X;
- steepest descent iteration:

$$x_n^{(k+1)} = x_n^{(k)} - \gamma_n^{(k)} \nabla f(x_n^{(k)}). \tag{3.3}$$

  Throughout this chapter, the local search procedure used by METOD is steepest descent and $\gamma_n^{(k)}$ is found using Brent's minimization method [14, Chpt 5.4], where $\gamma_n^{(k)}$ is an approximation of $\underset{\gamma>0}{\operatorname{argmin}} f(x_n^k - \gamma \nabla f(x_n^{(k)}))$. Iterations of steepest descent (3.3) are applied until the smallest $K_n = k$ is found such that,

$$\|\nabla f(x_n^{(k)})\| < \delta, \tag{3.4}$$

  where $\|.\|$ is the Euclidean norm and $\delta > 0$ is a small constant;
- partner point of $x_n^{(k)}$:

$$\tilde{x}_n^{(k)} = x_n^{(k)} - \beta \nabla f(x_n^{(k)}) \tag{3.5}$$

  where $\beta > 0$ is a small constant.

The point $x_n \in X$ is a potential starting point for local descent. The set X can be constructed in different ways. For example, if the total number of descents is fixed, it may be defined as the sequence with small covering radius or it could be any low discrepancy sequence in $\mathfrak{X}$.

3.2 Literature review

The main drawback of multistart is efficiency since the same local minimizers may be found repeatedly from applying local search to each sampled point. The following variations of multistart have been proposed in the literature to ensure that local search is only applied to promising points, which will lead to the discovery of a new local minimizer. This improves the efficiency of multistart since the number of repeated local searches to the same local minimizer is reduced.

3.2.1 Clustering methods

Clustering methods involve identifying clusters corresponding to the regions of attraction A_l $(l = 1, 2, ...)$ of a function f and are discussed in detail within [58, 107, 131, 132]. If clusters represent regions of attraction, the number of local searches can be significantly reduced by applying local search to a single point within each cluster.

A detailed overview of various classical algorithms which incorporate clustering methods for multistart is provided in [131], along with references. A general approach from [131, Sect. 5.3.1] utilized by various multistart algorithms that use clustering methods is presented in Table 3.1.

Step	**Action**	**Description**
1	Sample	In general, points are sampled uniformly at random or by using a stratified sample.
2	Concentrate	Approaches to concentrate a sample of points include, retaining a proportion of points with the smallest function values or applying a small number of local search steps to each point. Both approaches can be applied consecutively and the order of application is interchangeable.
3	Apply clustering method	Clustering methods include Density clustering, Single Linkage, Mode Analysis and Multi Level Single Linkage (see [58, 107] for an informative overview of each method).
4	Check stopping criterion	If some stopping criterion is met, go to Step (6). Otherwise, go to Step (5).
5	Transform	A sample may be transformed in the following ways. Firstly, by focusing on clusters with the smallest function values. Secondly, by considering a proportion of points from each cluster or finally, by constructing a sample containing new points and previously found minimizers and sample points. Once the sample has been transformed, go to Step (2).
6	Terminate	If some stopping criterion has been met, terminate the algorithm.

Table 3.1: A general approach from [131, Sect. 5.3.1] utilized by various multistart algorithms that use clustering methods.

To concentrate a sample of points generated by the global phase of multistart, Törn (see [131, Sect. 5.2.2]) introduced the idea of applying a small number of local search steps before implementing a clustering method. Other methods have recently been proposed to sample and concentrate points within the global phase of multistart. In [132], simulated annealing is used as a stochastic method to sample points. However, simulated annealing can also be applied independently to minimize multimodal functions with continuous variables [23]. In [132], it is also proposed to use an estimate of the Hessian to concentrate sample points when the function is smooth and twice differentiable. If a point is within a region of attraction, the function is convex and the Hessian is positive definite. However, if a point lies between two regions of attraction where the function is non-convex, the Hessian is non-positive definite. Hence, it is proposed in [132] to remove sample points that have a non-positive definite Hessian. Due to the computational cost associated with computing the Hessian, it is outlined in [132] that an estimate of the Hessian should be used. However, [132] states that it may be preferable to concentrate a sample of points according to smallest function values if there are many local minimizers, the regions of attraction are small, and the function is not smooth.

Clustering methods utilized by various multistart algorithms are outlined in Step 3 within Table 3.1. Several issues regarding Density clustering, Single Linkage and Mode Analysis are discussed in [58, 107], and are outlined as follows. Firstly, clusters discovered may contain several local minimizers and consequently, the global minimizer may be missed if local search is applied to a single point in each cluster. Secondly, function values at sample points are not incorporated to determine clusters, which, if used, could significantly improve the application of clustering for identifying regions of attraction. Consequently, Multi Level Single Linkage (MLSL) was proposed in [58, 107].

Consider the critical distance used within MLSL [58, Eq. 35]

$$r_k = \pi^{-1/2} \left(\Gamma \left(1 + \frac{d}{2} \right) \mu(\mathfrak{X}) \frac{\sigma \log kd}{kd} \right)^{1/d} \tag{3.6}$$

where Γ is the gamma function, $\mu(.)$ is the Lebesgue measure, k is the iteration number and σ is a parameter. An iteration of MLSL can be summarized in the following way. Once a set of points are sampled and concentrated, points are organized into sequences such that pairwise points are within a critical distance r_k (3.6) from each other and corresponding function values are monotonically increasing. Local search is applied to a point if all the following three conditions hold.

1. Point is not close to any discovered minimizers.
2. Point is not close to the boundary of the feasible domain.
3. There is no other point within a critical distance of r_k (3.6), which has a smaller function value.

MLSL portrays similar behaviour to a local search procedure when points are organized according to pairwise distances and function values, since a trajectory within a region of attraction will be identified. The choice of r_k (3.6) is discussed in [58, 107] and decreases as the number of iterations increase. Modified versions of MLSL have been proposed in [71, 72].

A self-organizing clustering technique is proposed in [132] which repeatedly splits and merges clusters. For each cluster configuration, a similarity measure is applied to evaluate the dispersion of clusters compared to the distances between different cluster centres. In order for two clusters to be dissimilar, it is desirable for the dispersion of each cluster to be small and the distance between the respective cluster centres to be large. Therefore, the self-organizing clustering process is continued until all the resultant clusters produce small values of the similarity measure. A recent review of methods inspired by clustering, distance and similarity based criteria to determine when to apply local search is presented in [73, Sect. 3.1].

Although clustering methods for multistart have proven successful in a variety of numerical experiments, a key drawback discussed in [29, 73] is that the clustering problem becomes increasingly difficult as the dimension increases. This is because it will be impossible to generate a finite sample of points such that the feasible domain is well covered.

3.2.2 Domain elimination and zooming

The domain elimination and zooming algorithms are proposed in [29] for global optimization problems. The objective of both algorithms is to explore the feasible domain, whilst ensuring sample points are not near previously discovered local minimizers and trajectories leading to the local minimizers.

The zooming algorithm considers the following constraint [29, Eq. 3],

$$f(x_n) < \gamma f(x_l^*), \tag{3.7}$$

where $f(x_l^*)$ is the function value at the previously found local minimizer x_l^* and γ is a constant. If $f(x_l^*)$ is positive, then $\gamma < 1$. Otherwise, if $f(x_l^*)$ is negative, then $\gamma > 1$. Suppose $f(x_l^*)$ is positive, then the value of γ can be set to a small value, or set to large value which is gradually reduced at each iteration of the zooming algorithm. Typically, (3.7) is applied until $f(x_n)$ is a very small value. The aim of (3.7) is to repeatedly reduce the feasible domain to find the region of attraction of the global minimizer.

The zooming and domain elimination algorithms consist of a global and local phase. For the global phase, starting points x_n are uniformly sampled from the feasible domain. To apply the local phase to x_n, all criteria in Table 3.2 must be satisfied.

Criteria	**Description**
C1	x_n must not be near any previously discovered local minimizers.
C2	x_n must not be near any previous starting points.
C3	x_n must not be near any previously rejected points.
C4	x_n must not be near a previous trajectory from a starting point to a local minimizer.

Table 3.2: Criteria C1-C4 that must be satisfied to apply the local phase within the zooming and domain elimination algorithms. The concept of x_n being near to other points or a trajectory is formally discussed in [29, Sect. 6].

Initially, the list of rejected points is empty. However, if at least one of the criteria C1-C4 in Table 3.2 is not satisfied, x_n is added to the set of rejected points and a new starting point x_{n+1} will be sampled uniformly at random from the feasible domain. On the other hand, if x_n satisfies all the criteria C1-C4 in Table 3.2, then x_n will be used for the local phase. The concept of x_n being near other points or a trajectory is formally discussed in [29, Sect. 6].

The local phase involves applying M iterations of a local search procedure to x_n and deciding whether to stop or apply more iterations of local search. The local search is stopped if at least one of the criteria C2-C4 in Table 3.2 fails or if a new local minimizer is discovered. Furthermore, the zooming algorithm also stops local search if the total number of iterations of local search is larger than some predefined tolerance. Otherwise, if local search is not stopped, more iterations are applied and the process is repeated until the local search is stopped. If a local minimizer is found within the zooming algorithm, then the zooming constraint (3.7) is redefined and the tolerance for the maximum number of local search iterations may be adjusted to improve efficiency. Explicit stopping criteria for both algorithms is provided in [29, Sect. 6.5].

The key difference between both algorithms is the application of (3.7) in order to repeatedly reduce the feasible domain. Moreover, the aim of domain elimination is to locate all local minima of an objective function whereas the aim of the zooming algorithm is to find the global minima. By [29], the domain elimination and zooming algorithms may not perform well if the dimension is very large since a local search trajectory or a random point may not be classified as being near previous trajectories. As a result, many points may not be rejected, which increases the computational cost of both algorithms.

3.2.3 Repulsion

The repulsion algorithm in [115] aims to reduce the number of local searches performed by diverse exploration of the feasible domain $\mathfrak{X}$. An iteration of the repulsion algorithm consists of sampling a point x_n uniformly at random from the feasible domain $\mathfrak{X}$, and then transforming x_n into a point z_n. The point z_n is constructed by allowing previous points x_i ($i = 1, \dots, n-1$) to repel x_n away if the distance between x_i and x_n is less than some tolerance. The following equation [115, Eq. 12] transforms x_n into z_n,

$$z_n = x_n + \sum_{i=1}^{n-1} \Delta(r_i) \frac{x_n - x_i}{r_i}, \tag{3.8}$$

where r_i is the distance between x_i and x_n, and r_0 is some threshold distance used to determine $\Delta(r_i)$. That is, if $r_i > r_0$ then $\Delta(r_i) = 0$. Otherwise, $\Delta(r_i) = \alpha - \beta r_i$, where $\beta = \alpha / r_0$ and $\alpha \in [0, r_0)$. Application of the repulsion algorithm ensures different subsets of the feasible domain $\mathfrak{X}$ are explored, which allows the opportunity to find the global minimizer.

Numerical examples in [115] suggest that increasing α will increase the probability of initiating local searches in undiscovered regions of attraction. However, there is no general method to determine values for r_0 or α.

3.2.4 Learning for Global Optimization

The LeGO (Learning for Global Optimization) method is proposed in [16], to determine promising points for local search. LeGO consists of two phases. The first phase involves generating a number of points x_n ($n = 1, \dots, N$) and applying several iterations of local search to each point to obtain $x_n^{(k)}$. In addition, a positive or negative label is assigned to each point x_n depending on whether $f(x_n^{(k)})$ is less than some particular value. The value can be chosen using knowledge of the objective function or using cross-validation. The training set consists of the starting points with associated labels, and a Support Vector Machine (SVM) is trained on the labelled training set. The second phase consists of generating more starting points and using SVM to predict a positive or negative label for each starting point. If the label is positive, then local search is applied. Numerical experiments in [16] show that LeGO is useful for finding promising points for local search, even for more complex methods than multistart.

A drawback of using SVM is that the training cost may become significant if SVM is retrained with additional training samples. Also, the key parameters used for SVM will need to be chosen carefully to ensure generalization to unseen training samples. Both issues have been discussed in [16] as possible research areas to consider.

3.2.5 Surrogate based multistart

Recently, variations of multistart have been proposed in [68, 98] for problems with expensive objective function evaluations. Furthermore, derivatives of the function are assumed to be unavailable. Although the class of objective functions applied with surrogate based multistart is different from those used for METOD, the purpose of discussing surrogate based multistart is to demonstrate that reducing the number of local searches within multistart is still a relevant research problem. The inclusion of surrogate models within multistart [68, 98] minimizes the number of expensive objective function evaluations performed. In addition, [68, 98] considers measures to reduce the number of repeated local searches to the same local minimizer.

The following surrogate based multistart algorithm (SOMS) is proposed in [68] to reduce the number of expensive function evaluations performed during the global phase of multistart. Initially, a response surface is built to approximate the objective function. An iteration of SOMS involves sampling a number of starting points uniformly at random from the feasible domain and evaluating the surrogate model at each starting point. A set containing the new sampled points and previously sampled points is formed. A proportion of points with the smallest surrogate model values is retained, and the true objective function is evaluated at each retained point. An additional point is sampled uniformly at random from the feasible domain and is added to the set of retained points. This encourages diversity since the additional point may provide extra information on the true objective function and can be used to improve the fit of the surrogate model. All retained points and the additional point are sorted according to objective function values. Similar to MLSL [58, 107], points are removed if they are within distance r_k (3.6) of other points that have smaller objective function values or if local search has already been applied. Consequently, local search is applied to the remaining points. At the end of the iteration, the algorithm terminates if some stopping criterion is met. Otherwise, the surrogate model is updated, and another iteration is started. Although surrogate models are used to reduce expensive objective function evaluations, the importance of limiting the number of local searches is clearly emphasized in [68]. That is, a sample of points is concentrated by removing points with large surrogate model values and also by applying a similar procedure to MLSL.

The objective of [98] is to use a surrogate model to determine a second order model, which will be used to compute the search direction. The convexity of the second order model will need to be checked to determine whether the Newton direction can be used as the search direction. Otherwise, an anti-gradient direction is used, and the step length is computed according to golden section search. In addition, it is proposed in [98] to use a trust region method to reduce the risk of the algorithm moving towards regions where the surrogate model is erroneous. Furthermore, to reduce the number of repeated local searches to the same local minimizer, [98] suggests using a de-clustering method described in [97]. That is, a linear interpolation model and kriging model (see [24, 65]) are constructed using points with known objective function values. The aim is to select points where the variance between both surrogate models is large, suggesting there is limited information for a particular region. Also, it is desirable to choose points with a lower expected value from the surrogate models than the mean of the known objective function values.

The purpose of outlining variations of multistart in [68, 98] is to illustrate that multistart and specifically, reducing the number of repeated local searches to the same local minimizers is still a significant research problem. However, for the remainder of this chapter it is assumed that the objective function and gradient evaluations are inexpensive to compute. Therefore, methods proposed in [68, 98] will not in general, be comparable with METOD.

3.2.6 Summary

Many variations of multistart have been proposed in the literature to reduce the number of repeated local searches to the same local minimizer (see Subsections 3.2.1 - 3.2.5). Furthermore, recent publications [16, 68, 73, 98] imply that reducing the number of repeated local searches in multistart is still very much an active research area in global optimization. A similarity between METOD and some of the discussed variations of multistart is that a small number of local search iterations are applied before deciding whether to stop local search. However, the performance of many methods discussed can depend on the size of dimension or the choice of algorithm parameters. Hence, the remainder of this chapter outlines a variation of multistart called METOD, where clear recommendations of algorithm parameters are provided and shown to perform well for global optimization problems in large dimensions.

3.3 Objective functions

Consider the following objective functions from [54, 144], which will be used throughout this chapter.

Minimum of several quadratic forms function

$$f(x_n) = \min_{1 \le p \le P} \frac{1}{2}(x_n - x_{0p})^T A_p^T \Sigma_p A_p (x_n - x_{0p}), \tag{3.9}$$

where $\mathfrak{X} = [0,1]^d$, $x_n \in \mathfrak{X}$, P is the number of minima, x_{0p} $(p = 1, \dots, P)$ is a random point in $\mathfrak{X}$, A_p $(p = 1, \dots, P)$ is a randomly chosen rotation matrix of size $d \times d$, Σ_p $(p = 1, \dots, P)$ is a diagonal positive definite matrix of size $d \times d$ with smallest and largest eigenvalues λ_{min} and λ_{max} respectively. All other eigenvalues are sampled uniformly from $(\lambda_{min}, \lambda_{max})$. All minima of (3.9) are global and $f(x_{0p}) = 0$ $(p = 1, \dots, P)$. The function parameters are $\lambda_{min} = 1$ and $\lambda_{max} = 10$, with the exception that different values of λ_{max} are tested within Subsection 3.6.3. Parameters d and P are explicitly defined in each section.

Sum of Gaussians function

$$f(x_n) = -\sigma^2 \sum_{p=1}^{P} c_p \exp\left\{ -\frac{1}{2\sigma^2}(x_n - x_{0p})^T A_p^T \Sigma_p A_p (x_n - x_{0p}) \right\}, \tag{3.10}$$

where $\mathfrak{X} = [0,1]^d$, $x_n \in \mathfrak{X}$, c_p $(p = 1, \dots, P)$ is sampled uniformly from $(0.5, 1)$, σ^2 is a fixed constant and all other notation is the same as (3.9). The function parameters are $\lambda_{min} = 1$ and $\lambda_{max} = 10$, with d, σ^2 and P explicitly defined in each section. Since x_{0p}, c_p, A_p and Σ_p $(p = 1, \dots, P)$ are chosen randomly, x^* and $f(x^*)$ will differ for each function of the form (3.10).

Modified Shekel function

$$f(x_n) = -\frac{1}{2} \sum_{p=1}^{P} \left((x_n - x_{0p})^T A_p^T \Sigma_p A_p (x_n - x_{0p}) + b_p \right)^{-1}, \tag{3.11}$$

where $d = 4$, $\mathfrak{X} = [0,10]^d$, $x_n \in \mathfrak{X}$, b_p $(p = 1, \dots, P)$ is a constant, x_{0p} $(p = 1, \dots, P)$ is a random point in $\mathfrak{X}$ and all other notation is the same as (3.9). The function parameters are $\lambda_{min} = 1$, $\lambda_{max} = 10$, $P = 10$, $x_{01} = (4,4,4,4)^T$, $x_{02} = (1,1,1,1)^T$, $x_{03} = (8,8,8,8)^T$, $x_{04} = (6,6,6,6)^T$, $x_{05} = (3,7,3,7)^T$, $x_{06} = (2,9,2,9)^T$, $x_{07} = (5,3,5,3)^T$, $x_{08} = (8,1,8,1)^T$, $x_{09} = (6,2,6,2)^T$, $x_{010} = (7,3.6,7,3.6)^T$, and $b = (1.25, 1.45, 1.45, 1.65, 1.7, 1.8, 1.75, 1.9, 1.7, 1.8)^T$.

The global minimizer is $x^* = (4,4,4,4)^T$. However, since A_p and Σ_p $(p = 1, \dots, P)$ are generated at random, $f(x^*)$ will differ for each function of the form

(3.11).

Hartmann 6 function

$$f(x_n) = -\sum_{j=1}^{4} \alpha_j \exp\left\{ -\sum_{i=1}^{d} c_{j,i}(x_{n_i} - b_{j,i})^2 \right\} \tag{3.12}$$

where $d = 6$, $\mathfrak{X} = [0,1]^d$, $x_n \in \mathfrak{X}$, with $C = (c_{j,i})_{j,i=1}^{4,d}$, $\alpha = (\alpha_j)_{j=1}^{4}$ and $B = (b_{j,i})_{j,i=1}^{4,d}$ where,

$$C = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \quad \alpha = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix} \quad \text{and}$$

$$B = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}.$$

Two local minimizers of (3.12) are identified when a local optimization algorithm is applied to a large number of starting points, which is in agreement with [125]. One of the identified local minimizers is the global minimizer, $x^* = (0.202, 0.150, 0.477, 0.275, 0.312, 0.657)^T$ with $f(x^*) = -3.322$.

Styblinski-Tang function

$$f(x_n) = \frac{1}{2}\sum_{i=1}^{d}(x_{n_i}^4 - 16x_{n_i}^2 + 5x_{n_i}), \tag{3.13}$$

where $\mathfrak{X} = [-5,5]^d$, $x_n \in \mathfrak{X}$ and the total number of local minima is 2^d. For $d = 5$, the global minimizer is $x^* = (-2.903534, -2.903534, -2.903534, -2.903534, -2.903534)^T$ and $f(x^*) = -195.83$.

Qing function

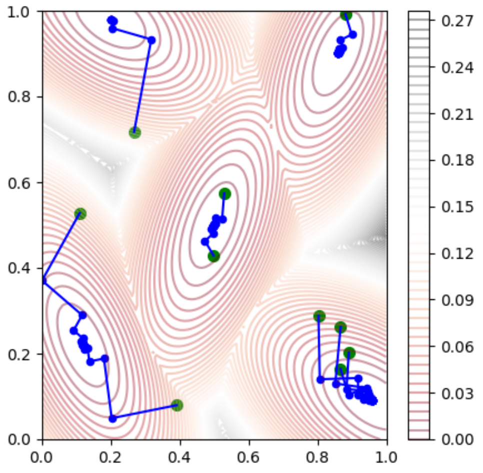
$$f(x_n) = \sum_{i=1}^{d}(x_{n_i}^2 - i)^2, \tag{3.14}$$

where $\mathfrak{X} = [-3,3]^d$, $x_n \in \mathfrak{X}$, the total number of local minima is 2^d and all local minima are global. For $d = 5$, the global minimizers are $x^* = (\pm\sqrt{1}, \pm\sqrt{2}, ..., \pm\sqrt{d})^T$, where $f(x^*) = 0$.

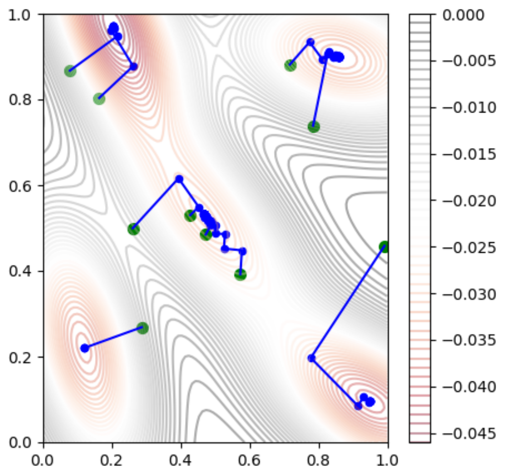
Zakharov function

$$f(x_n) = \sum_{i=1}^{d} x_{n_i}^2 + \left(\sum_{i=1}^{d} 0.5 i x_{n_i}\right)^2 + \left(\sum_{i=1}^{d} 0.5 i x_{n_i}\right)^4 \tag{3.15}$$

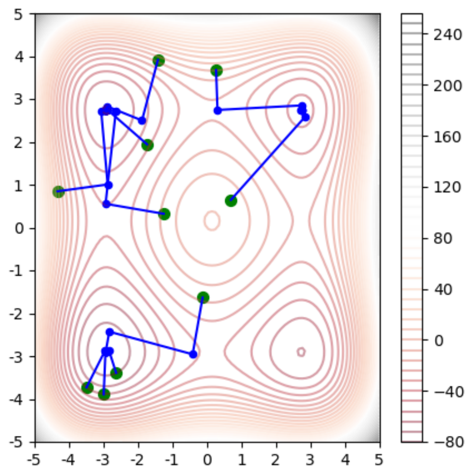
where $d = 10$, $\mathfrak{X} = [-5, 10]^d$ and $x_n \in \mathfrak{X}$. The Zakharov function has one global minimizer $x^* = (0, 0, \ldots, 0)^T$, where $f(x^*) = 0$.



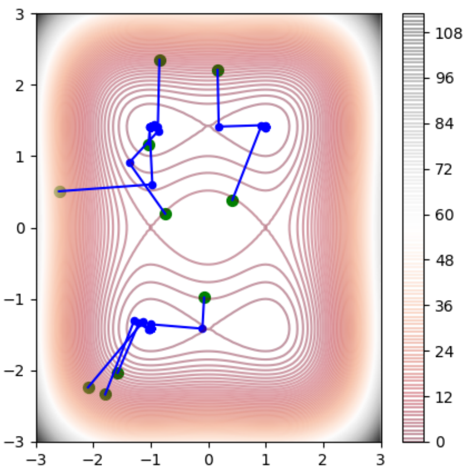
(a) Minimum of several quadratic forms (3.9)



(b) Sum of Gaussians (3.10)



(c) Styblinski-Tang (3.13)



(d) Qing (3.14)

Figure 3.1: Contour plots of various objective functions with trajectories from ten starting points (green) to the local minimizer.

Figure 3.1 shows contour plots of objective functions (3.9), (3.10), (3.13) and (3.14) along with trajectories from ten starting points to the corresponding local minimizer. Objective functions (3.11) and (3.12) have not been plotted since function parameters are defined for dimension greater than two and objective function (3.15) has not been plotted since there is only one minimizer.

3.4 Overview of the METOD algorithm

3.4.1 Overview

For objective functions with locally quadratic behaviour close to the neighbourhoods of local minimizers, the early termination of descents in METOD is achieved by means of a particular inequality. The inequality holds when trajectories are from the region of attraction of the same local minimizer and often violates when the trajectories belong to different regions of attraction. The main idea behind the inequality in METOD is the following theorem and proof from [144, Sect. 2.1].

Theorem 2. *Assume we have a quadratic function,*

$$f(x) = \frac{1}{2}x^T Ax + b^T x + c,$$

where $x, b \in \mathbb{R}^d$, A is a positive definite $d \times d$ matrix and c is a constant. For $x_1, x_2 \in \mathbb{R}^d$ we define their 'partner points' $\tilde{x}_i = x_i - \beta \nabla f(x_i)$ $(i = 1, 2)$. For all $0 < \beta < 1/\lambda_{\max}$ we have,

$$\|\tilde{x}_1 - \tilde{x}_2\| < \|x_1 - x_2\|, \tag{3.16}$$

where $\lambda_{\max}$ is the largest eigenvalue of A.

Proof.
Consider,

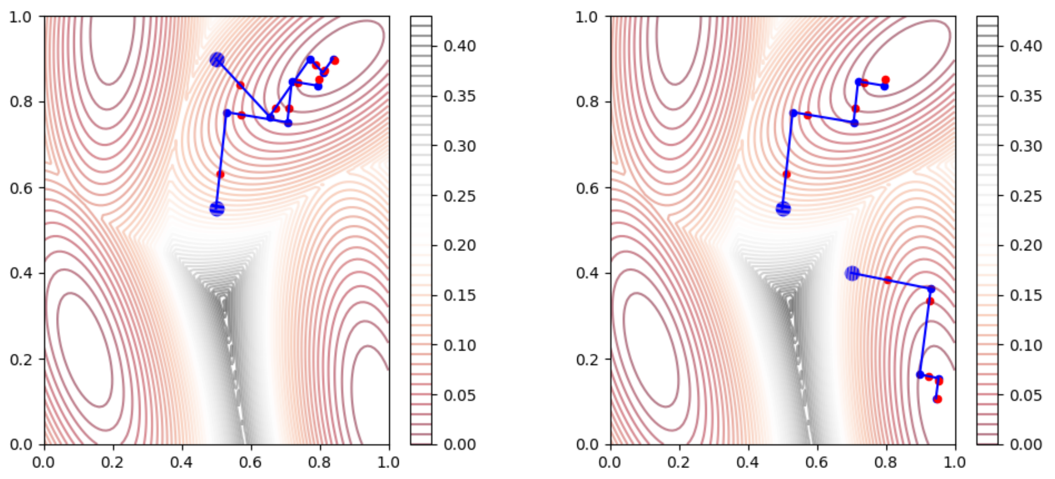
$$(\tilde{x}_1 - \tilde{x}_2) = [x_1 - \beta(Ax_1 + b)] - [x_2 - \beta(Ax_2 + b)] = (I_d - \beta A)(x_1 - x_2),$$

where I_d is an identity matrix of size $d \times d$. Since $0 < \beta < 1/\lambda_{\max}$, then $(I_d - \beta A) < I_d$ and consequently, $(I_d - \beta A)^2 < I_d$, where $<$ means that the difference between the matrices in the right-hand and left-hand sides is positive definite. Hence,

$$\|\tilde{x}_1 - \tilde{x}_2\|^2 = (x_1 - x_2)^T (I_d - \beta A)^2 (x_1 - x_2) < (x_1 - x_2)^T (x_1 - x_2) = \|x_1 - x_2\|^2 \quad \square$$

Inequality (3.16) is absolutely fundamental for METOD. With this inequality, METOD learns whether a particular trajectory has a chance of descending to a local minimizer which has not yet been identified.

An example illustrating the purpose of inequality (3.16) for two points belonging to the same region of attraction and different regions of attraction can be observed



(a) Inequality (3.16) holds for $(x_1^{(k)}, x_2^{(k)})$ and $(\tilde{x}_1^{(k)}, \tilde{x}_2^{(k)})$ with $k = 0, \dots, 4$.

(b) Inequality (3.16) fails for $(x_1^{(k)}, x_2^{(k)})$ and $(\tilde{x}_1^{(k)}, \tilde{x}_2^{(k)})$ with $k = 0, 2, 4$.

Figure 3.2: Steepest descent iterations $x_1^{(k)}$ and $x_2^{(k)}$ (blue) from two starting points with corresponding partner points $\tilde{x}_1^{(k)}$ and $\tilde{x}_2^{(k)}$ (red), where $k = 0, \dots, 4$.

	k	$\|x_1^{(k)} - x_2^{(k)}\|$	$\|\tilde{x}_1^{(k)} - \tilde{x}_2^{(k)}\|$	$\|\tilde{x}_1^{(k)} - \tilde{x}_2^{(k)}\| < \|x_1^{(k)} - x_2^{(k)}\|$
Figure 3.2(a)	0	0.350	0.217	✓
	1	0.125	0.100	✓
	2	0.162	0.127	✓
	3	0.093	0.081	✓
	4	0.077	0.064	✓
Figure 3.2(b)	0	0.250	0.384	✗
	1	0.575	0.560	✓
	2	0.619	0.661	✗
	3	0.731	0.729	✓
	4	0.745	0.761	✗

Table 3.3: Check (3.16) with distances $\|x_1^{(k)} - x_2^{(k)}\|$ and $\|\tilde{x}_1^{(k)} - \tilde{x}_2^{(k)}\|$, where points $x_1^{(k)}$ and $x_2^{(k)}$ are from Figures 3.2(a) and 3.2(b).

in Figure 3.2 and Table 3.3. It can be observed in Figure 3.2(a) and Table 3.3 that inequality (3.16) holds for all points $x_1^{(k)}$ and $x_2^{(k)}$ $(k = 0, \dots, 4)$ which belong to the same region of attraction. Conversely, Figure 3.2(b) and Table 3.3 portrays that the passing or failing of inequality (3.16) may depend on the iteration number k when points $x_1^{(k)}$ and $x_2^{(k)}$ belong to different regions of attraction. Therefore, the zig-zag nature of steepest descent iterations will need to be taken into account when deciding whether to stop local descents early using inequality (3.16) in METOD.

A full description of the METOD algorithm is provided in [144, Sect. 3.2]. However, since the publication of [144], investigations on the accuracy and efficiency of the METOD algorithm have continued and subsequently, a variety of enhancements have been proposed in Subsection 3.4.2. The updated description of the METOD algorithm is provided in Subsection 3.4.3.

Consider the following notation outlined in [144, Sect. 3.1], which will be used to describe the main conditions and proposed enhancements of the METOD algorithm in Subsections 3.4.2 and 3.4.3.

- δ and η: small positive constants;
- M: the minimum number of steepest descent iterations (3.3) applied at each initial point $x_n = x_n^{(0)}$ $(n = 1, 2, \dots, N)$ to obtain $x_n^{(M)}$;
- l: index for the local minimizers and regions of attraction $l = 1, \dots, L$, where L is the total number of local minimizers found so far;
- $x_l^{(K_l)}$: l-th local minimizer found by applying steepest descent iterations (3.3) until the smallest $k = K_l$ is found such that $\|\nabla f(x_n^{(k)})\| < \delta$;
- A_l: l-th region of attraction of local minimizer $x_l^{(K_l)}$.

3.4.2 Main conditions of the METOD algorithm and proposed enhancements

The following steps of the METOD algorithm are generalized in order to discuss the main conditions and enhancements.

1 *Initialization*
   Generate a starting point $x_1^{(0)}$ and apply local descent to find a local minimizer.

2 *n-th iteration*
   Generate a starting point $x_n^{(0)}$ $(n = 2, \dots, N)$.

   i Check whether a region of attraction has been previously discovered.

   ii Determine whether to stop steepest descent iterations (3.3) early.

3 *Return local minimizers*
   It is possible to find the same local minimizer more than once when local descent is applied to different x_n. Consequently, only unique local minimizers are returned at the end of the algorithm.

The necessary conditions applied to carry out each step will be discussed and compared with the original METOD algorithm in [144, Sect. 3.2].

Steps 1 and 2: Check adequacy of starting points $x_n^{(0)}$ $(n = 1, .., N)$

Starting points $x_n^{(0)}$ $(n = 1, \dots, N)$ are required for Steps 1 and 2. It may be possible that $\|\nabla f(x_n^{(0)})\| < \delta$. This may occur if $x_n^{(0)}$ is close to a local minimizer or if $x_n^{(0)}$ is located far away from a local minimizer and in-between two different regions of attraction [144, Sect. 3.3].

Törn introduced the idea of applying a small number of local search steps before implementing a clustering method (see [131, Sect. 5.2.2]). The METOD algorithm uses a similar approach since a warm-up of M steepest descent iterations (3.3) is applied to each starting point $x_n^{(0)}$ in Step 2 before checking whether a region of attraction has been previously identified. This is to ensure that points $x_n^{(M)}$ are within close vicinity of a region of attraction so that the correct decision can be made on terminating local descents early. However, if $\|\nabla f(x_n^{(0)})\| < \delta$ then it is possible that M will need to be considerably large, which will reduce the efficiency of the METOD algorithm. On the other hand, setting M too small will reduce accuracy since $x_n^{(M)}$ may not be within close vicinity of a region of attraction and local descents may be incorrectly terminated.

The following enhancement is proposed to improve the efficiency and accuracy of the METOD algorithm. If $\|\nabla f(x_n^{(0)})\| < \delta$, a new starting point $x_n^{(0)}$ will be generated until $\|\nabla f(x_n^{(0)})\| \geq \delta$. This ensures that starting points $x_n^{(0)}$ which satisfy $\|\nabla f(x_n^{(0)})\| \geq \delta$ are used within the METOD algorithm.

Step 2(i): Check whether a region of attraction has been previously discovered

Consider the following modification to inequality (3.16) in Theorem 2,

$$\|\tilde{x}_1 - \tilde{x}_2\| \leq \|x_1 - x_2\|, \tag{3.17}$$

where (3.17) holds for all $0 < \beta \leq 2/\lambda_{\max}$. A larger range of values for β can be used to compute partner points $\tilde{x}_1$ and $\tilde{x}_2$ since the inequality in (3.17) is relaxed.

Recall that the proof for [144, Thm. 1] states that if $0 < \beta < 1/\lambda_{\max}$, then $(I_d - \beta A)^2 < I_d$ and as a result (3.16) holds. In a similar manner, (3.17) holds when $0 < \beta \leq 2/\lambda_{\max}$, since $(I_d - \beta A)^2 \leq I_d$, where $\leq$ means that the difference between the matrices in the right-hand and left-hand sides is positive semi-definite. Throughout this chapter, inequality (3.17) will be used for the METOD algorithm.

It can be observed in Figure 3.2 that the trajectory of steepest descent iterations (3.3) to a local minimizer follow a zigzag pattern. This may result in inequality (3.17) holding for two points belonging to different regions of attraction, which is not desirable. For example, Table 3.3 and Figure 3.2(b) show that inequality (3.17) holds when $k = 1, 3$ for two points belonging to different regions of attraction. Henceforth, two consecutive points $x_2^{(M-1)}$ and $x_2^{(M)}$, along with each point $x_1^{(k)}$ ($k = M-1, \dots, K_n$), will be applied with inequality (3.17) to ensure that the decision on terminating local descents does not depend on the zigzag nature of steepest descent iterations.

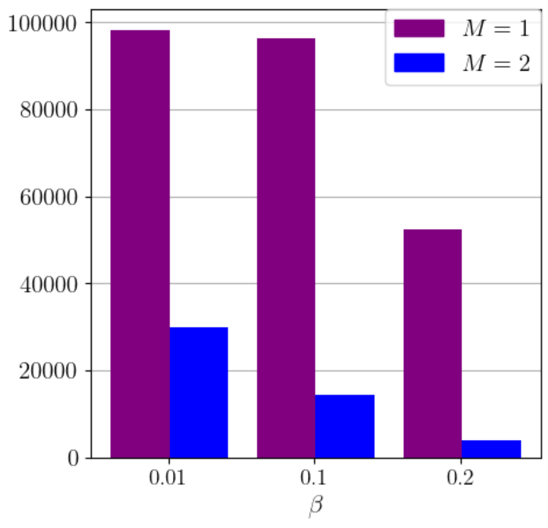
The following variation of [144, Eq. 9] enables the METOD algorithm to make a decision on terminating local descents in Step 2. For each l and all $i = M-1, \dots, K_l$, the algorithm will test the following condition:

$$\|\tilde{x}_n^{(M)} - \tilde{x}_l^{(i)}\| \leq \|x_n^{(M)} - x_l^{(i)}\| \text{ and } \|\tilde{x}_n^{(M-1)} - \tilde{x}_l^{(i)}\| \leq \|x_n^{(M-1)} - x_l^{(i)}\|. \tag{3.18}$$

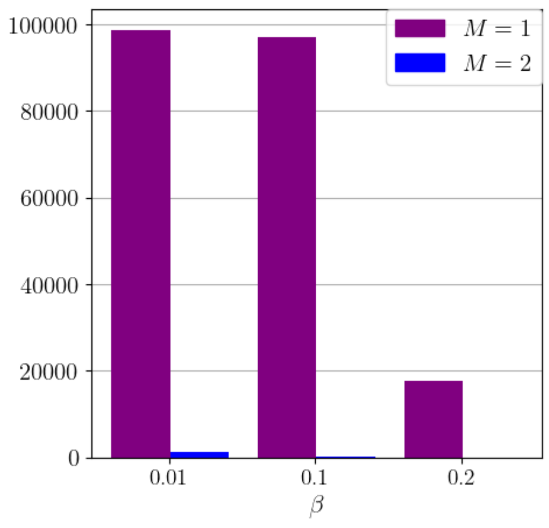
It can be observed that condition [144, Eq. 9] uses inequality (3.16), whereas condition (3.18) uses inequality (3.17).

Step 2(ii): Simplify the decision process for terminating steepest descent iterations early

To terminate local descents early, (3.18) must be satisfied for a least one index l. Let S_n be the set of indices l, such that x_n may belong to region of attraction A_l. If condition (3.18) holds, index l is added to the set S_n. The description of the METOD algorithm in [144, Sect. 3.2] accounts for the possibility of S_n containing several indices l since the checking of condition (3.18) is continued for $l = 1, \dots, L$.



(a) $d = 50$



(b) $d = 100$

Figure 3.3: Total number of points which satisfy (3.18) for more than one index $l = (1, \dots, L)$ for 100 functions of the form (3.9), with $N = 1000$ and $P = 50$

Figure 3.3 shows a large number of points satisfy (3.18) for more that one index l with functions of the form (3.9), especially for small values of β and M. Consequently, steepest decent iterations (3.3) are terminated early. However, the checking of condition (3.18) for different l is continued and will increase the time taken by the METOD algorithm to find new local minimizers. Since the main focus of the METOD algorithm is to discover local minimizers in an efficient manner, it is proposed to stop checking condition (3.18) immediately when S_n contains one index l. Otherwise, if (3.18) is not satisfied for any l, then steepest descent iterations (3.3) are continued. Consequently, only points that do not satisfy (3.18) will be assigned a region of attraction and points which satisfy (3.18) will be discarded. Therefore, steps outlined in [144, Sect. 3.2] to assign a region of attraction to all starting points x_n $(n = 1, \dots, N)$ will not be applicable.

Step 3: Check for unique local minimizers

If condition (3.18) does not hold for points $x_n^{(M-1)}$ and $x_n^{(M)}$ with any l, then steepest descent iterations (3.3) are applied until a local minimizer $x_n^{(K_n)}$ is found. However, we may have that local minimizer $x_n^{(K_n)}$ has already been discovered. As a consequence, the final step of the METOD algorithm is concerned with returning unique local minimizers only. Hence, the following condition is tested for $i = 1, \dots, L$ and $j = i + 1, \dots, L$

$$\|x_i^{(K_i)} - x_j^{(K_j)}\| > \eta. \tag{3.19}$$

If condition (3.19) fails for any j, then minimizers $x_i^{(K_i)}$ and $x_j^{(K_j)}$ are the same and j is removed from the set of indices $l = 1, \dots, L$.

A proposed enhancement is to check (3.19) at the end of the METOD algorithm. Previously, (3.19) was checked for each discovered local minimizer $x_n^{(K_n)}$. That is, if a local minimizer $x_n^{(K_n)}$ had already been discovered, the trajectory $x_n^{(k)}$ $(k = 0, \dots, K_n)$ was discarded. On reflection, evaluating (3.18) with more than one trajectory to the region of attraction of the same local minimizer may be beneficial in terminating other local descents early. Henceforth, the efficiency of the METOD algorithm can be improved by using the extra information available from repeated local descents.

3.4.3 METOD algorithm

The METOD algorithm can be split into the following three parts.

1. *Initialization*

   Choose $x_1 = x_1^{(0)} \in X$ that satisfies $\|f(x_1^{(0)})\| \geq \delta$. Use iterations (3.3) to find a minimizer $x_1^{(K_1)}$. For all points $x_1^{(k)}$ computed in (3.3) with $k = M-1, M, \ldots, K_1$, compute the associated partner points using (3.5) and set $L \leftarrow 1$.

2. *n-th iteration*

```
for $n = 2$ to $N$ do
    Choose $x_n = x_n^{(0)} \in X$ which satisfies $\|f(x_n^{(0)})\| \geq \delta$. Compute $x_n^{(j)}$ for
    $j = 1, \ldots, M$ and the associated partner points using (3.5).
    for $l = 1$ to $L$ do
        if condition (3.18) is satisfied for $i = M-1, \ldots, K_l$ then
            $S_n \leftarrow l$ and terminate iterations (3.3) which have started at $x_n$.
            Go to the beginning of Step (2) to generate a new starting point.
        end if
    end for
    At this stage, $S_n$ must be empty. Hence, let $x_{L+1} = x_n$ and continue
    iterations (3.3) until a minimizer $x_{L+1}^{(K_{L+1})}$ is found. For all points
    $x_{L+1}^{(k)}$ $(k = M-1, \ldots, K_{L+1})$, compute the associated partner points
    using (3.5) and set $L \leftarrow L+1$.
end for
```

3. *Return unique minimizers.*

```
for $i = 1$ to $L$ do
    for $j = i+1$ to $L$ do
        if condition (3.19) is not satisfied for $x_i^{(K_i)}$ and $x_j^{(K_j)}$ then
            Remove index $j$ from the set of indices $l = 1, \ldots, L$.
        end if
    end for
end for
```

3.4.4 Potential inaccuracies

There are two types of errors that may occur from applying the METOD algorithm. The first possible error is that condition (3.18) may not hold for points that belong to the same region of attraction. Consequently, steepest descent iterations (3.3) are continued, and the same local minimizer is found, which will reduce the efficiency of the METOD algorithm. However, this will not affect the accuracy of the METOD algorithm. The second possible error occurs if condition (3.18) holds for points belonging to different regions of attraction. As a result, a region of attraction of a new local minimizer may be missed, which reduces the accuracy of the METOD algorithm.

Both errors may occur if the parameters M and β are not chosen appropriately. Investigations on suitable values of M and β for a range of test functions are conducted, and general recommendations for M and β are provided for different classes of functions. Furthermore, both errors can occur if the total number of steepest descent iterations (3.3) to find a local minimizer is small and consequently, checking condition (3.18) will be unreliable. In that case, it is proposed not to apply condition (3.18) as this may lead to an error occurring.

3.5 Studying the violation of the fundamental inequality of METOD

3.5.1 Overview

Condition (3.18) within the METOD algorithm determines whether to terminate local descents early. Recall that condition (3.18) is based entirely on evaluating the fundamental inequality (3.17) for various points. This section aims to study the number of violations of (3.17) for points that belong to the same and different regions of attraction.

To study the number of violations of (3.17), 100 functions of the form (3.9) - (3.15) are generated. For functions of the form (3.9) and (3.10), the number of local minimizers is $P = 2$ and $P = 10$ respectively. Furthermore, $d = 100$ for functions (3.9) and $d = 20$ for functions (3.10) with $\sigma^2 = 0.7$. For each function (3.9) - (3.15), random starting points $x_n \in \mathfrak{X}$ ($n = 1, \ldots, 100$) are also generated. Steepest descent iterations (3.3) are applied to each x_n until the smallest $K_n = k$ is found such that $\|\nabla f(x_n^{(k)})\| < \delta$, where $\delta > 0$ is a small positive constant. The value of δ is chosen to ensure $x_n^{(K_n)}$ is close to a local minimizer of a function. It is possible that $x_n^{(k)} \notin \mathfrak{X}$ ($k = 1, \ldots, K_n - 1$). Whilst studying the number of violations of (3.17), $x_n^{(k)} \notin \mathfrak{X}$ is permitted.

The inequality (3.17) is evaluated for pairs of points $x_i^{(k_i)}$ ($i = 1, \ldots, 100$) and $x_j^{(k_j)}$ ($j = i + 1, \ldots, 100$), where points can either belong to the same region of attraction of a local minimizer or different regions of attraction associated with different local minimizers. The impact of β on the frequency of violations of (3.17) will be investigated. During testing, inequality (3.17) will be applied with ($x_i^{(k_i)}$, $x_j^{(k_j)}$), ($x_i^{(k_i)}$, $x_j^{(k_j-1)}$), ($x_i^{(k_i-1)}$, $x_j^{(k_j)}$) and ($x_i^{(k_i-1)}$, $x_j^{(k_j-1)}$). Specifically, it is checked whether all the following inequalities hold

$$\begin{gathered}
\|\tilde{x}_i^{(k_i)} - \tilde{x}_j^{(k_j)}\| \leq \|x_i^{(k_i)} - x_j^{(k_j)}\| \\
\|\tilde{x}_i^{(k_i)} - \tilde{x}_j^{(k_j-1)}\| \leq \|x_i^{(k_i)} - x_j^{(k_j-1)}\| \\
\|\tilde{x}_i^{(k_i-1)} - \tilde{x}_j^{(k_j)}\| \leq \|x_i^{(k_i-1)} - x_j^{(k_j)}\| \\
\|\tilde{x}_i^{(k_i-1)} - \tilde{x}_j^{(k_j-1)}\| \leq \|x_i^{(k_i-1)} - x_j^{(k_j-1)}\|.
\end{gathered}$$

A violation of (3.17) with ($x_i^{(k_i)}$, $x_j^{(k_j)}$), ($x_i^{(k_i)}$, $x_j^{(k_j-1)}$), ($x_i^{(k_i-1)}$, $x_j^{(k_j)}$) and ($x_i^{(k_i-1)}$, $x_j^{(k_j-1)}$) refers to at least one of the four inequalities above failing. Henceforth, only a subset of inequalities of (3.18) used by the METOD algorithm is tested in this section. Furthermore, k_i and k_j will be set to relatively small values which is equivalent to applying (3.17) with points near the beginning of the corresponding trajectories $x_i^{(k_i)}$ ($k_i = 1, \ldots, K_i$) and $x_j^{(k_j)}$ ($k_j = 1, \ldots, K_j$). It will be shown in Section 3.7 that evaluating (3.17) with points near the beginning of each trajectory significantly influences whether the main condition of the METOD algorithm (3.18) holds overall. That is, the key trends of numerical examples with the METOD algorithm in Section 3.7 can be explained by the violations of (3.17) with points near the beginning of each trajectory.

Table 3.4 shows the value of δ, average number of steepest descent iterations (3.3) and average $\|\nabla f(x_n)\|$ for functions of the form (3.9) - (3.15). For some functions, a very small value of δ is required to ensure local minimizers are discovered instead of saddle points. Alternatively, local minimizers of some functions can be found easily when δ is a slightly larger value, resulting in a smaller number of steepest descent iterations (3.3) and improved efficiency. Selecting $\delta = 0.0001$ will improve accuracy of steepest descent iterations (3.3) for functions (3.10), (3.11), (3.13), (3.14) and (3.15). Conversely, $\delta = 0.01$ is sufficient for functions (3.9) and (3.12).

Tables 3.6 - 3.12 show the proportion of violations of (3.17) with ($x_i^{(k_i)}$, $x_j^{(k_j)}$), ($x_i^{(k_i)}$, $x_j^{(k_j-1)}$), ($x_i^{(k_i-1)}$, $x_j^{(k_j)}$) and ($x_i^{(k_i-1)}$, $x_j^{(k_j-1)}$) compared to the total number of pairs (x_i, x_j) tested, for various β, k_i and k_j. For example, consider two pairs of points (x_1, x_2) and (x_3, x_4). Suppose a violation of (3.17) occurs with ($x_1^{(k_1)}$, $x_2^{(k_2)}$), ($x_1^{(k_1)}$, $x_2^{(k_2-1)}$), ($x_1^{(k_1-1)}$, $x_2^{(k_2)}$) and ($x_1^{(k_1-1)}$, $x_2^{(k_2-1)}$) and suppose (3.17) holds for all ($x_3^{(k_3)}$, $x_4^{(k_4)}$), ($x_3^{(k_3)}$, $x_4^{(k_4-1)}$), ($x_3^{(k_3-1)}$, $x_4^{(k_4)}$) and ($x_3^{(k_3-1)}$, $x_4^{(k_4-1)}$). Then the proportion of violations will be

$$\text{Proportion of violations} = \frac{\text{Number of violations of (3.17)}}{\text{Number of pairs of points}} = \frac{1}{2}.$$

Function	δ	Average number of steepest descent iterations (3.3)	Average $\|\nabla f(x_n)\|$
Minimum of several quadratic forms (3.9)	0.01	21.93	23.68196
Sum of Gaussians (3.10)	0.0001	39.91	0.01164
Modified Shekel (3.11)	0.0001	39.60	0.01488
Hartmann 6 (3.12)	0.01	148.89	1.82010
Styblinski-Tang (3.13)	0.0001	7.74	109.98983
Qing (3.14)	0.0001	25.79	50.28710
Zakharov (3.15)	0.0001	6.91	26739579.85486

Table 3.4: Value of δ, average number of steepest descent iterations (3.3) and average $\|\nabla f(x_n)\|$ for functions of the form (3.9) - (3.15).

Pairs of points (x_i, x_j) may either belong to the same region of attraction or different regions of attraction.

The number of pairwise points belonging to the same and different regions of attraction is presented in Table 3.5, where results for the Zakharov function (3.15) have been omitted since there is just one minimizer. Consequently, all pairwise points will belong to the same region of attraction. The purpose of Table 3.5 is to provide insight on the number of pairwise points belonging to the same and different regions of attraction, which are used to compute the proportion of violations. The sum of pairwise points belonging to the same and different regions of attraction for each function is $(N \times (N-1))/2$, where $N = 100$ is the total number of random starting points $x_n \in \mathfrak{X}$ $(n = 1, \dots, 100)$ generated. It can be observed in Table 3.5 that the sum of pairwise points belonging to the same and different regions of attraction is $100 \times (N \times (N-1))/2$ since the experiment is repeated for 100 functions of the form (3.9) - (3.14), each with a set of random starting points $x_n \in \mathfrak{X}$ $(n = 1, \dots, 100)$.

The values assigned to k_i and k_j are determined by the average number of steepest descent iterations (3.3) required to find a region of attraction of a local minimizer, and the Euclidean norm of the gradient at starting points x_n $(n = 1, \dots, 100)$. Functions of the form (3.9) - (3.15) satisfy one of the following three criteria to determine values for k_i and k_j. If the average number of steepest descent iterations (3.3) and average $\|\nabla f(x_n)\|$ is large (i.e. $\|\nabla f(x_n)\| > 10$ in Table 3.4), then $k_i, k_j = 1, 2, 3$. Alternatively, if the average number of steepest descent iterations (3.3) is large and average $\|\nabla f(x_n)\|$ is small (i.e. $\|\nabla f(x_n)\| < 10$ in Table 3.4), then $k_i, k_j = 2, 3$. Otherwise, if the average number of steepest descent iterations (3.3) is small

Function	Same region of attraction	Different regions of attraction
Minimum of several quadratic forms (3.9)	280266	214734
Sum of Gaussians (3.10)	84459	410541
Modified Shekel (3.11)	85795	409205
Hartmann 6 (3.12)	286378	208622
Styblinski-Tang (3.13)	15631	479369
Qing (3.14)	15383	479617

Table 3.5: The total number of pairwise points belonging to the same and different regions of attraction for functions of the form (3.9) - (3.14).

and average $\|\nabla f(x_n)\|$ is relatively large (i.e. $\|\nabla f(x_n)\| > 10$ in Table 3.4), then $k_i, k_j = 1, 2$.

Tables 3.6 - 3.8 show the proportion of violations for functions of the form (3.9) - (3.11), with $\lambda_{min} = 1$, $\lambda_{max} = 10$ and $\beta \in (0, 2/\lambda_{\max}]$. For functions (3.12) - (3.15), λ_{max} is unknown and an upper bound for β cannot be determined. Therefore, a range of values for β are tested in Tables 3.9 - 3.12.

For each function of the form (3.9) - (3.15), an adequate choice for the value of β should maximize the number of violations when points belong to different regions of attraction, and minimize the number of violations when points belong to the same region of attraction. Maximizing the number of violations for points belonging to different regions of attraction ensures that unknown regions of attraction are discovered, which improves the accuracy of the METOD algorithm. Furthermore, minimizing the number of violations for points belonging to the same region of attraction allows the early termination of local descents when a region of attraction has already been discovered. This improves the efficiency of the METOD algorithm by reducing the number of repeated local descents to the same region of attraction.

3.5.2 Minimum of several quadratic forms function

For functions of the form (3.9), there are no violations when points belong to the same region of attraction and consequently, results for points which belong to the same region of attraction have been omitted. Therefore, if a region of attraction has already been discovered by the METOD algorithm, iterations of steepest descent (3.3) are terminated early for points which belong to the same region of attraction. Table 3.4 shows the average number of steepest descent iterations (3.3) and average

β	k_i, k_j	1	2	3
	1	0.0003	0.0998	0.1001
0.01	2	0.1043	0.9995	0.9996
	3	0.1045	0.9996	0.9997
	1	0.0282	0.4118	0.4108
0.1	2	0.4153	1.0000	1.0000
	3	0.4144	1.0000	1.0000
	1	0.9983	0.9966	0.9964
0.2	2	0.9960	1.0000	1.0000
	3	0.9959	1.0000	1.0000

Table 3.6: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$ that belong to different regions of attraction, with 100 functions of the form (3.9) for various β, $P = 2$ and $d = 100$.

$\|\nabla f(x_n^{(0)})\|$ is large for functions of the form (3.9). Therefore, $k_i, k_j = 1, 2, 3$ in Table 3.6. In addition, Table 3.5 shows the number of pairs of points that belong to the same region of attraction and different regions of attraction is 280266 and 214734, respectively.

Table 3.6 shows that when $k_i = 1$ or $k_j = 1$, the proportion of violations is low when $\beta = 0.01, 0.1$ and high when $\beta = 0.2$. This suggests that if $M = 1$ and $\beta = 0.01, 0.1$, the METOD algorithm will assign many points to the wrong region of attraction and miss opportunities to identify new regions of attraction corresponding to new local minimizers. However, if $M = 1$ and $\beta = 0.2$, the METOD algorithm will identify regions of attraction which have not been discovered. For $k_i, k_j \geq 2$, the proportion of violations is large for all values of β, which suggests that setting $M \geq 2$ will reduce the risk of the METOD algorithm assigning points to the wrong region of attraction.

3.5.3 Sum of Gaussians function

Table 3.4 shows the average number of steepest descent iterations (3.3) is large, and average $\|\nabla f(x_n^{(0)})\|$ is small for functions of the form (3.10). As a consequence, $k_i, k_j = 2, 3$ in Tables 3.7(a) and 3.7(b). In addition, Table 3.5 shows the number of pairs of points that belong to the same region of attraction and different regions of attraction is 84459 and 410541, respectively.

It can be observed in Tables 3.7(a) and 3.7(b) that the proportion of violations increases as a function of β. However, the proportion of violations is still relatively low for all values of β in Table 3.7(b). For $k_i, k_j = 3$, the proportion of violations

β	k_i, k_j	2	3
0.01	2	0.8942	0.9295
	3	0.9283	0.9546
0.1	2	0.9171	0.9454
	3	0.9445	0.9655
0.2	2	0.9378	0.9598
	3	0.9588	0.9749

(a) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to different regions of attraction.

β	k_i, k_j	2	3
0.001	2	0.0602	0.0738
	3	0.0694	0.0235
0.1	2	0.0769	0.0861
	3	0.0814	0.0280
0.2	2	0.0976	0.1009
	3	0.0958	0.0331

(b) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to the same region of attraction.

Table 3.7: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$, for 100 functions of the form (3.10) with various β, $P = 10$, $\sigma^2 = 0.7$ and $d = 20$.

decreases for points belonging to the same region of attraction. Furthermore, Table 3.7(a) shows that large values of k_i and k_j result in a higher proportion of violations for points belonging to different attraction regions. Therefore, choosing larger M and β will reduce the risk of the METOD algorithm assigning points to the wrong region of attraction, whilst maximizing the likelihood of terminating descents early for points that belong to the same region of attraction.

3.5.4 Modified Shekel function

β	k_i, k_j	2	3
0.01	2	0.6977	0.7291
	3	0.7278	0.7516
0.1	2	0.6992	0.7308
	3	0.7295	0.7536
0.2	2	0.7008	0.7327
	3	0.7312	0.7557

(a) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to different regions of attraction.

β	k_i, k_j	2	3
0.01	2	0.3482	0.3458
	3	0.3471	0.2490
0.1	2	0.3512	0.3491
	3	0.3502	0.2526
0.2	2	0.3549	0.3530
	3	0.3538	0.2567

(b) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to the same region of attraction.

Table 3.8: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$, for 100 functions (3.11) with various β, $P = 10$ and $d = 4$.

The average number of steepest descent iterations (3.3) is large, and average $\|\nabla f(x_n^{(0)})\|$ is small for functions of the form (3.11) in Table 3.4. Consequently, $k_i, k_j = 2, 3$ in Tables 3.8(a) and 3.8(b). In addition, Table 3.5 shows the number of pairs of points that belong to the same region of attraction and different regions of attraction is 85795 and 409205, respectively.

Tables 3.8(a) and 3.8(b) show that large values of β result in a higher proportion of violations. However, Tables 3.8(a) and 3.8(b) portray that increasing β has only a limited impact on the number of violations. Table 3.8(a) illustrates that larger k_i and k_j increases the proportion of violations for points belonging to different regions of attraction. In contrast, larger values of k_i and k_j reduce the proportion of violations for points belonging to the same region of attraction in Table 3.8(b). Hence, selecting larger M will reduce the risk of the METOD algorithm assigning points to the wrong region of attraction, whilst reducing the number of repeated local descents to regions of attraction already discovered.

3.5.5 Hartmann 6 function

β	k_i, k_j	2	3
0.001	2	0.9473	0.9261
	3	0.9241	0.9079
0.01	2	0.9544	0.9343
	3	0.9328	0.9166
0.1	2	0.9902	0.9813
	3	0.9810	0.9697

(a) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to different regions of attraction.

β	k_i, k_j	2	3
0.001	2	0.2195	0.2095
	3	0.2102	0.2036
0.01	2	0.2362	0.2211
	3	0.2218	0.2120
0.1	2	0.9466	0.9511
	3	0.9524	0.9513

(b) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to the same region of attraction.

Table 3.9: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$, for 100 functions (3.12) with various β and $d = 6$.

Table 3.4 shows that many steepest descent iterations (3.3) are required to find a region of attraction of a local minimizer, and that average $\|\nabla f(x_n^{(0)})\|$ is small for functions (3.12). Therefore, $k_i, k_j = 2, 3$ in Tables 3.9(a) and 3.9(b). In addition, Table 3.5 shows the number of pairs of points that belong to the same region of attraction and different regions of attraction is 286378 and 208622, respectively.

Tables 3.9(a) and 3.9(b) illustrate that the proportion of violations increases as β grows. Table 3.9(b) shows that when $\beta = 0.1$, the proportion of violations is very high for points belonging to the same region of attraction. Therefore, selecting either $\beta = 0.001$ or $\beta = 0.01$ for functions of the form (3.12) reduces the risk of the METOD algorithm assigning points to the wrong region of attraction, whilst also reducing the risk of repeated local descents to regions of attraction already discovered. Furthermore, large values of k_i and k_j reduce the proportion of violations in Tables 3.9(a) and 3.9(b) when $\beta = 0.001, 0.01$. Nevertheless, Table 3.9(a) portrays that the proportion of violations for points belonging to different regions of attraction remains high for all values of k_i and k_j with $\beta = 0.001, 0.01$.

3.5.6 Styblinski-Tang function

β	k_i, k_j	1	2
0.001	1	0.8820	0.9197
	2	0.9197	0.9665
0.005	1	0.8852	0.9223
	2	0.9222	0.9682
0.01	1	0.8892	0.9255
	2	0.9253	0.9702
0.05	1	0.9437	0.9607
	2	0.9602	0.9843

(a) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to different regions of attraction.

β	k_i, k_j	1	2
0.001	1	0.1123	0.1155
	2	0.1177	0.1366
0.005	1	0.1187	0.1208
	2	0.1238	0.1429
0.01	1	0.1267	0.1290
	2	0.1319	0.1506
0.05	1	0.8368	0.7294
	2	0.7211	0.2418

(b) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to the same region of attraction.

Table 3.10: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$, for 100 functions (3.13) with various β and $d = 5$.

For functions (3.13), the average number of steepest descent iterations (3.3) is small, and average $\|\nabla f(x_n^{(0)})\|$ is large in Table 3.4. As a result, $k_i, k_j = 1, 2$ in Tables 3.10(a) and 3.10(b). In addition, Table 3.5 shows the number of pairs of points that belong to the same region of attraction and different regions of attraction is 15631 and 479369, respectively.

Tables 3.10(a) and 3.10(b) show that choosing $\beta = 0.05$ maximizes the proportion of violations for points belonging to the same and different regions of attraction. Consequently, there will be many repeated local descents to the same local minimizer if $\beta = 0.05$ is selected for the METOD algorithm. Hence, selecting $\beta = 0.001, 0.005, 0.01$ reduces the proportion of violations for points belonging to the same region of attraction, resulting in far less repeated local descents to the same local minimizer.

Choosing $k_i, k_j = 2$ increases the proportion of violations for $\beta = 0.001, 0.005, 0.01$ with points belonging to the same and different regions of attraction. However, the proportion of violations increases only marginally for points belonging to the same region of attraction in Table 3.10(b). Overall, selecting small M and $\beta = 0.001, 0.005, 0.01$ will reduce the risk of the METOD algorithm assigning points to the wrong region of attraction, whilst reducing the number of repeated local descents to regions of attraction already discovered.

3.5.7 Qing function

The average number of steepest descent iterations (3.3) and average $\|\nabla f(x_n^{(0)})\|$ is large for functions of the form (3.14) in Table 3.4. Therefore, $k_i, k_j = 1, 2, 3$ in Tables 3.11(a) and 3.11(b). In addition, Table 3.5 shows the number of pairs of points that belong to the same region of attraction and different regions of attraction is 15383 and 479617, respectively.

β	k_i, k_j	1	2	3
0.001	1	0.8405	0.8729	0.8399
	2	0.8720	0.9215	0.9132
	3	0.8384	0.9121	0.9238
0.005	1	0.8442	0.8760	0.8432
	2	0.8749	0.9235	0.9152
	3	0.8417	0.9140	0.9251
0.01	1	0.8491	0.8796	0.8474
	2	0.8789	0.9259	0.9175
	3	0.8461	0.9165	0.9269
0.05	1	0.9028	0.9186	0.8926
	2	0.9184	0.9471	0.9388
	3	0.8921	0.9383	0.9423
0.1	1	0.9808	0.9761	0.9641
	2	0.9759	0.9754	0.9694
	3	0.9637	0.9694	0.9709

(a) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to different regions of attraction.

β	k_i, k_j	1	2	3
0.001	1	0.1350	0.1353	0.0662
	2	0.1342	0.1513	0.0933
	3	0.0662	0.0923	0.0758
0.005	1	0.1391	0.1397	0.0692
	2	0.1383	0.1559	0.0969
	3	0.0688	0.0952	0.0777
0.01	1	0.1450	0.1450	0.0718
	2	0.1439	0.1620	0.1002
	3	0.0721	0.0993	0.0811
0.05	1	0.4637	0.3938	0.3304
	2	0.3867	0.2382	0.1574
	3	0.3213	0.1585	0.1306
0.1	1	0.9579	0.8983	0.8656
	2	0.8944	0.7876	0.7604
	3	0.8608	0.7633	0.7788

(b) $x_i^{(k_i)}$ and $x_j^{(k_j)}$ belong to the same region of attraction.

Table 3.11: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$, for 100 functions (3.14) with various β and $d = 5$.

Tables 3.11(a) and 3.11(b) show the proportion of violations increases as β grows. For functions (3.14), if $\beta = 0.1$ is selected for the METOD algorithm there will be many repeated local descents to the same local minimizer. Selecting $\beta = 0.001, 0.005, 0.01$ ensures that the proportion of violations for points belonging to the same region of attraction in Table 3.11(b) remains relatively low for all values of k_i and k_j. However, the proportion of violations for points belonging to the same region of attraction can be large for $\beta = 0.05$ and small k_i and k_j. Similarly, the largest proportion of violations can also be observed when $\beta = 0.05$ in Table 3.11(a) for points belonging to different regions of attraction.

Setting $k_i, k_j = 3$ for $\beta = 0.001, 0.005, 0.01, 0.05$ will reduce the risk of the METOD algorithm assigning points to the wrong region of attraction, whilst reducing the number of repeated local descents to regions of attraction already discovered.

3.5.8 Zakharov function

β	k_i, k_j	1	2
0.00000001	1	0.0000	0.0000
	2	0.0000	0.0000
0.000001	1	0.8434	0.4532
	2	0.4725	0.0000
0.0001	1	0.9985	0.9593
	2	0.9605	0.0000

Table 3.12: Proportion of violations with points $x_i^{(k_i)}$ and $x_j^{(k_j)}$ that belong to the same region of attraction, for 100 functions (3.15) with various β and $d = 10$.

Since function (3.15) has only one minimizer, the purpose of investigating the number of violations of (3.17) is to observe the choice of β which reduces the number of repeated local descents to the same region of attraction. Table 3.4 shows the average number of steepest descent iterations (3.3) is small, and average $\|\nabla f(x_n^{(0)})\|$ is large for functions (3.15). Therefore, $k_i, k_j = 1, 2$ in Table 3.12.

Table 3.12 highlights that the proportion of violations is high for points belonging to the same region of attraction for $\beta = 0.000001, 0.0001$ and small values of k_i and k_j. If $k_i, k_j \geq 2$ there will be no violations for points belonging to the same region of attraction when $\beta = 0.000001, 0.0001$. However, selecting $\beta = 0.00000001$ will minimize the risk of repeated local descents to the same region of attraction for all values of k_i and k_j. Hence, selecting $\beta = 0.00000001$ is recommended for the METOD algorithm.

3.6 Selection of β

3.6.1 Overview

Recall that β is used to compute partner points in (3.5). It is required that $0 < \beta \leq 2/\lambda_{max}$ for inequality (3.17) to hold for two points belonging to the same region of attraction. In Section 3.5, the effect of different β on inequality (3.17) holding for points belonging to the same and different regions of attraction is investigated. Specifically, Tables 3.6 - 3.12 show the values of β that reduce the risk of the METOD algorithm assigning points to the wrong region of attraction and also reduce the number of repeated local descents to a region of attraction already discovered for functions (3.9) - (3.15). The purpose of this section is to determine why particular values of β improve the accuracy of the METOD algorithm for functions of the form (3.9) - (3.15).

Functions of the form (3.9) - (3.15) will be split into two categories in order to study the influence of β. The first category consists of functions with known λ_{max} and the second category outlines an heuristic approximation of β which can be used for functions with unknown λ_{max}. The motivation for separating functions according to known and unknown λ_{max} is due to the requirement that $\beta \in (0, 2/\lambda_{max}]$ in (3.17). Hence, a clear upper bound on the value of β is provided for functions with known λ_{max}, which is not the case for unknown λ_{max}. Therefore, it is vital to identify methods to approximate adequate values of β for functions with unknown λ_{max}.

3.6.2 Functions with known λ_{max}

Recall objective functions (3.9), (3.10) and (3.11). Suppose that point x_1 belongs to the region of attraction of minimizer x_{01} and another point x_2 belongs to the region of attraction of minimizer x_{02}. Furthermore, suppose x_1 and x_2 are not close the local minimizers x_{01} and x_{02}. Inequality (3.17) can be expanded as follows

$$\begin{aligned}\|x_1 - x_2\|^2 \geq \|\tilde{x_1} - \tilde{x_2}\|^2 &= \|(x_1 - \beta\nabla f(x_1)) - (x_2 - \beta\nabla f(x_2))\|^2 \\ &= \|x_1 - x_2\|^2 + \|b\|^2 + 2b^T(x_1 - x_2),\end{aligned}$$

where $b = \beta(\nabla f(x_2) - \nabla f(x_1))$. Inequality (3.17) holds when $\|b\|^2 + 2b^T(x_1 - x_2) \leq 0$. Let $c_1 = \|b\|^2$ and $c_2 = 2b^T(x_1 - x_2)$, then

$$c_1 + c_2 = \|b\|^2 + 2b^T(x_1 - x_2). \tag{3.20}$$

Since x_1 and x_2 are two different points which are not near the local minimizers x_{01} and x_{02}, we have $c_1 > 0$ and $c_2 \neq 0$.

To test the effect of different β on (3.20), one random function of form (3.9) - (3.11) will be generated, along with two random starting points $x_1, x_2 \in \mathfrak{X}$, which belong to different regions of attraction. For each function of the form (3.9) and (3.10), the number of local minimizers is $P = 2$ and $P = 10$ respectively. Furthermore, $d = 100$ for function (3.9) and $d = 20$ for function (3.10) with $\sigma^2 = 0.7$. Steepest descent iterations (3.3) will be applied to obtain $x_1^{(k_1)}$ and $x_2^{(k_2)}$. Values c_1 and c_2 in (3.20) will be calculated for different β.

Figures 3.4 - 3.6 show values of c_1 and c_2 in (3.20) calculated for various β with points $x_1^{(k_1)}$ and $x_2^{(k_2)}$ that belong to different regions of attraction for a function of the form (3.9) - (3.11), with $k_1, k_2 = 1, 2$. Table 3.13 shows $|c_1/c_2|$ for various β with values of c_1 and c_2 from Figures 3.4 - 3.6.

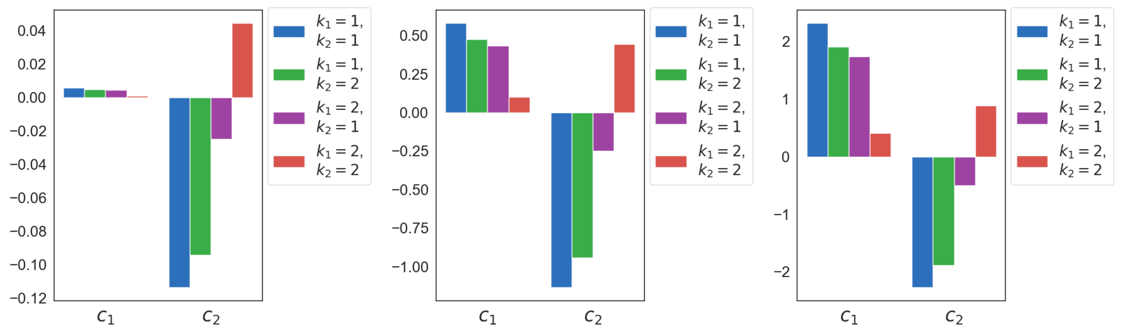


Figure 3.4: Components c_1 and c_2 calculated for $\beta = 0.01$ (left), $\beta = 0.1$ (centre) and $\beta = 0.2$ (right) with $x_1^{(k_1)}$ and $x_2^{(k_2)}$ that belong to different regions of attraction, where the function is of the form (3.9).

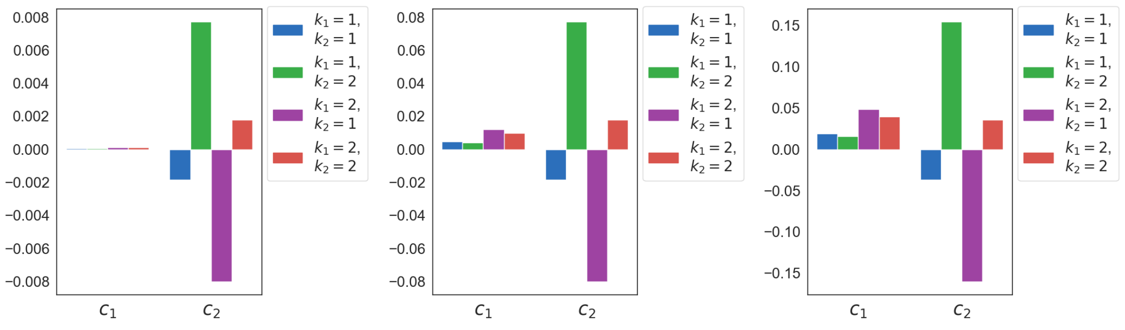


Figure 3.5: Components c_1 and c_2 calculated for $\beta = 0.01$ (left), $\beta = 0.1$ (centre) and $\beta = 0.2$ (right) with $x_1^{(k_1)}$ and $x_2^{(k_2)}$ that belong to different regions of attraction, where function is of the form (3.10).

Figures 3.4 - 3.6 and Table 3.13 show the magnitude of c_1 and c_2 increases as a function of β. Recall that if (3.20) is positive, then inequality (3.17) will fail, which is desirable for two points belonging to different regions of attraction. For functions of the form (3.9) and (3.10), large β has a great effect on the magnitude of $|c_1/c_2|$,

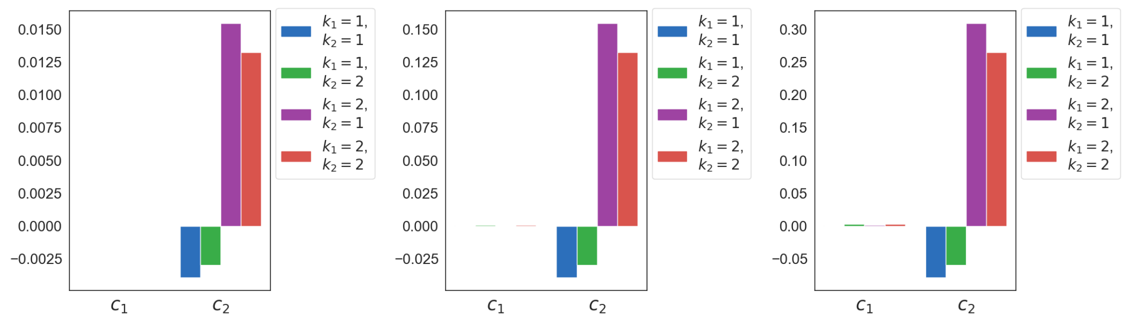


Figure 3.6: Components c_1 and c_2 calculated for $\beta = 0.01$ (left), $\beta = 0.1$ (centre) and $\beta = 0.2$ (right) with $x_1^{(k_1)}$ and $x_2^{(k_2)}$ that belong to different regions of attraction, where function is of the form (3.11).

Function	β	$k_1 = 1$, $k_2 = 1$	$k_1 = 1$, $k_2 = 2$	$k_1 = 2$, $k_2 = 1$	$k_1 = 2$, $k_2 = 2$
Minimum of several Quadratic forms (3.9)	0.01	0.051	0.0506	0.1741	0.023
	0.1	0.5102	0.5063	1.7413	0.23
	0.2	1.0203	1.0125	3.4827	0.46
Sum of Gaussians (3.10)	0.01	0.0260	0.0051	0.0152	0.0557
	0.1	0.2605	0.0506	0.1517	0.5571
	0.2	0.5210	0.1013	0.3034	1.1143
Modified Shekel (3.11)	0.01	0.0003	0.0027	0.0003	0.0006
	0.1	0.0029	0.0271	0.0025	0.0056
	0.2	0.0058	0.0543	0.0051	0.0112

Table 3.13: Computation of $|c_1/c_2|$ for points $x_1^{(k_1)}$ and $x_2^{(k_2)}$ that belong to different regions of attraction, for $\beta = 0.01, 0.1, 0.2$.

which is useful to observe when $c_2 < 0$. This is because if the magnitude of $|c_1/c_2|$ grows and $c_2 < 0$, then the likelihood of (3.20) being greater than zero increases, which will result in inequality (3.17) failing. This can be observed in Figure 3.4, where inequality (3.17) fails for $\beta = 0.2$, since the magnitude of c_1 is greater than c_2 for the combinations; $(k_1 = 1,\ k_2 = 1)$, $(k_1 = 1,\ k_2 = 2)$, and $(k_1 = 2,\ k_2 = 1)$. However, this is not the case for $\beta = 0.01, 0.1$ in Figure 3.4 as inequality (3.17) may hold for different combinations of k_1 and k_2 since the magnitude of c_1 is less than the magnitude of c_2.

For functions of the form (3.11), large β has a very small effect on the magnitude of c_1. This is also highlighted in Table 3.13, where $|c_1/c_2|$ remains very small for all values of β, suggesting that (3.20) is effected by c_2 only. Hence, increasing β will have a limited impact on the number of violations of (3.17) for points belonging to different regions of attraction when functions are of the form (3.11). This can also be observed in Table 3.8(a). Nevertheless, Tables 3.6 - 3.8 show that the number of violations of (3.17) is largest when $\beta = 0.2$ for functions of the form (3.9) - (3.11). Consequently, it is recommended to set $\beta = 2/\lambda_{max}$ when applying the METOD algorithm for functions with known λ_{max}.

3.6.3 Heuristic approximation of β

It can be observed in the previous subsection that setting $\beta = 2/\lambda_{max}$ can improve the accuracy of the METOD algorithm. Recall that λ_{max} is unknown for objective functions (3.12) - (3.15). Therefore, it can be difficult to determine an upper bound for the value of β.

Suppose starting points $x_n^{(0)}$ $(n = 1, \ldots, N)$ are sampled uniformly at random from $\mathfrak{X}$. The reciprocal of the average norm of the gradient,

$$\overline{\|\nabla f(x_n^{(0)})\|}^{-1} = \left(\frac{1}{N} \sum_{n=1}^{N} ||\nabla f(x_n^{(0)})|| \right)^{-1} \tag{3.21}$$

will be investigated as a metric to determine an approximation of β.

Figure 3.7 shows the average of (3.21) for 100 functions of the form (3.9) with $P = 5$ and various λ_{max} and d, where starting points $x_n^{(0)}$ $(n = 1, \ldots, 30)$ are sampled uniformly at random from $\mathfrak{X}$ for each function to compute (3.21). Figure 3.7 illustrates that (3.21) is largest for small d and λ_{max}, and (3.21) decreases as d grows. It is recommended in Subsection 3.6.2 to set $\beta = 2/\lambda_{max}$ to increase the number of violations of (3.17) for points belonging to different regions of attraction. To derive an approximation of β, the values of (3.21) will need to be reduced for small d and increased for large d in order to approximate $2/\lambda_{max}$ for various λ_{max}. Therefore, consider the following approximation of β,

$$\beta \approx \frac{(d-1)^2}{cd\sqrt{d}} \left(\frac{1}{N} \sum_{n=1}^{N} ||\nabla f(x_n^{(0)})|| \right)^{-1}, \tag{3.22}$$

where $c = 2.25$. Figure 3.7 shows the average approximation of β (3.22) for 100 functions of the form (3.9) with $P = 5$ and various λ_{max} and d, where starting points $x_n^{(0)}$ $(n = 1, \ldots, 30)$ are sampled uniformly at random from $\mathfrak{X}$ for each function to compute (3.22). It can be observed that (3.22) approximates $\beta = 2/\lambda_{max}$ generally well for functions of the form (3.9) with $P = 5$, and different values of d and λ_{max}.

In order to apply (3.22) to approximate a value for β, the magnitude of the gradient at various starting points $||\nabla f(x_n^{(0)})||$ $(n = 1, \ldots, N)$ should be reasonably large to ensure that $x_n^{(0)}$ is within the vicinity of a region of attraction of a local minimizer. If $||\nabla f(x_n^{(0)})||$ is small, then the approximation of β in (3.22) will be very

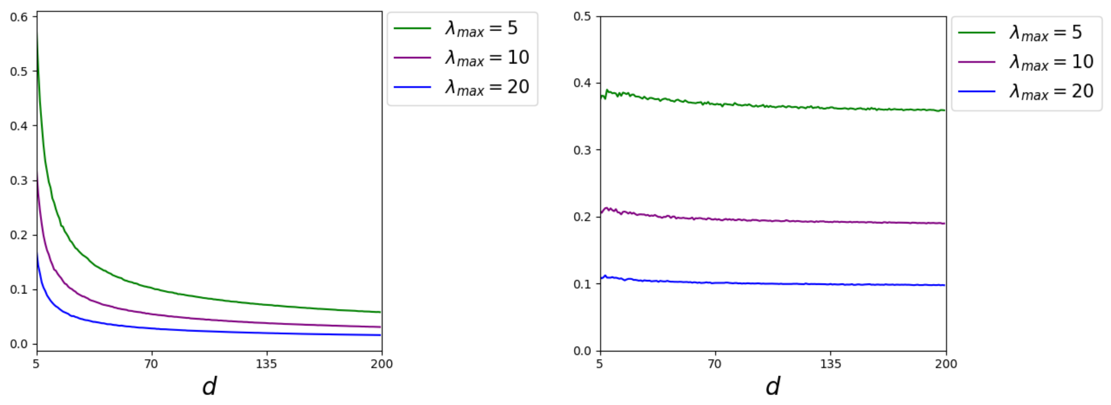


Figure 3.7: Average value of (3.21) (left) and average approximation of β (3.22) (right) for 100 functions (3.9) with $P = 5$, various d and λ_{max}, with starting points $x_n^{(0)}$ $(n = 1, \ldots, 30)$ sampled uniformly at random from $\mathfrak{X}$ for each function to compute (3.21) and (3.22), respectively.

large and may have an adverse effect on the accuracy and efficiency of the METOD algorithm.

To ensure the approximation of β in (3.22) is suitable for functions with a different number of local minima P, 100 functions of the form (3.9) will be generated with various P, d and λ_{max}. For each function (3.9), starting points $x_n^{(0)}$ $(n = 1, \ldots, 100)$ are sampled uniformly at random from $\mathfrak{X}$. It will be investigated whether (3.22) approximates $\beta = 2/\lambda_{max}$ for $\lambda_{max} = 5, 10, 20$, with various d and P.

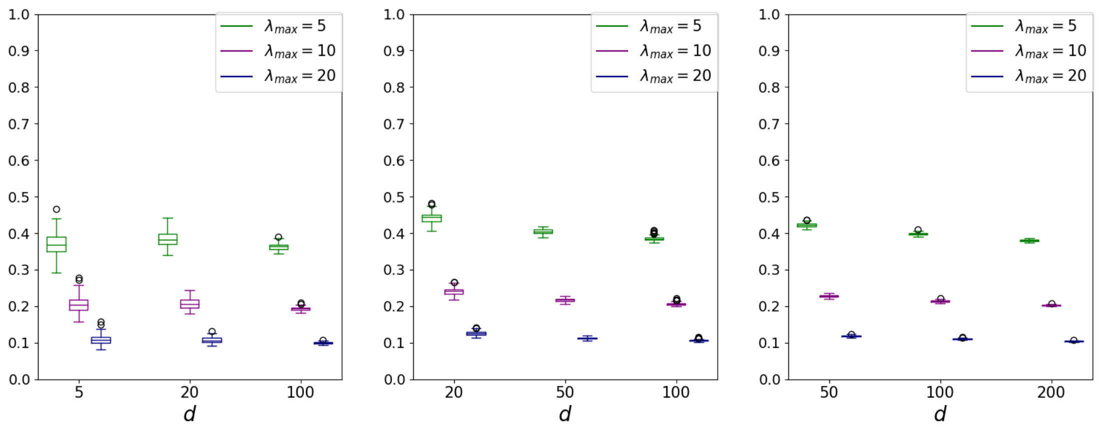


Figure 3.8: Boxplots of the approximation of β in (3.22) for each random function (3.9) with $P = 5$ (left), $P = 20$ (centre) and $P = 50$ (right) for various d and λ_{max}.

Figure 3.8 shows (3.22) approximates $\beta = 2/\lambda_{max}$ generally well. When $P = d$, the approximation (3.22) can be slightly lower than $\beta = 2/\lambda_{max}$ for small d and slightly higher than $\beta = 2/\lambda_{max}$ for large d. Nevertheless, (3.22) can be used as a guide to generate a suitable estimate of β. In addition, if the gradient of the objective function is unknown, then an approximation of the gradient may be used to approximate β in (3.22). The approximation of the gradient may be computed, for example, by applying the finite difference method.

For objective functions (3.13) - (3.15), Table 3.4 shows that average $\|\nabla f(x_n^{(0)})\|$ is relatively large. Therefore, (3.22) will be used to approximate values of β for functions (3.13) - (3.15). For numerical examples, 100 functions (3.13) - (3.15) will be generated along with starting points $x_n^{(0)}$ $(n = 1, \dots, 100)$ sampled uniformly at random from $\mathfrak{X}$.

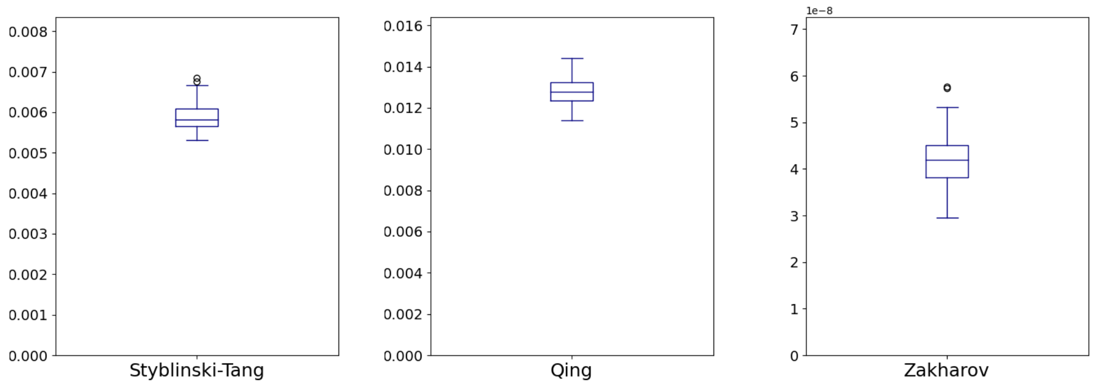


Figure 3.9: Approximation of β using (3.22) for functions (3.13) - (3.15). It should be noted that the values on the y axis are different for each function and for the Zakharov function (3.15), the y axis is scaled to e^{-8}.

It can be observed in Figure 3.9 that the medians of the approximation of β using (3.22) for functions (3.13) - (3.15) are close to values of β in Tables 3.10 - 3.12. The corresponding values of β for functions (3.13) and (3.14) produced a high number of violations for points belonging to different regions of attractions and a low number of violations for points belonging to the same region of attraction. Furthermore, the median of the approximation of β using (3.22) for function (3.15) is close to the optimal choice of β which produced no violations for points belonging to the same region of attraction. Tables 3.10 and 3.11 show that a larger value of β may improve accuracy and efficiency of the METOD algorithm. Nevertheless, since λ_{max} is unknown for functions (3.13) - (3.15), it can be extremely useful to apply (3.22) to approximate a value for β.

If $\|\nabla f(x_n^{(0)})\|$ is small for a function with unknown λ_{max}, (3.22) cannot be used to approximate a value of β. This is the case for function (3.12), where average $\|\nabla f(x_n^{(0)})\|$ is relatively small in Table 3.4. Therefore, it is recommended to assign a small value to β (e.g. $\beta = 0.001$ or $\beta = 0.01$), which is consistent with the advice in [144].

3.7 METOD algorithm numerical examples

3.7.1 Overview

For numerical examples, 100 functions f_j $(j = 1, \dots, 100)$ of the form (3.9) - (3.14) will be generated along with N starting points $x_n^{(0)}$ $(n = 1, \dots, N)$ sampled uniformly at random from $\mathfrak{X}$. For each function, the METOD algorithm will be applied with $x_n^{(0)}$ $(n = 1, \dots, N)$. In order to compare the results of the METOD algorithm, multistart will also be applied to the same set of starting points $x_n^{(0)}$ $(n = 1, \dots, N)$. That is, local descent will be applied to each $x_n^{(0)}$ $(n = 1, \dots, N)$ until the smallest $K_n = k$ is found such that $\|\nabla f(x_n^{(k)})\| < \delta$. Values of δ for each function (3.9) - (3.14) can be found in Table 3.4. The choice of dimension d, total number of local minima of each objective function and total number of starting points N for functions (3.9) - (3.14) are presented in Table 3.14.

Function	d	N	Number of local minima
Minimum of several quadratic forms (3.9)	100	1000	50
Sum of Gaussians (3.10)	20	100	10
Modified Shekel (3.11)	4	100	10
Hartmann 6 (3.12)	6	50	2
Styblinski-Tang (3.13)	5	1000	32
Qing (3.14)	5	1000	32

Table 3.14: Choice of dimension d, total number of local minima of each objective function and total number of starting points N for functions (3.9) - (3.14).

Recall that application of the following condition (3.18) determines whether local descents are terminated early within the METOD algorithm,

$$\|\tilde{x}_n^{(M)} - \tilde{x}_l^{(i)}\| \leq \|x_n^{(M)} - x_l^{(i)}\| \text{ and } \|\tilde{x}_n^{(M-1)} - \tilde{x}_l^{(i)}\| \leq \|x_n^{(M-1)} - x_l^{(i)}\|,$$

where l is the index for a local minimizer and $i = M-1, \dots, K_l$. Condition (3.18) is based entirety on testing inequality (3.17) for various points. In Section 3.5, the frequency of violations of inequality (3.17) is investigated for points belonging to the same and different regions of attraction for functions (3.9) - (3.15). In addition, only several steepest descent iterations (3.3) have been applied to points evaluated with inequality (3.17) in Section 3.5, which is equivalent to testing only a subset of inequalities of (3.18) used by the METOD algorithm. Evaluating a subset of inequalities of (3.18) with small M and i can significantly influence whether all inequalities (3.18) hold with $i = M - 1, \dots, K_l$. Therefore, results of numerical examples will be compared with the results observed in Tables 3.6 - 3.11 in Section 3.5 to explain key trends. Functions (3.15) are not considered for numerical examples since there is only one minimizer.

The results of numerical examples will be categorized into two parts. The first category presents results on the accuracy of the METOD algorithm compared to multistart. That is, the total number of local minimizers found by the METOD algorithm and multistart are compared for each f_j $(j = 1, \dots, 100)$ of the form (3.9) - (3.14). Since local descent is applied to each sampled point $x_n^{(0)} \in \mathfrak{X}$ $(n = 1, \dots, N)$ within multistart, each region of attraction corresponding to the local minimizer $x_n^{(K_n)}$ will be discovered. It may be possible that the METOD algorithm completely misses an opportunity to identify new local minimizers through condition (3.18) repeatedly holding for points belonging to undiscovered regions of attraction. Although new local minimizers may be missed in the early stages, it is still possible for the METOD algorithm to find all the same local minimizers as multistart provided that condition (3.18) eventually fails for points belonging to a new region of attraction.

The second category of results highlights the efficiency of the METOD algorithm. Specifically, the total number of gradient evaluations and local descents computed by METOD and multistart are compared. The efficiency of the METOD algorithm is reduced if condition (3.18) fails for two points that belong to the same region of attraction. Hence, if the proportion of local descents by METOD compared to multistart is large, then condition (3.18) fails for many points belonging to the same region of attraction.

All results for accuracy and efficiency of the METOD algorithm will be presented for various values of β and M.

3.7.2 Accuracy

Let f_{min} be the smallest function value among all local minima found by applying multistart with starting points $x_n^{(0)} \in \mathfrak{X}$ $(n = 1, \dots, N)$. It should be noted that f_{min} may not be the global minimum of a function f_j $(j = 1, \dots, 100)$. Consequently, if the global minimum of an objective function is not discovered by multistart, it will also not be discovered by METOD. Table 3.15 shows the total number of times f_{min} is found by the METOD algorithm for f_j $(j = 1, \dots, 100)$ of the form (3.9) - (3.14). All minima of functions (3.9) and (3.14) are global, and as a consequence, f_{min} will be the same for each discovered local minimizer. Therefore, the total number of local minimizers found by applying the METOD algorithm and multistart are also compared for each f_j $(j = 1, \dots, 100)$ of the form (3.9) - (3.14).

Let $L^{(j)}_{_{METOD}}$ and $L^{(j)}_{_{Mult}}$ be the total number of local minimizers found by METOD and multistart respectively for each objective function f_j $(j = 1, \dots, 100)$ of the form (3.9) - (3.14). The percentage of local minimizers (PLM) and average number of local minimizers (ALM) missed by the METOD algorithm are computed as follows

$$\text{PLM} = \frac{\sum_{j=1}^{100}(L^{(j)}_{_{Mult}} - L^{(j)}_{_{METOD}})}{\sum_{j=1}^{100} L^{(j)}_{_{Mult}}} \times 100 \tag{3.23}$$

and

$$\text{ALM} = \frac{1}{100}\sum_{j=1}^{100}(L^{(j)}_{_{Mult}} - L^{(j)}_{_{METOD}}). \tag{3.24}$$

Table 3.16 shows the percentage of local minimizers (PLM) missed by METOD (3.23) and average number of local minimizers (ALM) missed by METOD (3.24) for functions (3.9), (3.10), (3.11) and (3.14). Results for functions (3.12) and (3.13) have been omitted in Table 3.16 since all local minimizers are found by METOD in Table 3.15 for various M and β.

Tables 3.15 and 3.16 show that the number of local minimizers found by the METOD algorithm compared to multistart is greatest when $\beta = 0.2$ for functions (3.9) and (3.10). It can be shown in Tables 3.6 and 3.7 that selecting $\beta = 0.2$ minimizes the risk of the METOD algorithm assigning points to the wrong region of attraction, and consequently, there is a greater opportunity of discovering new local minimizers.

Tables 3.15 and 3.16 highlight that the selection of β for functions of the form (3.11) has a limited effect on the total number of local minimizers found by the METOD algorithm. A similar effect can also be observed in Table 3.8, where larger values of β have a limited impact on the probability of the METOD algorithm assigning points to the wrong region of attraction. Furthermore, Table 3.15 shows that the METOD algorithm may not find as many local minimizers as multistart for some functions of the form (3.11). In fact, results in Tables 3.15 and 3.16 are poorer for functions (3.11) in comparison to all other classes of functions applied with the METOD algorithm for $M = 2, 3$. This is due to the narrow regions of attraction of local minimizers. Nevertheless, Table 3.15 demonstrates that selecting $M = 3$ ensures that the METOD algorithm finds the same number of local minimizers as multistart for a large number of functions of form (3.11). Also, selecting $\beta = 0.2$ results in reduced values for (3.23) and (3.24) in Table 3.16. Hence, setting $\beta = 2/\lambda_{max}$ is recommendable.

Table 3.15 shows the METOD algorithm finds all the same local minimizers as multistart for 100 functions of the form (3.12) and (3.13) with all values of β and M.

Function	M	β	Number of functions				
			Local minimizers missed by METOD				METOD finds f_{min}
			0	1	2	≥ 3	
Minimum of several quadratic forms (3.9)	1	0.01	0	0	0	100	100
		0.1	0	0	0	100	100
		0.2	95	5	0	0	100
	2	0.01	91	8	1	0	100
		0.1	100	0	0	0	100
		0.2	100	0	0	0	100
	3	0.01	95	4	1	0	100
		0.1	100	0	0	0	100
		0.2	100	0	0	0	100
Sum of Gaussians (3.10)	2	0.01	92	8	0	0	99
		0.1	96	4	0	0	99
		0.2	97	3	0	0	100
	3	0.01	92	8	0	0	99
		0.1	95	5	0	0	99
		0.2	97	3	0	0	100
Modified Shekel (3.11)	2	0.01	73	22	5	0	100
		0.1	73	22	5	0	100
		0.2	74	21	5	0	100
	3	0.01	78	14	8	0	100
		0.1	78	14	8	0	100
		0.2	78	15	7	0	100
Hartmann 6 (3.12)	2	0.001	100	0	0	0	100
		0.01	100	0	0	0	100
	3	0.001	100	0	0	0	100
		0.01	100	0	0	0	100
Styblinski-Tang (3.13)	1	0.001	100	0	0	0	100
		0.005	100	0	0	0	100
		0.01	100	0	0	0	100
	2	0.001	100	0	0	0	100
		0.005	100	0	0	0	100
		0.01	100	0	0	0	100
Qing (3.14)	1	0.005	25	23	25	27	100
		0.01	27	28	17	28	100
		0.05	77	21	2	0	100
	2	0.005	91	9	0	0	100
		0.01	93	7	0	0	100
		0.05	98	2	0	0	100
	3	0.005	100	0	0	0	100
		0.01	100	0	0	0	100
		0.05	100	0	0	0	100

Table 3.15: Number of functions f_j ($j = 1, \ldots, 100$) of the form (3.9) - (3.14) in which the METOD algorithm finds 0, 1, 2 or ≥ 3 less local minimizer(s) than multistart and the number of functions in which the METOD algorithm finds the same f_{min} as multistart with various M and β.

Function	M	β	PLM (3.23)	ALM (3.24)
Minimum of several quadratic forms (3.9)	1	0.01	90.67%	43.42
		0.1	78.58%	37.63
		0.2	0.10%	0.05
	2	0.01	0.21%	0.10
		0.1	0%	0
		0.2	0%	0
	3	0.01	0.13%	0.06
		0.1	0%	0
		0.2	0%	0
Sum of Gaussians (3.10)	2	0.01	0.83%	0.08
		0.1	0.42%	0.04
		0.2	0.31%	0.03
	3	0.01	0.83%	0.08
		0.1	0.52%	0.05
		0.2	0.31%	0.03
Modified Shekel (3.11)	2	0.01	3.45%	0.32
		0.1	3.45%	0.32
		0.2	3.34%	0.31
	3	0.01	3.24%	0.30
		0.1	3.24%	0.30
		0.2	3.13%	0.29
Qing (3.14)	1	0.005	5.31%	1.70
		0.01	4.97%	1.59
		0.05	0.78%	0.25
	2	0.005	0.28%	0.09
		0.01	0.22%	0.07
		0.05	0.06%	0.02
	3	0.005	0%	0
		0.01	0%	0
		0.05	0%	0

Table 3.16: Percentage of local minimizers (PLM) computed in (3.23) and average number of local minimizers (ALM) computed in (3.24) missed by the METOD algorithm for functions (3.9), (3.10), (3.11) and (3.14).

For functions of the form (3.14), Table 3.15 demonstrates that a warm-up period of $M = 1$ may result in some local minimizers being missed by the METOD algorithm. However, selecting $\beta = 0.05$ increases the number of functions in which the METOD algorithm finds the same number of local minimizers as multistart. Furthermore, the METOD algorithm finds the same number of local minimizers as multistart with all 100 functions (3.14) for $M = 3$ and all values of β.

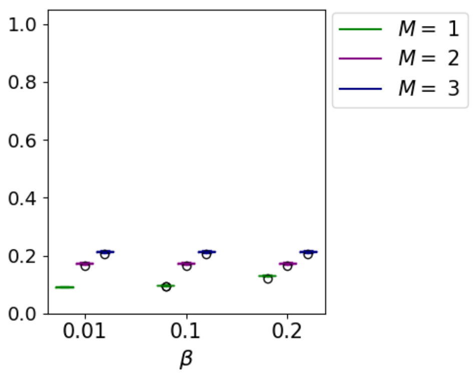
For all functions (3.9) - (3.14), Tables 3.15 and 3.16 show that increasing the warm-up period M will either maintain or improve accuracy of the METOD algorithm if β is chosen appropriately. Therefore, all results in Tables 3.15 and 3.16 demonstrate that if β and M are suitably chosen, the accuracy of the METOD algorithm will be high for a variety of different test functions.

3.7.3 Efficiency

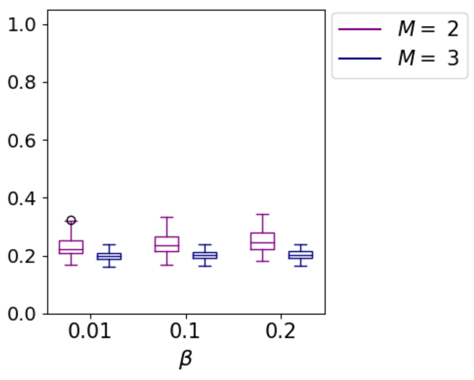
Figure 3.10 shows boxplots of the proportion of gradient evaluations by the METOD algorithm compared to multistart. Also, Figure 3.11 shows boxplots of the proportion of local descents by the METOD algorithm compared to multistart. Figures 3.10 and 3.11 display results for 100 functions of the form (3.9) - (3.14), each with a set of starting points $x_n^{(0)} \in \mathfrak{X}$ $(n = 1, \dots, N)$ for various M and β.

In general, the proportion of gradient evaluations in Figure 3.10 will vary for different classes of functions. If few steepest iterations (3.3) are applied to find a local minimizer $x_n^{(K_n)}$ $(n = 1, \dots, N)$ in multistart, then the proportion of gradient evaluations may be large. This is due to the METOD algorithm applying a warm-up period of M steepest descent iterations to obtain $x_n^{(M-1)}$ and $x_n^{(M)}$ $(n = 1, \dots, N)$ to evaluate condition (3.18). However, if many steepest iterations (3.3) are applied to find a local minimizer $x_n^{(K_n)}$ $(n = 1, \dots, N)$ in multistart, then the proportion of gradient evaluations will be small. In some cases, increasing the warm-up period M may lead to an increasing proportion of gradient evaluations since more steepest descent iterations (3.3) are computed. On the other hand, increasing the warm-up period M may reduce repeated local descents to the same local minimizer for some functions. As a consequence, the proportion of gradient evaluations may decrease as M increases.

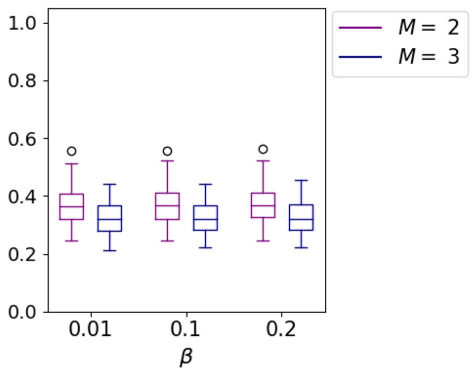
Figures 3.10(a) and 3.11(a) show that for functions of the form (3.9), the proportion of gradient evaluations and local descents is smaller for $\beta = 0.01, 0.1$ in comparison to $\beta = 0.2$ when $M = 1$. This is due to a very small proportion of violations in Table 3.6 for points belonging to different regions of attraction when $M = 1$ and $\beta = 0.01, 0.1$, which suggests a large number of local minimizers will be missed by the METOD algorithm. On the other hand, a large proportion of violations for



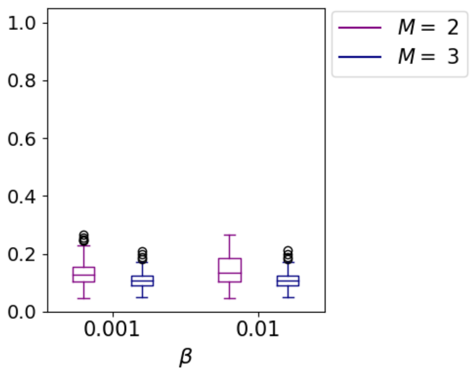
(a) Several quadratic forms function (3.9).



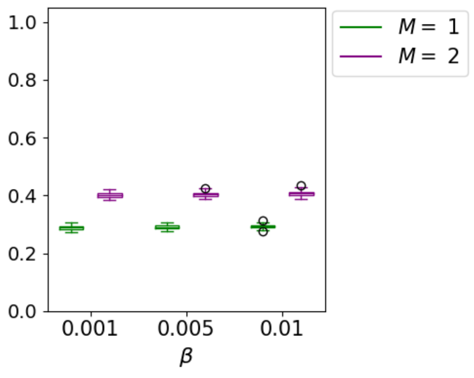
(b) Sum of Gaussians function (3.10).



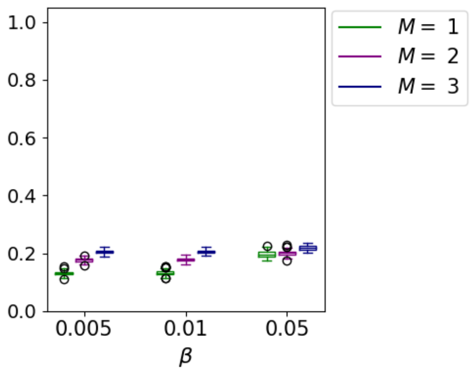
(c) Modified Shekel function (3.11).



(d) Hartmann 6 function (3.12).



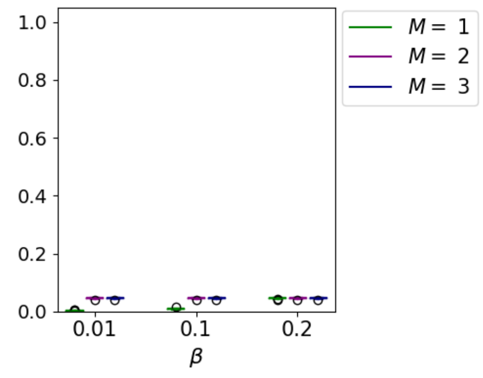
(e) Styblinski-Tang function (3.13).



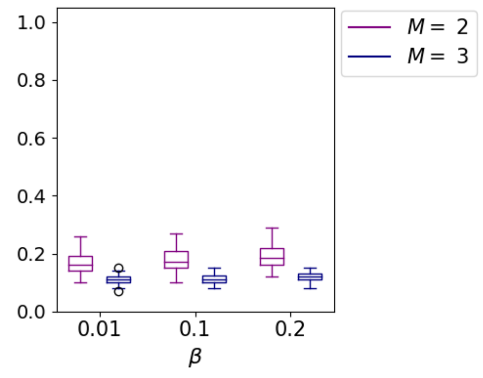
(f) Qing function (3.14).

Figure 3.10: Boxplots of the proportion of gradient evaluations used by the METOD algorithm compared to multistart for all 100 functions of the form (3.9) - (3.14), with various M and β.

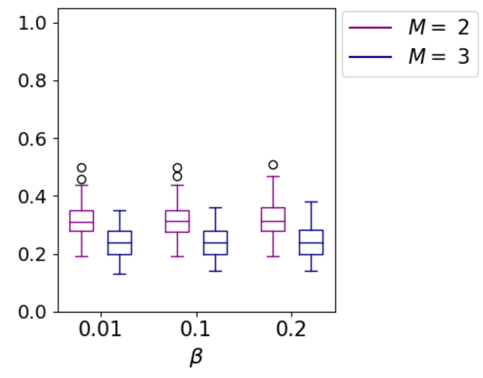
$M = 1$ and $\beta = 0.2$ can be observed in Table 3.6 for points belonging to different regions of attraction. Figure 3.10(a) shows that the proportion of gradient evaluations grows as M increases for functions of the form (3.9), which is due to two reasons. Firstly, the number of additional steepest descent iterations (3.3) grows as a function of M, which results in more gradient evaluations. Secondly, condition (3.18) holds for all points belonging to the same region of attraction with $M = 1, 2, 3$ and



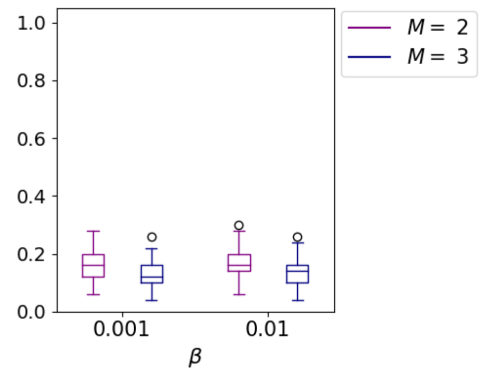
(a) Several quadratic forms function (3.9).



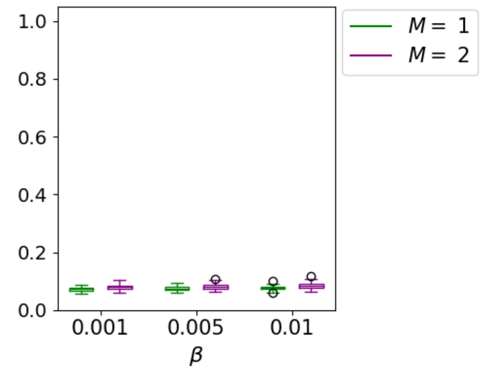
(b) Sum of Gaussians function (3.10).



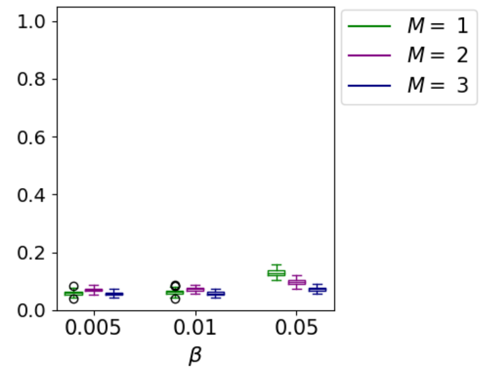
(c) Modified Shekel function (3.11).



(d) Hartmann 6 function (3.12).



(e) Styblinski-Tang function (3.13).



(f) Qing function (3.14).

Figure 3.11: Boxplots of the proportion of local descents by the METOD algorithm compared to multistart for all 100 functions of the form (3.9) - (3.14), with various M and β.

$\beta = 0.01, 0.1, 0.2$ for functions of the form (3.9). As a result, there are no repeated local descents to the same local minimizer for any combination of parameters β and M, which explains the low proportion of local descents for $M = 1$ and $\beta = 0.01, 0.1$, and the similar proportion of local descents for all other combinations of M and β in Figure 3.11(a).

For functions of the form (3.10) - (3.12), the proportion of gradient evaluations in Figures 3.10(b) - 3.10(d) and proportion of local descents in Figures 3.11(b) - 3.11(d) decreases as M increases. This is due to the decreasing proportion of violations for increasing M in Tables 3.7 - 3.9 with points belonging to the same region of attraction. Consequently, the number of repeated local descents to a known region of attraction is reduced.

The proportion of gradient evaluations in Figure 3.10(e) grows as M increases for functions (3.13) since additional steepest descent iterations (3.3) are computed within the METOD algorithm. Also, increasing M appears to marginally increase the proportion of local descents in Figure 3.11(e). This can be explained by a larger proportion of violations in Table 3.10 for increasing M with points belonging to the same region of attraction, resulting in an increased number of repeated local descents. The proportion of gradient evaluations in Figure 3.10(e) is relatively large since only few steepest descent iterations (3.3) are required to find a local minimizer $x_n^{(K_n)}$ $(n = 1, \dots, N)$ in multistart for functions (3.13), which can be observed in Table 3.4.

The key trends of Figures 3.10(f) and 3.11(f) vary depending on the value of β and M chosen. Therefore, results for different M will be discussed individually. For $M = 1$, Figures 3.10(f) and 3.11(f) show that the proportion of gradient evaluations and local descents is smaller for $\beta = 0.005, 0.01$ compared to $\beta = 0.05$. This is due to a smaller proportion of violations in Table 3.11 for points belonging to the same and different regions of attraction when $M = 1$ and $\beta = 0.005, 0.01$. Consequently, some local minimizers can be missed by the METOD algorithm with $\beta = 0.005, 0.01$. However, a smaller number of repeated local descents to the same local minimizer is also observed for $\beta = 0.005, 0.01$ compared to $\beta = 0.05$ with $M = 1$.

When $M = 2$, the proportion of local descents in Figure 3.11(f) is marginally larger than when $M = 1, 3$ for $\beta = 0.005, 0.01$. The same trend can be observed in Table 3.11, where the proportion of violations with points belonging to the same region of attraction for $\beta = 0.005, 0.01$ is largest when $M = 2$ compared to $M = 1, 3$. However, the same theme is not apparent for the proportion of gradient evaluations in Figure 3.10(f). That is, for $\beta = 0.005, 0.01$ the proportion of gradient evaluations appears to increase as a function of M. This is due to the additional steepest descent iterations (3.3) required for a larger warm-up period M. Furthermore, increasing M does not significantly reduce the proportion of local descents and consequently, the number of gradient evaluations is not reduced.

For $M = 3$ and all values of β, the METOD algorithm finds the same local minimizers as multistart for all f_j $(j = 1, \dots, 100)$ of the form (3.14). Furthermore, the proportion of gradient evaluations and local descents is largest when $\beta = 0.05$ in comparison to $\beta = 0.005, 0.01$ for corresponding $M = 1, 2, 3$. This is highlighted in

Table 3.11, where the number of violations for points belonging to the same region of attraction is largest for $\beta = 0.05$ compared to $\beta = 0.005, 0.01$ for all M. The proportion of gradient evaluations for $\beta = 0.05$ differs slightly for increasing M due to two conflicting objectives. That is, there is a trade-off between a lower number of gradient evaluations from fewer repeated local descents to the same local minimizer, and an increased number of steepest descent iterations (3.3) from larger values of M.

3.7.4 Summary

In numerical examples, the accuracy and efficiency of the METOD algorithm with various M and β have been compared with multistart for a variety of different test functions. In this subsection, the main conclusions on the choice of parameters M and β are presented for each class of function.

Recall that inequality (3.17) is an integral part of METOD and is applied with various points in the main condition (3.18) of the METOD algorithm to decide on terminating local descents early. For objective functions with locally quadratic behaviour close to the neighbourhoods of local minimizers, inequality (3.17) holds for two points that belong to the same region of attraction for $\beta \in (0, 2/\lambda_{max}]$. In Section 3.6, the values of β which improve the accuracy of the METOD algorithm are investigated.

For functions with known λ_{max}, it is recommended to set $\beta = 2/\lambda_{max}$. It can be observed in numerical examples that setting $\beta = 2/\lambda_{max}$ can improve the accuracy of the METOD algorithm for functions (3.9), (3.10) and (3.11), whilst ensuring efficiency is maintained. For functions (3.9), setting $M = 2$ and $\beta = 2/\lambda_{max}$ will improve accuracy and efficiency of the METOD algorithm. Setting $M = 3$ and $\beta = 2/\lambda_{max}$ for functions (3.10) and (3.11) will improve efficiency of the METOD algorithm by reducing the proportion of gradient evaluations and local descents. A larger warm-up period M is required for functions (3.10) and (3.11) since the magnitude of the gradient at various starting points in Table 3.4 can be small, suggesting that starting points are far away from a local minimizer and in-between regions of attraction.

For functions with unknown λ_{max}, two approaches may be taken to determine a value for β. If the magnitude of the gradient at several starting points is relatively large, then β may be approximated using (3.22). Otherwise, β can be set as a small positive constant (i.e. $\beta = 0.001, 0.01$), which is consistent with the advice provide in [144].

Table 3.4 shows that the magnitude of the gradient at a number of starting points is relatively small for functions of the form (3.12) and consequently $\beta = 0.001, 0.01$ is chosen for numerical examples. For all 100 functions (3.12), the METOD algorithm finds the same number of local minimizers as multistart for $M = 2, 3$ and $\beta = 0.001, 0.01$. Furthermore, selecting $M = 3$ will improve the efficiency of the METOD algorithm since the proportion of gradient evaluations and local descents in Figures 3.10(d) and 3.11(d) are smaller in comparison to $M = 2$.

For functions (3.13) and (3.14), the magnitude of the gradient at a number of starting points is relatively large in Table 3.4. Therefore, β can either be assigned a small value or approximated using (3.22). For all 100 functions (3.13), the METOD algorithm finds the same number of local minimizers as multistart for $M = 1, 2$ and $\beta = 0.001, 0.005, 0.01$. Also, the METOD algorithm is most efficient when $M = 1$, since only a moderate number of steepest descent iterations (3.3) are required to find a local minimizer for functions of the form (3.13). Hence, small M enhances the efficiency of the METOD algorithm with functions of the form (3.13). Furthermore, the median of the approximation of β using (3.22) in Figure 3.9 is 0.0058. High accuracy and efficiency of the METOD algorithm can be observed for $\beta = 0.005$ with functions (3.13). In this case, (3.22) is beneficial in approximating β.

The accuracy of the METOD algorithm for functions (3.14) improves for large values of M. When $M = 2$, selecting $\beta = 0.05$ produces the largest number of functions in which METOD finds the same local minimizers as multistart. When $M = 3$, all values of β result in the METOD algorithm finding the same number of local minimizers as multistart for all 100 functions (3.14). The efficiency of the METOD algorithm improves marginally for smaller values of β at corresponding values of $M = 1, 2, 3$, since the proportion of gradient evaluations and local descents are slightly smaller in Figures 3.10(f) and 3.11(f) for decreasing values of β. Although different values of β are applied with the METOD algorithm for numerical examples, this would not be done typically in practice since numerous runs of the METOD algorithm would be computationally expensive. The median of the approximation of β using (3.22) in Figure 3.9 is 0.01277, which underestimates the promising value of $\beta = 0.05$ that has proven the most successful in improving the accuracy of the METOD algorithm for functions (3.14). However, suppose the approximation of β using (3.22) is applied for the METOD algorithm. In that case, it can be observed for $M = 2, 3$ that $\beta = 0.01$ results in a large number of functions in which METOD finds the same number of local minimizers as multistart. Hence, approximating β using (3.22) may improve accuracy and efficiency of the METOD algorithm as opposed to guessing a small value of β.

3.8 METOD algorithm in large dimensions

3.8.1 Overview

Recall that condition (3.18) is based on evaluating the fundamental inequality (3.17) with various points, and is essential for terminating local descents early within the METOD algorithm. The METOD algorithm has been applied with a variety of functions (3.9) - (3.14) in previous sections to provide general recommendations for the values of β and M. The purpose of this section is to analyze the impact of different d on the accuracy of condition (3.18) of the METOD algorithm. That is, the number of times condition (3.18) holds for points belonging to undiscovered regions of attraction and the effect this has on the total number of local minimizers discovered by the METOD algorithm.

Functions (3.9) and (3.10) can be used to investigate the performance of the METOD algorithm for different d since the function parameters can be set for any d. This is not the case for functions (3.11) - (3.14), where either parameters of the functions are predefined for certain values of d or the number of local minimizers increase exponentially as a function of d. To investigate the accuracy of the METOD algorithm for different d, 100 functions of the form (3.9) and (3.10) will be generated along with $N = 1000$ starting points $x_n^{(0)}$ $(n = 1, \dots, N)$ sampled uniformly at random from $\mathfrak{X} = [0,1]^d$. For functions (3.9), the different dimensions considered are $d = 100, 200$ and the number of local minimizers is $P = 50$. For functions (3.10), the number of local minimizers is $P = 10$ and the different dimensions considered are $d = 20, 50$ with $\sigma^2 = 0.7, 1.6$ respectively. Since $\lambda_{max} = 10$ for functions (3.9) and (3.10), $\beta = 2/\lambda_{max}$ will be used when applying the METOD algorithm (see Section 3.6).

The METOD algorithm and multistart will be applied to each function with a set of starting points $x_n^{(0)}$ $(n = 1, \dots, N)$. Multistart consists of applying steepest descent iterations (3.3) to each $x_n^{(0)}$ $(n = 1, \dots, N)$ until a local minimizer is discovered. Therefore, it is possible to check the accuracy of the METOD algorithm through comparison with multistart. Specifically, the local minimizers discovered by the METOD algorithm and mulitistart will be compared. Since the focus of analysis is on the accuracy of METOD with different d, a large number of starting points are generated to investigate the frequency of times inequality (3.18) holds for points belonging to undiscovered regions of attraction.

In [144, Sect. 5], accuracy is investigated by observing the total number of local minimizers discovered by METOD and the total number of local descents performed from condition (3.18) failing. Also investigated is the total percentage of points x_n

($n = 1, \dots, N$) that are assigned the wrong region of attraction, which is a consequence of incorrect classification in [144, Sect. 3.2]. The same investigations as [144, Sect. 5] are also presented in Tables 3.18 - 3.19 and 3.21 - 3.22, with the exception that the total percentage of misclassifications of points x_n ($n = 1, \dots, N$) assigned the wrong region of attraction is not considered. This is because it is unnecessary to assign a region of attraction to a point where local descent is terminated early. That is, the process of continuously checking condition (3.18) for all $l = 1, \dots, L$ can be computationally expensive when condition (3.18) has already been satisfied for one index l. As a result, only points that do not satisfy (3.18) will be assigned a region of attraction and points which satisfy (3.18) will be discarded. Consequently, the METOD algorithm will be able to find new regions of attraction more efficiently.

Results in Tables 3.18 - 3.19 and 3.21 - 3.22 are different to those in [144, Sect. 5] since the objective functions (3.9) and (3.10) are slightly different. Specifically, each objective function (3.9) and (3.10) is multiplied by a constant, whereas the objective functions considered in [144, Sect. 5] are not multiplied by a constant. Furthermore, different d and P are used for functions (3.9) and (3.10). In addition, $\beta = 0.2$ is applied with the METOD algorithm for functions (3.9) and (3.10) since $\beta = 2/\lambda_{max}$ is recommended in Section 3.6 for functions with known λ_{max}.

Tables 3.18 - 3.19 and 3.21 - 3.22 also display the total percentage of times a local minimizer is missed due to condition (3.18) holding for points belonging to undiscovered regions of attraction, which is not considered in [144, Sect. 5]. Although new local minimizers may be missed in the early stages, the METOD algorithm can still find all the same local minimizers as multistart provided that condition (3.18) eventually fails for points belonging to undiscovered regions of attraction. A consequence of condition (3.18) failing is that local descent is applied to the corresponding point and a new region of attraction of a local minimizer will be discovered. Hence, a non-zero percentage of local minimizers missed in Tables 3.18 - 3.19 and 3.21 - 3.22 does not necessarily indicate that a local minimizer is completely missed by the METOD algorithm.

3.8.2 Minimum of several quadratic forms function (3.9)

Table 3.17 shows the number of functions (3.9) in which 50, 49 or ≤ 48 local minimizers are discovered from applying local descent to each set of starting points x_n ($n = 1, \dots, 1000$), where $d = 100, 200$.

Tables 3.18 and 3.19 show that the number of local descents is equivalent to the number of local minimizers discovered by the METOD algorithm for all M and d. This illustrates that condition (3.18) always holds for points belonging to the same

d	Number of local minimizers discovered		
	50	49	≤ 48
100	34	32	34
200	16	32	52

Table 3.17: Number of functions (3.9) in which the number of local minimizers discovered is 50, 49 or $\leq$ 48 from applying local descent to each set of starting points x_n ($n = 1, \dots, 1000$) for each function.

β	M	Number of local minimizers			Number of local descents			Missed minimizers %
		50	49	≤ 48	50	49	≤ 48	
	1	32	33	35	32	33	35	0.327
0.2	2	34	32	34	34	32	34	0
	3	34	32	34	34	32	34	0

Table 3.18: Number of functions (3.9) in which the METOD algorithm finds 50, 49 or $\leq$ 48 local minimizers and performs 50, 49 or $\leq$ 48 local descents. Also, the total percentage of new local minimizers missed through condition (3.18) holding for points belonging to different regions of attraction. All results are for $P = 50$, $d = 100$, and various M and β.

β	M	Number of local minimizers			Number of local descents			Missed minimizers %
		50	49	≤ 48	50	49	≤ 48	
	1	16	32	52	16	32	52	0.007
0.2	2	16	32	52	16	32	52	0
	3	16	32	52	16	32	52	0

Table 3.19: Number of functions (3.9) in which the METOD algorithm finds 50, 49 or $\leq$ 48 local minimizers and performs 50, 49 or $\leq$ 48 local descents. Also, the total percentage of new local minimizers missed through condition (3.18) holding for points belonging to different regions of attraction. All results are for $P = 50$, $d = 200$, and various M and β.

region of attraction, and there are no repeated local descents to regions of attraction already identified.

Table 3.18 shows a smaller number of local minimizers are discovered by METOD compared to multistart when $M = 1$ and $d = 100$. However, this is not the case in Table 3.19 as METOD finds the same local minimizers as multistart for $M = 1$ and $d = 200$. In addition, Tables 3.18 and 3.19 show that the percentage of local minimizers missed is significantly reduced for larger d when $M = 1$. Hence, the accuracy of the METOD algorithm with small M improves as d increases. For $M \geq 2$, Tables 3.18 and 3.19 show that METOD finds the same local minimizers as multistart for different d.

3.8.3 Sum of Gaussians function (3.10)

Table 3.20 shows the number of functions (3.10) in which 10, 9 or ≤ 8 local minimizers are discovered from applying local descent to each set of starting points x_n $(n = 1, \dots, 1000)$, where $d = 20, 50$.

d	Number of local minimizers discovered		
	10	9	≤ 8
20	99	1	0
50	100	0	0

Table 3.20: Number of functions (3.10) in which the number of local minimizers discovered is 10, 9 or ≤ 8 from applying local descent to each set of starting points x_n $(n = 1, \dots, 1000)$ for each function.

β	M	Number of local minimizers			Average local descents	Missed minimizers %
		10	9	≤ 8		
0.2	2	99	1	0	99.92	0.033
	3	99	1	0	31.97	0.036
	4	99	1	0	15.66	0.020

Table 3.21: Number of functions (3.10) in which the METOD algorithm finds 10, 9 or ≤ 8 local minimizers and average number of local descents. Also, the total percentage of new local minimizers missed through condition (3.18) holding for points belonging to different regions of attraction. All results are for $P = 10$, $d = 20$, $\sigma^2 = 0.7$ and various M and β.

Tables 3.21 and 3.22 show that the METOD algorithm finds the same total number of local minimizers as multistart in Table 3.20 for all M and d. Furthermore, the average number of local descents decrease as a function of M. However, when M is small, the average number of local descents increases significantly for large d.

β	M	Number of local minimizers			Average local descents	Missed minimizers %
		10	9	≤ 8		
0.2	2	100	0	0	243.57	0.001
	3	100	0	0	45.64	0.001
	4	100	0	0	15.57	0

Table 3.22: Number of functions (3.10) in which the METOD algorithm finds 10, 9 or ≤ 8 local minimizers and average number of local descents. Also, the total percentage of new local minimizers missed through condition (3.18) holding for points belonging to different regions of attraction. All results are for $P = 10$, $d = 50$, $\sigma^2 = 1.6$ and various M and β.

This illustrates that it will be increasingly likely that many points drawn randomly from the feasible domain will not be within close vicinity of a region of attraction when d is large. Hence, the warm-up period M should be set to a larger value to ensure $x_n^{(M)}$ is within close vicinity of a region of attraction.

It can also be observed in Tables 3.21 and 3.22 that the percentage of missed local minimizers reduces as d increases. Although the percentage of local minimizers missed is non-zero, Tables 3.21 and 3.22 display that eventually condition (3.18) fails for a point belonging to a new region of attraction. This enables the METOD algorithm to identify the same local minimizers as multistart in Table 3.20 for each function (3.10).

3.8.4 Summary

Two key messages are portrayed from evaluating the accuracy of the METOD algorithm for different d within Subsections 3.8.2 and 3.8.3. Firstly, the accuracy of the METOD algorithm improves as d increases for functions of the form (3.9) when M is small, and for functions of the form (3.10) with all M.

Secondly, condition (3.18) may hold for several points belonging to undiscovered regions of attraction. However, the METOD algorithm can still find all the same local minimizers as multistart, provided that condition (3.18) eventually fails for points belonging to undiscovered regions of attraction. A consequence of condition (3.18) failing is that local descent is applied to the corresponding point, and a new region of attraction of a local minimizer will be discovered. If condition (3.18) holds for numerous points belonging to undiscovered regions of attraction, then the METOD algorithm is likely to completely miss local minimizers, as shown in Tables

3.18 and 3.19. Nevertheless, the number of times condition (3.18) holds for points belonging to undiscovered regions of attraction decreases as d and M increase.

3.9 Implementation and usage of the METOD algorithm in Python

3.9.1 Overview

Chapter 5 describes the process of implementing the METOD algorithm in Python, where details illustrating compliance with a variety of software development practices is provided. These include, ensuring that a version control system is used, software is readily accessible, all results can be reproduced, testing on all source code is conducted with 100% test coverage, continuous integration is employed and documentation is provided. In addition, the METOD algorithm has been made publicly available on GitHub, to enable other researchers to inspect and apply the algorithm for global optimization problems.

The purpose of this section is to provide details on the various input and output parameters of the METOD algorithm. In addition, instructions on how to install and apply the METOD algorithm are also provided. The input and output parameters of the METOD algorithm are displayed in Figure 3.12. The function (`f`), gradient (`g`), function arguments (`func_args`) and dimension (`d`) must be provided before applying the METOD algorithm.

```
>>> (discovered_minimizers,
...  number_minimizers,
...  func_vals_of_minimizers,
...  excessive_no_descents,
...  starting_points,
...  no_grad_evals) = metod_alg.metod(f, g, func_args, d,
...                                   num_points=1000, beta=0.01,
...                                   tolerance=0.00001, projection=False,
...                                   const=0.1, m=3, option='minimize_scalar',
...                                   met='Brent', initial_guess=0.005,
...                                   set_x='sobol', bounds_set_x=(0, 1),
...                                   relax_sd_it=1)
```

Figure 3.12: Example of calling the METOD algorithm in Python. To apply the METOD algorithm, the function (`f`), gradient (`g`), function arguments (`func_args`) and dimension (`d`) must be provided. Default values for the input parameters may be updated accordingly.

3.9.2 Input parameters

A range of METOD algorithm parameters have been discussed in [144, Sect. 3.3], along with recommendations on suitable choices for the parameters. All parameters discussed in [144, Sect. 3.3] form a subset of the required input parameters for the METOD algorithm in Python. Additional input parameters are also included to enhance the adaptability and usage of the METOD algorithm. The remainder of this subsection consists of several categories outlining each input parameter.

Objective function, gradient and dimension

The following input parameters in Figure 3.12 will need to be updated each time the METOD algorithm is applied.

- `f`: Objective function f applied with the METOD algorithm. The objective function will be evaluated at each $x_n^{(k)}$.
- `g`: Gradient of the objective function f, which is used to compute $\nabla f(x_n^{(k)})$ for steepest descent iterations (3.3).
- `func_args`: Function arguments required to compute $f(x_n^{(k)})$ and $\nabla f(x_n^{(k)})$.
- `d`: Size of dimension which is required to generate starting points $x_n^{(0)}$ and may also be used to generate function arguments.

The METOD algorithm can be applied by only updating `f`, `g`, `func_args` and `d` since the remaining input parameters are assigned default values. However, to improve the efficiency and accuracy of the METOD algorithm, it is strongly recommended to update values of the input parameters based on the chosen objective function, which will be discussed in the following categories.

Selection of starting points $x_n^{(0)}$ $(n = 1, \dots, N)$

Multistart consists of a global and local phase. For the global phase, typically starting points $x_n^{(0)} \in \mathfrak{X}$ $(n = 1, \dots, N)$ are sampled at random, which allows diverse exploration of the feasible domain and increases the possibility of finding the global minimizer. The METOD algorithm also incorporates the same global phase. The following input parameters in Figure 3.12 are utilized for the selection of starting points $x_n^{(0)}$ $(n = 1, \dots, N)$ within the METOD algorithm.

- `num_points`: The total number of starting points N generated before stopping the METOD algorithm.

- `set_x`: Starting points can either be generated uniformly at random from $\mathfrak{X}$ or by using a quasi-random sequence, such as Sobol sequence samples (see [118]). Sobol sequence samples will be transformed to ensure all samples are within $\mathfrak{X}$, and shuffled at random before applying the METOD algorithm. The Numpy library [47] is used to generate points uniformly at random and the SALib library [50] is used to generate Sobol sequence samples.
- `bounds_set_x`: The feasible domain $\mathfrak{X}$.

If the SALib library [50] is employed to generate Sobol sequence samples, an array is returned. Subsection 3.4.2 outlines a condition of the METOD algorithm which checks the adequacy of starting points $x_n^{(0)}$ $(n = 1, .., N)$. That is, if $\|\nabla f(x_n^{(0)})\| < \delta$ then a new starting point $x_n^{(0)} \in \mathfrak{X}$ is produced. Since all Sobol sequence samples are generated in advance of applying the METOD algorithm, $2N$ Sobol sequence samples will be generated as opposed to N, to ensure that starting points $x_n^{(0)}$ can be replaced if adequacy fails.

Partner point step size β

The partner point step size β in (3.5) is represented by the input parameter `beta` in Figure 3.12. It is extremely important that β is chosen appropriately in (3.5) since accuracy or efficiency of the METOD algorithm may be compromised. The effect of β on the accuracy and efficiency of the METOD algorithm with a variety of different test functions is studied in Section 3.6, and the following recommendations for the value of β are provided. For functions with known λ_{max}, it is recommended to set $\beta = 2/\lambda_{max}$. For functions with unknown λ_{max} and relatively large $\|\nabla f(x_n^{(0)})\|$ for each $n = 1, \ldots, N$, (3.22) may be used to approximate a value for β. Alternatively, β may be assigned a small positive value (i.e. $\beta = 0.001$ or $\beta = 0.01$).

Stopping criterion for steepest descent iterations (3.3)

Recall that steepest descent iterations (3.3) are terminated when (3.4) is satisfied. Also, recall in Subsection 3.4.2 that the adequacy of a starting point $x_n^{(0)}$ is checked by ensuring that $\|\nabla f(x_n^{(0)})\| \geq \delta$ for each $n = 1, \ldots, N$. The value of δ used to terminate steepest descent iterations (3.3) and to check adequacy of starting points is represented by the input parameter `tolerance` in Figure 3.12. It is recommended to assign δ a very small positive value (i.e. 0.00001 or 0.0001), to ensure steepest descent iterations (3.3) are not terminated too early when attempting to find a local minimizer. If steepest descent iterations (3.3) are terminated too early, saddle points may be found as opposed to local minimizers.

Projection of steepest descent iterations (3.3)

It is possible that an iteration of steepest descent $x_n^{(k)} \notin \mathfrak{X}$ for $k = 1, \dots, K_n - 1$. If this occurs, there is an option to project $x_n^{(k)} \in \mathfrak{X}$. That is, suppose $\mathfrak{X} = [a, b]^d$, then coordinates of $x_n^{(k)}$ which are less than a will be set to a. Alternatively, coordinates of $x_n^{(k)}$ which are greater than b will be set to b. It is not a concern if $x_n^{(k)} \notin \mathfrak{X}$, as long as the discovered local minimizer $x_n^{(K_n)} \in \mathfrak{X}$. The input parameter `projection` in Figure 3.12 represents the choice of projecting points $x_n^{(k)}$.

Selection of η in (3.19)

Recall that η is used within condition (3.19) to identify unique local minimizers found by the METOD algorithm. The value of η is represented by the input parameter `const` in Figure 3.12. If η is a large value, then two different local minimizers $x_1^{(K_1)}$ and $x_2^{(K_2)}$ may be classified as the same local minimizer by condition (3.19). Alternatively, if η is a very small value then two matching local minimizers $x_1^{(K_1)}$ and $x_2^{(K_2)}$, may be classified as different local minimizers by condition (3.19).

Warm up period M

The warm up period M is represented by the input parameter `m` in Figure 3.12. Since starting points may be located in between two regions of attraction or far away from a local minimizer, a warm up period of M steepest descent iterations (3.3) is applied to a starting point $x_n^{(0)}$ to obtain $x_n^{(M)}$. As a result, $x_n^{(M)}$ will be within close vicinity of a region of attraction and application of condition (3.18) can determine whether steepest descent iterations (3.3) should be terminated early. However, the efficiency of the METOD algorithm will be reduced if M is set to a large value, particularly if only a small or moderate number of steepest descent iterations (3.3) are required to find a local minimizer. Sections 3.5, 3.7 and 3.8 show that a warm up period of $M = 2$ or $M = 3$ yields promising results by the METOD algorithm.

Step length $\gamma_n^{(k)}$

Recall in Section 2.3 that for a steepest descent iteration, it is desirable to select a value for $\gamma_n^{(k)}$ such that the following minimization problem outlined in (2.5) is satisfied,

$$\gamma_n^{(k)} = \underset{\gamma > 0}{\operatorname{argmin}}\, f(x_n^{(k)} - \gamma \nabla f(x_n^{(k)})).$$

However, it can be computationally expensive to identify an exact value of $\gamma^{(k)}$ that satisfies (2.5). Therefore, a solution of (2.5) is often approximated and methods applied to determine $\gamma_n^{(k)}$ are discussed in Section 2.3.

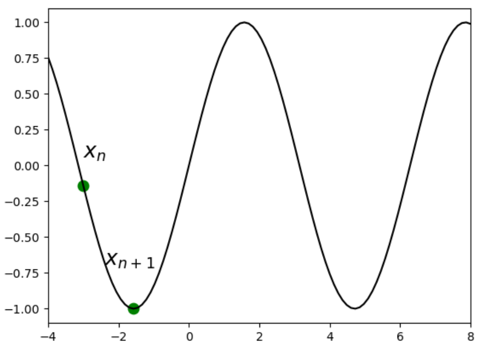
The METOD algorithm applies Brent's minimization method [14, Chpt 5.4] to approximate $\gamma_n^{(k)}$. Brent's minimization method has been implemented within the SciPy library [134] in Python. The SciPy library offers a wide range of optimization algorithms to find a local minimizer of a function. The `scipy.optimize.minimize_scalar` solver can be used to find a local minimizer of a univariate function. Alternatively, the `scipy.optimize.minimize` solver can be used to find a local minimizer of a univariate or multivariate function. Table 3.23 details each of the available solvers from the SciPy library to approximate $\gamma_n^{(k)}$.

Solver	**Features**
`scipy.optimize.minimize`	Requires an initial guess $\gamma > 0$ of $\gamma_n^{(k)}$ to initialize the solver. It must be ensured that γ is not too large, otherwise $x_n^{(k+1)}$ in (3.3) may overshoot the region of attraction that $x_n^{(k)}$ belongs to, which can be observed in Figure 3.13. Hence, it is recommended that a small value for γ is chosen (i.e. $\gamma = 0.005$ or $\gamma = 0.01$)
`scipy.optimize.minimize_scalar`	A bracket interval can be provided as an optional input parameter, which can either be of the form $[a, b]$ or $[a, b, c]$, where $a < b < c$ and $f(b) < f(a), f(c)$. If the bracket interval is of the form $[a, b]$, then the `scipy.optimize.bracket` function is executed to find an interval which brackets the minimum of a function and returns points $[a, b, c]$ such that $f(b) < f(a), f(c)$. Hence, the minimum of a function will lie in the interval $[a, b] \leftarrow [a, c]$ if $a < c$ or $[a, b] \leftarrow [c, a]$ if $a \geq c$. Figure 3.14 presents an application of `scipy.optimize.bracket`. To compute $\gamma_n^{(k)}$, it is recommended to set $a = 0$ and b as a small number (i.e. $b = 0.005$ or $b = 0.01$). This is to reduce the risk of $x_n^{(k+1)}$ in (3.3) overshooting the region of attraction that $x_n^{(k)}$ belongs to.

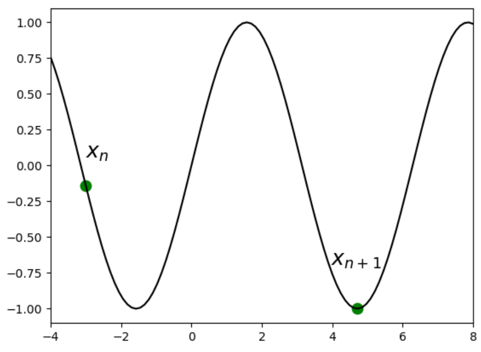
Table 3.23: Type of solver in SciPy [134] that can be used within the METOD algorithm to approximate $\gamma_n^{(k)}$.

The following input parameters represent the type of solver, method and additional parameters for each solver to compute $\gamma_n^{(k)}$.

- `option`: Option of solver in Python to compute $\gamma_n^{(k)}$ for steepest descent iterations (3.3). Can either choose `scipy.optimize.minimize_scalar` or

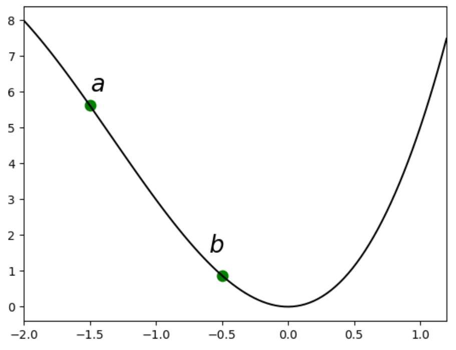


(a) Initial guess $\gamma = 0.005$

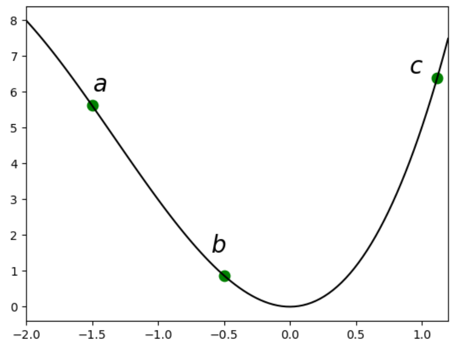


(b) Initial guess $\gamma = 5$

Figure 3.13: Effect of choosing different values for the initial guess γ in `scipy.optimize.minimize` to obtain $\gamma_n^{(k)}$ with function $f(x) = \sin(x)$.



(a) Initial guess of the interval $[a, b] = [-1.5, -0.5]$.



(b) Point c is returned by `scipy.optimize.bracket`.

Figure 3.14: Graphical representation of `scipy.optimize.bracket`, where $f(x) = 4x^2 + x^3$ on the interval $[-2, 1.2]$.

`scipy.optimize.minimize`, which is represented as `option='minimize_scalar'` or `option='minimize'` respectively.

- `met`: A method is required for `scipy.optimize.minimize_scalar` and `scipy.optimize.minimize`. For example, if `option='minimize_scalar'`, then Brent's minimization method can be used by setting `met='Brent'`.
- `initial_guess`: If `option='minimize'`, then `initial_guess` is the value of γ. Alternatively, if `option='minimize_scalar'`, then `initial_guess` is the value of the upper bound of the interval $[a, b]$.

Relaxed steepest descent

Relaxed steepest descent is described in [105] and involves multiplying the step length $\gamma_n^{(k)}$ by a small constant $r \in [0, 2]$, to obtain a new step length for steepest descent iterations (3.3). It is possible that multiplying $\gamma_n^{(k)}$ by r can improve the rate of convergence of steepest descent iterations [49]. The constant r is represented by the input parameter `relax_sd_it` in Figure 3.12.

Summary

All input parameters required by the METOD algorithm in Python are summarized in Table 3.24, along with details on whether an input parameter is compulsory to update for the application of the METOD algorithm. An optional input parameter is assigned a value which does not have to be updated. However, providing values for the optional input parameters will enhance the accuracy and efficiency of the METOD algorithm. In particular, values for `beta` and `m` should be updated according to the advice given in this chapter.

Input parameter	**Update**	**Type**	**Description**
`f`	Yes	function	Objective function evaluated at a point, which is a 1-D array with shape `(d,)`, and outputs a float.
`g`	Yes	function	Gradient evaluated at a point, which is a 1-D array with shape `(d,)`, and outputs a 1-D array with shape `(d,)`.
`func_args`	Yes	tuple	Extra arguments passed to `f` or `g`.
`d`	Yes	integer	Size of dimension.
`num_points`	Optional	integer	Number of starting points generated before stopping the METOD algorithm. The default is `num_points = 1000`.
`beta`	Optional	float	Small constant step size β to compute the partner points (3.5). The default is `beta = 0.01`.

`tolerance`	Optional	float or integer	Stopping condition for steepest descent iterations (3.3). That is, apply steepest descent iterations (3.3) until $\|\nabla f(x_n^{(k)})\| < \delta$, where the value of δ is represented by `tolerance`. Furthermore, if $\|\nabla f(x_n^{(0)})\| < \delta$, another starting point $x_n^{(0)}$ is used. The default is `tolerance = 0.00001`.
`projection`	Optional	boolean	If `projection = True`, then $x_n^{(k)}$ is projected into a feasible domain $\mathfrak{X}$. If `projection = False`, then $x_n^{(k)}$ is not projected. The default is `projection = False`.
`const`	Optional	float	Value of η used in (3.19). The default is `const = 0.1`.
`m`	Optional	integer	The number of steepest descent iterations (3.3) to apply to a starting point $x_n^{(0)}$ before making a decision on terminating descents. The default is `m = 3`.
`option`	Optional	string	Option of solver to compute $\gamma_n^{(k)}$ for steepest descent iterations (3.3). Choose from `"minimize"` or `"minimize_scalar"` (i.e. `scipy.optimize.minimize` or `scipy.optimize.minimize_scalar`). The default is `option = "minimize_scalar"`.
`met`	Optional	string	A method is required for `option = "minimize"` or `option = "minimize_scalar"`. The default is `met = "Brent"`.
`initial_guess`	Optional	float	Initial guess passed to `option = "minimize"` and the upper bound for the bracket interval when `option = "minimize_scalar"` for `met = "Brent"` and `met = "Golden"`. The default is `initial_guess = 0.005`.

`set_x`	Optional	string	If `set_x = "random"`, then starting points $x_n^{(0)} \in \mathfrak{X}$ are generated uniformly at random for the METOD algorithm. If `set_x = "sobol"`, then a 2-D array of Sobol sequence samples, introduced in [118], are generated using the SALib library [50]. Sobol sequence samples are transformed so that samples are within $\mathfrak{X}$. The Sobol sequence samples are then shuffled at random and selected by the METOD algorithm. Default is `set_x = "sobol"`.
`bounds_set_x`	Optional	tuple	Feasible domain $\mathfrak{X}$. The Default is `bounds_set_x = (0, 1)`.
`relax_sd_it`	Optional	float or integer	Multiply the step length $\gamma_n^{(k)}$ by a constant in [0, 2], to obtain a new step size for steepest descent iterations (3.3). This process is known as relaxed steepest descent (see [105]). Default is `relax_sd_it = 1`.

Table 3.24: Summary of the input parameters for the METOD algorithm, as observed in Figure 3.12.

3.9.3 Output parameters

Table 3.25 summarizes the outputs of the METOD algorithm. The output parameters `excessive_descents` and `no_grad_evals` may be inspected to assess the efficiency of the METOD algorithm. It is desirable for `excessive_descents` to be a small value since this outlines the number of repeated local descents to a known local minimizer. Also, it is desirable that the majority of entries in `no_grad_evals` are labelled as $M + 1$, which implies that many local descents have been terminated early.

The starting points used within the METOD algorithm are returned to allow the comparison of METOD algorithm outputs with other global optimization algorithms.

Output	**Type**	**Description**
`unique_minimizers`	list	Each unique minimizer found.
`unique_number_of_minimizers`	integer	Total number of unique minimizers found.
`func_vals_of_minimizers`	list	Objective function evaluated at each discovered unique minimizer.
`excessive_descents`	integer	Number of repeated local descents to the same local minimizer (see (3.19) for more details).
`starting_points`	list	Starting points used by the METOD algorithm.
`no_grad_evals`	1-D array	Array containing the number of gradient evaluations computed for each point x_n $(n = 1, \dots, N)$.

Table 3.25: Summary of the output parameters for the METOD algorithm, as observed in Figure 3.12.

3.9.4 Quickstart example

Figure 3.15 provides instructions on how to install and test the METOD Algorithm via the command line. An application of the METOD algorithm with an objective

```
$ git clone https://github.com/Megscammell/METOD-Algorithm.git
$ cd METOD-Algorithm
$ python setup.py develop
$ pytest
```

Figure 3.15: Install and test the METOD algorithm.

function and gradient is presented in Figure 3.16. Also included within the example are checks to ensure that outputs from the METOD algorithm are of expected form. The purpose of each line of code in Figure 3.16 is discussed in more detail within Table 3.26.

3.10 Summary

This chapter presents the METOD algorithm [144], which can reduce the number of repeated local descents to the same local minimizer. The early termination of descents in METOD is achieved through repeatedly evaluating the fundamental inequality (3.17). A variety of enhancements to the METOD algorithm have been implemented to improve efficiency and accuracy. Furthermore, extensive analysis on inequality (3.17) has been conducted, along with an investigation on suitable values for the essential parameters β and M of the METOD algorithm. Numerical examples show the high efficiency and accuracy of the METOD algorithm compared to multistart with various test functions. Moreover, the accuracy of the METOD algorithm improves as the dimension increases.

Finally, the implementation of the METOD algorithm in Python is outlined, which allows other researchers to apply the software for global optimization problems. A thorough discussion on the input parameters of the METOD algorithm and recommended values is presented. Furthermore, all METOD algorithm outputs are described, and an example applying the METOD algorithm is displayed.

```
>>> import numpy as np
>>> import math
>>> import metod_alg as mt
>>>
>>> np.random.seed(90)
>>> def f(x, x0, A, rotation):
...     return 0.5 * (x - x0).T @ rotation.T @ A @ rotation @ (x - x0)
...
>>> def g(x, x0, A, rotation):
...     return rotation.T @ A @ rotation @ (x - x0)
...
>>> d = 2
>>> A = np.array([[1, 0],[0, 10]])
>>> theta = np.random.uniform(0, 2 * math.pi)
>>> rotation = np.array([[math.cos(theta), -math.sin(theta)],
...                      [math.sin(theta), math.cos(theta)]])
>>> x0 = np.array([0.5, 0.2])
>>>
>>> args = (x0, A, rotation)
>>> (discovered_minimizers,
...  number_minimizers,
...  func_vals_of_minimizers,
...  excessive_no_descents,
...  starting_points,
...  no_grad_evals) = mt.metod(f, g, args, d, num_points=10)
>>> assert(np.all(np.round(discovered_minimizers[0], 3) ==
...        np.array([0.500, 0.200])))
>>> assert(number_minimizers == 1)
>>> assert(np.round(func_vals_of_minimizers, 3) == 0)
>>> assert(excessive_no_descents == 0)
>>> assert(np.array(starting_points).shape == (10, d))
>>> assert(np.all(no_grad_evals[1:] == 4))
```

Figure 3.16: Application of the METOD algorithm.

Line number	**Purpose of each line of code in Figure 3.16**
1 - 3	Import the required libraries.
5	Initialize the pseudo-random number generator seed.
6 - 7	Define a function `f` to apply the METOD algorithm, where only one local minimizer is to be found.
9 - 10	Define the gradient `g`, which returns the gradient of `f`.
12	Set the dimension as `d = 2`.
13	Create the variable `A`, which is assigned a diagonal matrix.
14	Create the variable `theta`, which is assigned a value chosen uniformly at random from $[0, 2\pi]$.
15 - 16	Create the variable `rotation`, which is assigned a rotation matrix using `theta`.
17	Create the variable `x0`, which is the minimizer of `f`.
19	Set `x0`, `A` and `rotation` as objective function arguments. The function arguments are required to run `f` and `g`.
20 - 25	Run the METOD algorithm with `f`, `g`, `args`, `d` and optional input `num_points=10` to obtain the outputs; `discovered_minimizers`, `number_minimizers`, `func_vals_of_minimizers`, `excessive_no_descents`, `starting_points` and `no_grad_evals`.
26-32	Check outputs of the METOD algorithm. Specifically, check that one minimizer $x_1^{(K_1)}$ is found, the function value at the discovered minimizer is $f(x_1^{(K_1)}) = 0$, all local descents are terminated early after discovering $x_1^{(K_1)}$, ten starting points are generated before stopping the METOD algorithm and only 4 gradient evaluations are computed for nine starting points.

Table 3.26: Purpose of each line of code in Figure 3.16.

Chapter 4

The choice of direction in high-dimensional response surface optimization

This chapter is summarized as follows.

- Section 4.1 provides an overview of response surface methodology (RSM) for stochastic optimization problems. In addition, the scope of the chapter is presented, which is to propose a new search direction that can be used to improve the performance of RSM for high-dimensional problems.
- Section 4.2 presents the Box-Wilson (BW) algorithm, which is the most often used and cited RSM strategy. The main focus is on Phase I of the BW algorithm, where steepest descent iterations are employed to approach a neighbourhood of a minimizer. Section 4.2 also outlines the strategies proposed in the RSM-related literature to compute the search direction and step length for steepest descent iterations.
- Section 4.3 details the response functions employed to demonstrate the performance of Phase I of BW with different search directions for steepest descent iterations.
- In Section 4.4, a new search direction is proposed for Phase I of BW for high-dimensional problems.
- Section 4.5 presents two modifications to the least-squares estimator as the choice of search direction for Phase I of BW for high-dimensional problems. The purpose of the modifications is to enable comparisons with the new search direction presented in Section 4.4.
- Section 4.6 outlines the strategy used to compute the step length for steepest

descent iterations within Phase I of BW for numerical experiments in Section 4.7.

- Section 4.7 provides the results of numerical experiments for Phase I of BW for different search directions presented in Sections 4.4 and 4.5.

The new search direction presented in Section 4.4 has been implemented in Python and made publicly available on GitHub. Hence, all source code, tests and analysis results are readily available. Furthermore, Chapter 5 illustrates the measures taken to ensure that all software developed is accurate and reliable.

4.1 Introduction

Response surface methodology (RSM) is a collection of methods for approximating a minimizer x^* of a regression function $\eta(x)$ using a series of observations

$$y_j = \eta(z_j) + \varepsilon_j, \quad j = 1, 2, \dots, \tag{4.1}$$

where $z_1, z_2, \dots$ are the values of a d-dimensional predictor (input variables) $x \in \mathbb{R}^d$. Usually the function $\eta(x)$ is assumed to be unimodal and errors ε_j are assumed to be rather large but not necessarily independent nor even random.

Stochastic optimization (also known as simulation optimization) involves finding values of the input parameters to minimize the output of a stochastic function. Stochastic optimization has been discussed in [2, 19, 35, 36, 74, 129], where various methods to solve different stochastic optimization problems are outlined. Typically, unconstrained stochastic functions follow the form (4.1) and are black-box since the true analytical form of $\eta(x)$ is unknown. Hence, RSM is often applied to solve stochastic optimization problems. Other popular techniques used for stochastic optimization include stochastic approximation methods [61, 108] such as finite-difference stochastic approximation or simultaneous perturbation stochastic approximation (see Subsection 2.4 for more details). In addition, direct search methods such as the Nelder-Mead simplex method [91], and random search algorithms such as simulated annealing [63] are also applied to solve stochastic optimization problems.

The relationship between stochastic optimization and several machine learning methods are described in [2] and include active learning [22] and reinforcement learning [127]. Specifically, the process of active learning involves querying labels of additional observations when necessary. Therefore, improving the accuracy of the prediction model can be achieved with very few labelled observations, which reduces the computational cost, especially when the labelled observations are expensive to obtain. According to [2], the process of active learning is similar to stochastic optimization since a selection of points is chosen at each iteration of an algorithm to evaluate the response function. Reinforcement learning involves discovering the set of actions to perform to maximize some reward. A key challenge in reinforcement learning is balancing the trade-off between exploring the feasible domain for different sets of actions and exploiting what has been learnt so far [127]. This is a similar dilemma often encountered with stochastic optimization. That is, [2] outlines that a trade-off exists between exploring the feasible domain for suitable points to improve the response function value, and exploiting the knowledge gained so far on the true analytical form of $\eta(x)$.

Typically, stochastic optimization methods are selected depending on whether the input variables are continuous or discrete and whether local or global optimization is required. RSM is often applied to solve stochastic optimization problems with continuous input variables [2]. Furthermore, RSM embodies one of the following two forms. The first form of RSM is a sequential strategy for solving stochastic local optimization problems. This consists of iteratively exploring small subregions of the feasible domain to determine search directions toward subregions where improvement in the response function is observed. Once a subregion of the optimum is located, a higher-order model is fitted to approximate the optimal point. The second form of RSM is used for stochastic global optimization problems, and involves exploring the entire feasible domain and fitting a global approximation model (i.e. using a neural network or kriging model). Exploration of the global approximation is conducted during the optimization process [4]. Throughout this chapter, the focus is on the first form of RSM.

Various authors have applied RSM in a wide range of disciplines. In [137], a review of the various applications of RSM within the food industry is considered. Some applications considered in [137] are optimization of the conditions to maximize extraction yield, and optimization of the drying process to improve the quality of a product and minimize production costs. In [80], a tutorial detailing the use of experimental designs and RSM in energy applications is provided. Some examples in [80] include observing the effect of different experimental conditions on biodiesel production and on the performance of batteries and fuel cells. RSM has also been used for optimization in analytical chemistry [8] and to determine the optimal extraction conditions to enhance the yield and quality of plant materials [3]. Work in [78] uses design of experiments and RSM to screen and tune the hyperparameters of a machine learning algorithm. Furthermore, [78] provides a case study where the hyperparameters of the random forest algorithm have been tuned with a given dataset.

The most often used and cited RSM strategy is the so-called *Box-Wilson* (BW) algorithm (see [13] and [51]). This algorithm and its modifications are discussed in Section 4.2. The BW algorithm consists of two phases. On Phase I, a succession of moves toward the neighbourhood of x^* is performed. On Phase II, a locally quadratic model of the response function is assumed for estimating the location of x^*. Different modifications of the BW algorithm are distinguished by using different designs on Phase I and different rules for choosing the step length of the descents. What has never been challenged is the choice of descent direction in the BW algorithm, which is generally chosen to be the least-square estimator of the gradient of the response function at a current point. This chapter presents an alternative descent direction, which can significantly improve the first phase of BW and RSM, in general, for high-dimensional problems.

Consider the following notation that will be used throughout this chapter.

- $M = (m_{j,i})_{j,i=1}^{N,d}$, is a design matrix with N observations and d variables. Also, $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,d})^T$ is the j-th row of M;
- $m_{j,i} \in \{-1, +1\}$, each entry of M is either -1 or +1;
- M_+ has the following form;

$$M_+ = \underbrace{\begin{pmatrix} 1 & m_{1,1} & m_{1,2} & \dots & m_{1,d} \\ 1 & m_{2,1} & m_{2,2} & \dots & m_{2,d} \\ & & \dots\dots\dots\dots & & \\ 1 & m_{N,1} & m_{N,2} & \dots & m_{N,d} \end{pmatrix}}_{q=d+1} \left.\vphantom{\begin{pmatrix}1\\1\\1\\1\end{pmatrix}}\right\} N \tag{4.2}$$

- $x^{(k)} = (x_1^{(k)}, ..., x_d^{(k)})^T \in \mathbb{R}^d$, is a point at iteration $k = 0, 1, ...$;
- $R = (x_1^{(k)} \pm r, \dots, x_d^{(k)} \pm r)^T$, is a small neighbourhood centred at the point $x^{(k)}$, where $r > 0$ is a small constant. The neighbourhood R is often called the region of interest or subregion in RSM-related literature;
- Throughout this chapter, it is assumed that ± 1 in the design matrix represents a change of $\pm r$ in each coordinate of $x^{(k)}$ when evaluating the response function (4.1). Specifically, $z_1, z_2, \dots, z_n$, are the values of a d-dimensional predictor at the k-th iteration such that,

$$z_j = x^{(k)} + r m_j, \tag{4.3}$$

  and entries of the response vector $Y = (y(z_j))_{j=1}^N$ consist of the response function evaluations (4.1) at z_j $(j = 1, \dots, N)$.

The selection of r for the region of interest R is essential in RSM. If r is too small, the errors ε_j $(j = 1, \dots, N)$ in (4.1) will be the main reason for considerable differences between the response function values y_j $(j = 1, \ldots, N)$. Consequently, the search within Phase I of BW will be inaccurate. However, selecting r too large will result in inadequate local models, and therefore, the search within Phase I of BW may stop too early. Hence, r must be chosen carefully.

4.2 Classical strategies for response surface optimization

4.2.1 Overview

The Box-Wilson (BW) algorithm is the key RSM strategy and consists of two phases which are described in Subsection 4.2.2. This chapter focuses on Phase I of BW, which performs a succession of moves towards the neighbourhood of x^*. During Phase I of BW, it is assumed that $x^{(k)}$ is far away from x^* and the response function is approximately linear in the neighbourhood of $x^{(k)}$. Hence, a first-order model is constructed to approximate the response function in the neighbourhood of $x^{(k)}$. If the first-order model is significant, then coefficients of the first-order model are used to determine the search direction, and a sequence of experiments are performed to determine a suitable step length. This process is known as steepest descent and is frequently used within Phase I of BW to move towards the neighbourhood of x^*. This section aims to provide a literature review of the processes and methods associated with Phase I of BW. Specifically, details on the computation of the search direction and step length for steepest descent are outlined in Subsections 4.2.3 and 4.2.4.

4.2.2 Box-Wilson algorithm

The BW algorithm is the key RSM strategy and is summarized below.

Box-Wilson version of RSM

- *Phase I: descent to a neighbourhood of a local minimizer*
  - *Initialization:* Choose a point $x^{(0)} \in \mathbb{R}^d$ (an initial approximation) and set $k = 0$.
  - *k-th iteration of Phase I*

    I.1 Assume $\eta(x)$ is approximately linear in the neighbourhood of $x^{(k)}$:

$$\eta(x) \approx \theta_0 + \theta_1(x_1 - x_1^{(k)}) + \cdots + \theta_d(x_d - x_d^{(k)}), \tag{4.4}$$

where $x = (x_1, \dots, x_d)^T$ and $x^{(k)} = (x_1^{(k)}, \dots, x_d^{(k)})^T$. If the model (4.4) is adequate then

$$\theta_0 = \eta(x^{(k)}),\ \theta_i = \frac{\partial \eta(x)}{\partial x_i}\Big|_{x=x^{(k)}} \text{ and } \nabla\eta(x^{(k)}) = (\theta_1, \dots, \theta_d)^T.$$

Typically, a two-level factorial or fractional factorial design centred at $x^{(k)}$ is used (see [89]) to perform observations (4.1) and to compute the least-squares estimator of the unknown parameters $(\theta_0, \dots, \theta_d)^T$ of the model (4.4). Let $\hat{\theta}^{(k)}$ be the least-squares estimator of the gradient $\nabla\eta(x^{(k)}) = (\theta_1, \dots, \theta_d)^T$.

The significance of the linear regression model is tested by computing, for example, the R^2 or $F-$statistic. If the linear model is insignificant, go to Phase II. Otherwise, go to Step I.2.

I.2 Perform steepest descent iteration:

$$x^{(k+1)} = x^{(k)} - \gamma^{(k)} s^{(k)}, \tag{4.5}$$

where the search direction is $s^{(k)} = \hat{\theta}^{(k)}$ and $\gamma^{(k)} \geq 0$ is some step length. Set $k \to k+1$ and return to I.1.

- *Phase II: fitting a second-order local regression model in a neighbourhood of* x^*
  Since the linear model is insignificant, a second-order model is required. The two-level factorial or fractional factorial design is extended to a design which is suitable for estimating coefficients of the quadratic regression model. This gives an estimator for x^*.

Before conducting Phase I of BW, a screening experiment is typically performed to identify a subset of input variables that have a significant effect on the response (see [89]). This chapter proposes an alternative search direction for steepest descent iterations (4.5) and can be easily computed for all input variables. Including all input variables will potentially enhance the accuracy of search directions utilized in steepest descent iterations (4.5). In addition, computational resources are saved by not performing the screening experiments.

A detailed overview of the Box-Wilson algorithm and RSM in general can be found in [89] and [84, Chapter 11]. Surveys detailing the development and progress of RSM are provided in [59, 87, 88]. Furthermore, frameworks to promote the automation of RSM are discussed in [90, 93]. According to [17], the following two key problems are associated with RSM. Firstly, the RSM process requires human involvement (i.e. selecting R and the appropriate local model), and secondly, RSM is heuristic with no convergence guarantees. To combat the discussed issues, a stochastic trust-region response surface method (STRONG) is proposed in [17], which combines RSM and trust region methods. In addition, STRONG-S is proposed in [18], which utilizes screening designs to solve simulation optimization problems with a large number of variables.

This chapter focuses on Phase I, where the goal is to perform a succession of moves towards the neighbourhood of x^*. The key ingredients of Phase I are:

(a) iterative updating of the sequence of points $x^{(0)}, x^{(1)}, \ldots$;

(b) use of the linear model (4.4) and the iterative update (4.5);

(c) use of the least-squares estimator for estimating coefficients in the model (4.4) and hence constructing the search direction $s^{(k)}$;

(d) a particular choice of the step length $\gamma^{(k)}$.

Steps (a) and (b) are not challenged throughout this chapter. Different options are available for choosing the step length $\gamma^{(k)}$ in (d), and are discussed in Subsection 4.2.3. In this chapter, the primary concern is Step (c), which is the choice of search direction $s^{(k)}$ for steepest descent (4.5). In the RSM-related literature, the choice of $s^{(k)}$ was never seriously questioned and the least-squares estimator $\hat{\theta}^{(k)}$ of the gradient $\nabla\eta(x^{(k)})$ was routinely suggested as $s^{(k)}$. Nevertheless, some minor adjustments to the choice $s^{(k)} = \hat{\theta}^{(k)}$ have been considered in the literature and are discussed in Subsection 4.2.4.

4.2.3 Line search process in Phase I of BW

The succession of moves performed in Phase I of BW to approach the neighbourhood of x^* are determined by steepest descent iterations (4.5), which require a search direction $s^{(k)}$ and step length $\gamma^{(k)}$. The least-squares estimator $\hat{\theta}^{(k)}$ of $\nabla\eta(x^{(k)})$ is suggested in the literature as the search direction $s^{(k)}$. A procedure known as line search is conducted to determine a suitable step length $\gamma^{(k)}$, which involves performing experimental runs along the descent direction until some stopping criterion is met.

Consider

$$y(\gamma) = y(x^{(k)} - \gamma s^{(k)}), \tag{4.6}$$

where $\gamma > 0$ is some initial guess of the step length and $s^{(k)}$ is the search direction. Experimental runs along the search direction $s^{(k)}$ consist of modifying γ and evaluating (4.6). The value of γ which provides the most improvement in the response (4.6) is set as $\gamma^{(k)}$ for the steepest descent iteration (4.5). This process is known as line search. Consider the following procedure for modifying γ,

$$\gamma \leftarrow h(\gamma), \tag{4.7}$$

where $h : \mathbb{R} \to \mathbb{R}$ is chosen depending on the line search strategy. In this subsection, various line search strategies are discussed, along with stopping conditions to determine when a suitable step length is found.

The following basic line search strategy is outlined in [84, 89], and involves moving along increments of the search direction. An increment $\Delta x^{(k)}$ of the search direction is computed as follows,

$$\Delta x^{(k)} = \frac{\hat{\theta}^{(k)}}{\max\limits_{i=1,\ldots d} |\hat{\theta}_i^{(k)}|}. \tag{4.8}$$

The line search process moves sequentially along the increments $\Delta x^{(k)}$ until some stopping criterion is met. Hence, the discussed line search strategy involves setting $s^{(k)} = \Delta x^{(k)}$ and the initial guess as $\gamma = 1$, which is modified by setting $h(\gamma) = \gamma + 1$ in (4.7) until some stopping criterion is met. It is assumed in this chapter that ± 1 in the design matrix represents a change of $\pm r$ in each coordinate of $x^{(k)}$. Therefore, $\Delta x^{(k)}$ may be converted back to the natural units of measurement by setting $s^{(k)} = r\Delta x^{(k)}$. However, this is not necessary if the initial guess γ and choice of h are updated to account for the magnitude of entries of $\Delta x^{(k)}$.

More effective strategies such as forward or backward tracking can be applied for computing the step length $\gamma^{(k)}$. Forward and backward tracking involves repeatedly doubling or halving the initial guess γ until some stopping criterion is met. The first step for forward and backward tracking involves evaluating the response function (4.6) with $\gamma = 0$ and an initial guess $\gamma > 0$, to obtain $y(0)$ and $y(\gamma)$, respectively. Forward tracking is applied if $y(\gamma) < y(0)$ and backward tracking is applied if $y(\gamma) \geq y(0)$. The updating of γ in (4.7) for forward and backward tracking involves setting $h(\gamma) = 2 \times \gamma$ and $h(\gamma) = 0.5 \times \gamma$, respectively. Since there is an option to increase or decrease γ iteratively, the search direction does not need to be of the form (4.8), provided that the starting value of γ is chosen appropriately. That is, selecting the initial value of γ too small will result in little movement along the search direction since changes in the response function (4.6) will be mainly influenced by noise. On the other hand, if the initial value of γ is too large, then many experiments along the search direction may be conducted, and will result in slow convergence since the true form of η in (4.1) is assumed to be unimodal.

Another effective line search strategy is outlined in [34] and involves conducting a set of experiments along the search direction and fitting a second-order polynomial. The second-order polynomial is minimized to determine the step length $\gamma^{(k)}$ for steepest descent (4.5). The set of experiments conducted along the search direction can be obtained, for example, by applying forward or backward tracking.

Deciding when to stop a line search strategy can be difficult when response values contain noise. That is, line search may be stopped before observing the smallest response value along the search direction. As a result, another iteration of Phase I of BW is performed, and consequently, $s^{(k)}$ and $\gamma^{(k)}$ are recomputed. This may be a waste of computational resources since $s^{(k)}$ may be similar to the previous search direction $s^{(k-1)}$. Therefore, line search may be stopped when several consecutive response values are larger than the minimum response value observed so far. This rule is known as the n-in-a-row stopping condition [93]. Typically in practice, if 2 or 3 consecutive experiments along the search direction result in larger response values, the line search is stopped [89]. Formal stopping rules have been proposed by Myers and Khuri [86] and Del Castillo and Miró-Quesada [83] to combat the issues of stopping the line search too early or late.

4.2.4 Adjustments proposed for steepest descent in Phase I of BW

In the literature, the choice of $s^{(k)}$ for steepest descent iterations (4.5) is routinely suggested as the least-squares estimator $\hat{\theta}^{(k)}$ of $\nabla\eta(x^{(k)})$. Nevertheless, some minor adjustments to the choice of $s^{(k)} = \hat{\theta}^{(k)}$ have been considered in the literature, which are discussed in this subsection.

Adapted steepest ascent/descent (ASA/ASD) [64, 66]

Steepest descent (4.5) within Phase I of BW can suffer from two problems. Namely, steepest descent is scale dependent and the step length $\gamma^{(k)}$ is selected manually [64]. To address both issues, adapted steepest ascent (ASA) was proposed in [64], and its counterpart, adapted steepest descent (ASD) was proposed in [66]. The only difference between ASA and ASD is the sign of the adapted search direction (i.e. negative for steepest descent and positive for steepest ascent). Since minimization is the focus of this chapter, ASD will be discussed.

The following is a brief summary of the main ideas behind ASD. Recall from (4.2) that a design matrix M_+ consists of the original design matrix M with an additional first column of ones. Furthermore, suppose $\hat{\theta}_+^{(k)} = (\hat{\theta}_0^{(k)}, \ldots, \hat{\theta}_d^{(k)})$ and $\hat{\theta}^{(k)} = (\hat{\theta}_1^{(k)}, \ldots, \hat{\theta}_d^{(k)})$. Consider the covariance matrix of $\hat{\theta}_+^{(k)}$

$$\begin{aligned} cov(\hat{\theta}_+^{(k)}) &= \sigma^2 (M_+^T M_+)^{-1} && (4.9) \\ &= \sigma^2 \begin{pmatrix} a & b^T \\ b & C \end{pmatrix} && (4.10) \end{aligned}$$

where a is a scalar, b is a vector of size d and C is a $d \times d$ matrix. Also consider the covariance matrix of $\hat{\theta}^{(k)}$

$$cov(\hat{\theta}^{(k)}) = \sigma^2 C.$$

The error variance σ^2 can be estimated by using the mean squared residual estimator with the observed responses and the regression predictor $\hat{y} = M\hat{\theta}^{(k)}$ (see [64, Eq. 5]). Suppose x is some arbitrary point. The predictor variance $\mathrm{Var}(\hat{y}|x)$ increases as x moves further away from the region where the gradient is approximated [64, 66]. Hence, [64] suggests starting the search from the point which achieves minimal predictor variance to ensure the approximated gradient is an accurate estimate of the true gradient. Consider the following ASD procedure outlined in [66],

$$x^{(k+1)} = -C^{-1}b - \gamma^{(k)}C^{-1}\hat{\theta}^{(k)}, \tag{4.11}$$

where $-C^{-1}b$ is the point which achieves minimal predictor variance, $\gamma^{(k)}$ is the step length and $-C^{-1}\hat{\theta}^{(k)}$ is the modified search direction that takes into account the covariance matrix of $\hat{\theta}^{(k)}$. As a consequence, the search direction is scale independent. Discussion on the choice of step length $\gamma^{(k)}$ is also provided in [64].

Incorporation of gradient deflection methods to compute search directions [57]

For each iteration of Phase I of BW, the choice of $s^{(k)}$ for steepest descent iterations (4.5) is frequently suggested as the least-squares estimator $\hat{\theta}^{(k)}$ of $\nabla\eta(x^{(k)})$. However, an opportunity is missed to exploit information gained from previous search directions to improve $s^{(k)}$. In addition, the zigzag behaviour of steepest descent iterations (4.5) can slow convergence. Therefore, a gradient deflection approach for computing the search direction is proposed in [57] to combat the discussed issues. That is, the following gradient deflection method [57, Eq. 3-4] is used to improve the performance of Phase I of BW,

$$x^{(k+1)} = x^{(k)} - \gamma^{(k)}s^{(k)},$$

where

$$s^{(k)} = \hat{\theta}^{(k)} - \psi^{(k)}s^{(k-1)} \tag{4.12}$$

with $s^{(0)} = \hat{\theta}^{(0)}$, multiplier $\psi^{(k)}$ and step length $\gamma^{(k)}$. Possible choices for $\psi^{(k)}$ are discussed in [57, Sect. 1.2], along with restarting criteria for (4.12) to improve the performance of gradient deflection methods.

Incorporation of the conjugate gradient method to compute search directions [31]

A procedure for augmenting the search direction by the conjugate gradient method is proposed in [31]. This involves applying the conjugate gradient method to derive consecutive search directions $s^{(k)}$ instead of using the coefficients of the least-squares estimator $\hat{\theta}^{(k)}$.

Since minimization is the focus of this chapter, the following description of the procedure outlined in [31] is amended for steepest descent. Suppose that a first-order model without interaction is fitted, where coefficients are computed by the least-squares estimator. The first search direction is $s^{(0)} = \Delta x^{(0)}$, where $\Delta x^{(0)}$ is computed using (4.8). The line search strategy involves modifying γ according to (4.7) with $h(\gamma) = \gamma + 1$ until an increase in the response value is observed. The step length γ which corresponds to the smallest response value along the search direction will be set as $\gamma^{(0)}$, and $x^{(1)}$ in (4.5) is obtained.

The k-th iteration involves solving the following with respect to the vector v [31, Eq. 12],

$$(\Delta x^{(k-1)})^T \hat{H} v = 0, \tag{4.13}$$

where $\hat{H}$ is the estimate of the Hessian. Since the fitted first-order model does not contain interaction terms, it is proposed in [31] to use a central composite design centred at $x^{(k)}$ to estimate the Hessian. The search direction is set to $s^{(k)} = v$ and steepest descent (4.5) is applied, where $\gamma^{(k)}$ is computed using the same line search strategy mentioned previously. The process is repeated until several consecutive response values are greater than the minimum response value observed so far. The benefits of using the proposed method are outlined in [31] and include, simple implementation in practical experiments and consideration of possible curvature in the function being investigated.

Direct gradient methods [70]

Direct gradient augmented response surface methodology (DiGARSM) is proposed in [70] and aims to enhance RSM by using response values and gradient estimates to fit regression models. White-box methods utilize available gradient estimates [66], and hence, DiGARSM is a white-box method. However, throughout this chapter, it is assumed that gradient estimates are unavailable since the focus is on black-box stochastic functions.

4.2.5 Summary

Throughout this section, a wide range of literature regarding RSM and, in particular, Phase I of BW has been discussed. Specifically, the strategies for selecting the search direction $s^{(k)}$ and step length $\gamma^{(k)}$ within Phase I are presented. The line search strategies to obtain $\gamma^{(k)}$ involve conducting experiments along the search direction by modifying an initial guess γ using (4.7) until some stopping criterion is met. The choice of $s^{(k)}$ has not been questioned in the RSM-related literature, and the least-squares estimator $\hat{\theta}^{(k)}$ of $\nabla\eta(x^{(k)})$ is routinely suggested as $s^{(k)}$. In some cases, the search direction $s^{(k)}$ is set proportional to $\hat{\theta}^{(k)}$ by employing, for example (4.8), to determine increments along the search direction.

Minor adjustments to $\hat{\theta}^{(k)}$ as the search direction have been considered in the literature and are discussed in Subsection 4.2.4. However, the remainder of this chapter will focus on an alternative search direction to the least-squares estimator $\hat{\theta}^{(k)}$. It will be shown that the alternative search direction can significantly improve Phase I of BW and RSM, in general, for high-dimensional problems.

4.3 Test functions

The purpose of this section is to outline two response functions that will be used throughout this chapter to evaluate the performance of Phase I of BW with different search directions $s^{(k)}$. Consider the following two response functions,

$$y(x) = (x - x^*)^T A^T \Sigma A (x - x^*) + \epsilon, \tag{4.14}$$

and

$$y(x) = \sqrt{(x - x^*)^T A^T \Sigma A (x - x^*)} + \epsilon, \tag{4.15}$$

where $x \in R^d$, $x^* = (0, 0, ..., 0)^T$, A is a random orthogonal matrix, Σ is a diagonal positive definite matrix with smallest and largest eigenvalues λ_{min} and λ_{max} respectively, and $\epsilon \sim N(0, \sigma^2)$. It can be observed that functions (4.14) and (4.15) have the form,

$$y(x) = \eta(x) + \epsilon. \tag{4.16}$$

The performance of Phase I of BW will be measured with varying degrees of noise. Since σ^2 is the variance used to simulate the noise term ϵ, the following modified signal to noise ratio (SNR) will be used to construct values for σ^2,

$$\text{SNR} = \frac{\text{Var}(r \times \eta(x))}{\sigma^2}. \tag{4.17}$$

The quantity $\mathrm{Var}(r \times \eta(x))$ is computed instead of $\mathrm{Var}(\eta(x))$ to account for the region of exploration R. Without this adjustment, the values of σ^2 will be too large. Since $\eta(x)$ is known for (4.14) and (4.15), it is possible to compute $\mathrm{Var}(r \times \eta(x))$ empirically. Therefore, σ^2 can be computed by rearranging (4.17) for various values of SNR. Small SNR will produce large values of σ^2, which will increase the difficulty in finding the minimizer x^*.

4.4 New search direction for Phase I of BW with large dimensions

4.4.1 Overview

Phase I of BW consists of performing a succession of moves towards the neighbourhood of x^*. The succession of moves is determined by steepest descent iterations (4.5) which require a search direction $s^{(k)}$ and step length $\gamma^{(k)}$. In this section, a new search direction for steepest descent iterations (4.5) is proposed to allow efficient and accurate application of Phase I of BW for large dimensions. Specifically, the new search direction has a simple form and can be easily computed, even if the design matrix $M = (m_{j,i})_{j,i=1}^{N,d}$ has less observations N than dimension d.

Typically, the least-squares estimator $\hat{\theta}^{(k)}$ of the gradient $\nabla\eta(x^{(k)})$ is used as the search direction $s^{(k)}$ for steepest descent iterations (4.5). Recall from (4.2) that a design matrix M_+ consists of the original design matrix M with an additional first column of ones. A design matrix M_+ is used to compute the least-squares estimator, where M is a two level fractional factorial design with a larger number of observations N than dimension d. If d is large, deriving the least-squares estimator at each iteration of Phase I of BW can be computationally expensive for the following two reasons. Firstly, the response function will need to be evaluated at a large number of observations to derive the response vector $Y = (y(z_j))_{j=1}^{N}$. Secondly, the inverse $(M_+^T M_+)^{-1}$ will need to be determined, which can significantly reduce efficiency since the size of $(M_+^T M_+)$ is $q \times q$, where $q = d + 1$.

The new search direction is proposed in Subsection 4.4.2 and employs the optimal search direction outlined in [141, Chpt. 8] and [38, 39]. Since a design matrix M of any size can be used to construct the new search direction, a strategy for generating M with any number of observations N and dimension d is discussed and investigated in Subsection 4.4.3.

4.4.2 Motivation of new search direction

The least-squares estimator $\hat{\theta}^{(k)}$ of the gradient $\nabla\eta(x^{(k)})$ is frequently selected as the choice of search direction $s^{(k)}$ for steepest descent iterations (4.5) within Phase I of BW. In general, the main ingredients required for computing $s^{(k)}$ are a design matrix M and a response vector $Y = (y(z_j))_{j=1}^{N}$, where the response function (4.1) is evaluated at each z_j defined in (4.3). Since random variables are used to compute the search directions, the true function value $\eta(x)$ may increase rather than decrease along some search directions. As a result, [141, Chpt. 8] proposes that the probability of decrease in η should be selected as the optimality criterion for choosing the search direction. If a function η decreases in some direction s, then,

$$\frac{\partial\eta}{\partial(-s)}(x^{(k)}) = [\nabla\eta(x^{(k)})]^T(-s) = \theta^T(-s) < 0.$$

Hence, the probability of decrease of η in the direction $-s$ is

$$\Pr\{\theta^T s > 0\}. \tag{4.18}$$

The problem of estimating the optimal direction in regression by maximizing the probability (4.18) that the scalar product between the vector of unknown parameters and the chosen direction is positive is considered in [141, Chpt. 8] and [38, 39]. If $M = (m_{j,i})_{j,i=1}^{N,d}$ is a design matrix and $Y \sim N(0, \sigma^2 I_d)$, then the statistic

$$\hat{\theta}_* = M^T Y, \tag{4.19}$$

maximizes (4.18) over all vectors $s \in \mathbb{R}^d$. The optimal direction vector (4.19) has a simple form and can be easily computed with a design matrix M of any size. The following theorem and proof from [39, Thm. 2.1] verifies that the statistic (4.19) maximizes (4.18) over all vectors $s \in \mathbb{R}^d$.

Theorem 3. *Consider the linear regression model $Y = M\theta + \epsilon$, where $\epsilon \sim N(0, \sigma^2 I_d)$ and $\sigma^2 > 0$. Consider the following family of linear statistics*

$$\hat{\theta}_C = CY \tag{4.20}$$

where C is a $d \times N$ matrix. Also consider the scalar product in $\mathbb{R}^d$,

$$\langle a, b\rangle = a^T S b, \tag{4.21}$$

where $a, b \in \mathbb{R}^d$ and S is some positive definite matrix of size $d \times d$. It is desirable to find the set of $d \times N$ matrices C_* such that,

$$\Pr\{\langle\hat{\theta}_{C_*}, \theta\rangle \ > 0\} = \max_C \Pr\{\langle\hat{\theta}_C, \theta\rangle \ > 0\}. \tag{4.22}$$

For any θ, it will be shown that $C_* = S^{-1}M^T$.

Proof.

Consider

$$t(C,\theta) = \langle \hat{\theta}_C, \theta \rangle = \theta^T SCY.$$

Then

$$\mathbb{E}\, t(C,\theta) = \theta^T SCM\theta \quad \textit{and} \quad \mathrm{Var}[t(C,\theta)] = \sigma^2 \theta^T SCC^T S\theta.$$

Suppose that $\theta \neq 0$ *and* $C^T S\theta \neq 0$ *(see [39, Thm. 2.1] for more details). Then* $\mathrm{Var}[t(C,\theta)] > 0$ *and the following random variable can be defined,*

$$v(C,\theta) = \frac{\sigma[t(C,\theta) - \mathbb{E}\, t(C,\theta)]}{\sqrt{\mathrm{Var}[t(C,\theta)]}} = \frac{t(C,\theta) - \theta^T SCM\theta}{\sqrt{\theta^T SCC^T S\theta}},$$

which is normally distributed with zero mean and variance σ^2*. For any matrix* C*,*

$$\Pr\{\langle \hat{\theta}_C, \theta \rangle \ > 0\} = \Pr\{v(C,\theta) > -\varphi(C,\theta)\},$$

where

$$\varphi(C,\theta) = \frac{\theta^T SCM\theta}{\sqrt{\theta^T SCC^T S\theta.}}$$

Hence, maximizing the probability $\Pr\{\langle \hat{\theta}_C, \theta \rangle \ > 0\}$ *is equivalent to maximizing* $\varphi(C,\theta)$ *with respect to* C*. Therefore, it is required to find,*

$$C_* = \operatorname*{argmax}_C \varphi(C,\theta). \tag{4.23}$$

Recall the Cauchy-Schwartz inequality which states that for two non-zero vectors a *and* b $\sqrt{a^T a}\sqrt{b^T b} \geq a^T b$*. If* $a = C^T S\theta$ *and* $b = M\theta$*, then*

$$\sqrt{\theta^T SCC^T S\theta}\sqrt{\theta^T M^T M\theta} \geq \theta^T SCM\theta.$$

As a consequence,

$$\varphi(S^{-1}M^T,\theta) = \sqrt{\theta^T M^T M\theta} \geq \frac{\theta^T SCM\theta}{\sqrt{\theta^T SCC^T S\theta}} = \varphi(C,\theta),$$

for all θ*. Hence,* $C_* = S^{-1}M^T$ *is one of the matrices that satisfies* (4.23) *and* $C_* = M^T$ *when* $S = I_d$*.* □

Analytical and numerical examples in [39] confirm the superiority of (4.19) over other linear estimates, such as the least-squares estimator and the Lasso. The result in (4.19) is utilized to derive the following new search direction,

$$s^{(k)} = M^T Y, \tag{4.24}$$

where $M = (m_{j,i})_{j,i=1}^{N,d}$ is a design matrix and $Y = (y(z_j))_{j=1}^{N}$ is the response vector.

Optimization methods for model training within machine learning are discussed in Section 2.6.4, and include batch gradient descent (BGD) and stochastic gradient descent (SGD). For high-dimensional problems, the number of observations N and dimension d will be very large for a design matrix M. Hence, deriving the gradient within BGD will be computationally expensive and result in slow convergence. To reduce the computational cost, SGD is often applied and involves approximating the true gradient by computing the gradient for a subset of observations selected at random. However, for large dimensions, [140] outlines that the approximated gradient in SGD may be improved by employing search directions of the form (4.24), where M is the design matrix at chosen observations, which are updated at each iteration. Nevertheless, the main focus of this chapter is to investigate the performance of Phase I of BW for large dimensions with search directions of the form (4.24).

4.4.3 Choice of design matrix M

The new search direction proposed in (4.24) presents two immediate benefits in comparison to the least-squares estimator $\hat{\theta}^{(k)}$ as the choice of search direction. Firstly, the new search direction (4.24) can be computed with design matrices $M = (m_{j,i})_{j,i=1}^{N,d}$, where the number of observations N is less than the dimension d. Secondly, the computational complexity of constructing the new search direction (4.24) is more efficient than the least-squares estimator as only matrix multiplication is performed as opposed to matrix inversion and multiplication.

The response vector $Y = (y(z_j))_{j=1}^{N}$ is used to construct the new search direction (4.24) and as a result, N response function evaluations are required to compute $s^{(k)}$. Suppose there is a bound on the number of response function evaluations performed during Phase I of BW. In that case, selecting the number of observations N for a design matrix M is determined by obtaining a suitable balance between the number of iterations of Phase I of BW to perform, and the quality of search directions $s^{(k)}$ constructed at each iteration. Specifically, if N is small, then many more iterations of Phase I of BW are permitted but the quality of search directions $s^{(k)}$ may vary. On the other hand, if N is relatively large, then fewer iterations of Phase I of BW will be performed but search directions $s^{(k)}$ may be more accurate. Hence, the construction of design matrices M with different N will need to be investigated when $s^{(k)}$ is computed using (4.24). The purpose of this subsection is to outline a strategy to generate design matrices M of any size that can be used for the computation of the new search direction (4.24).

Strategy for constructing a design matrix M

Consider the following strategy to produce design matrices $M = (m_{j,i})_{j,i=1}^{N,d}$ for new search directions (4.24). Entries $m_{j,i}$ are randomly selected as +1 or -1, with the condition that each column of M contains the same number of ±1's. That is,

$$\sum_{j=1}^{N} m_{j,i} = 0, \tag{4.25}$$

for all $i = 1, \dots, d$. Therefore, the number of observations N of the design matrix M can be chosen accordingly. The only restriction is that the choice of N must be even due to condition (4.25).

The following is an example of a possible choice of M,

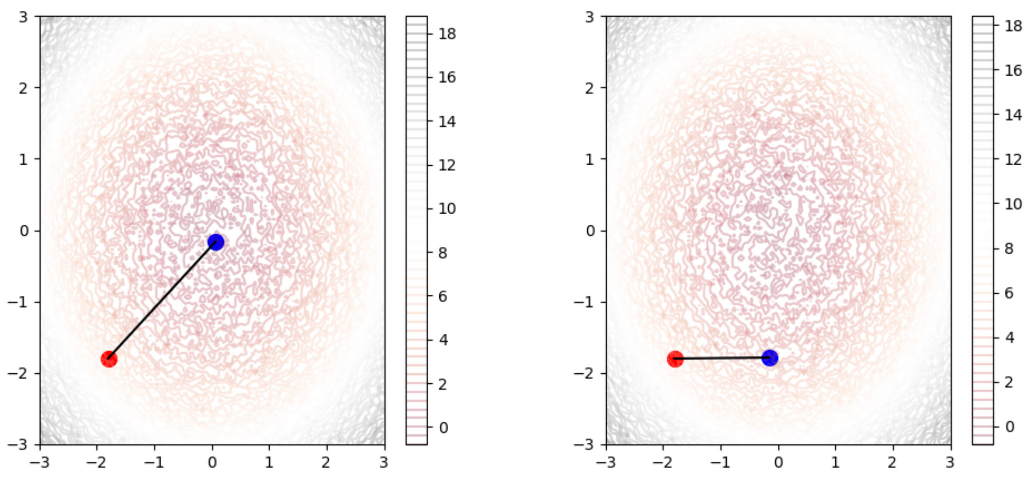
$$M = \begin{pmatrix} -1 & +1 & +1 & -1+1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1+1 & +1 & +1 & -1 \\ -1 & +1 & -1 & +1+1 & -1 & -1 & +1 \\ +1 & -1 & +1 & -1-1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1-1 & -1 & +1 & +1 \\ +1 & -1 & +1 & +1-1 & +1 & -1 & +1 \end{pmatrix}, \tag{4.26}$$

where $N = 6$ and $d = 8$.

Investigation of the strategy for constructing a design matrix M

The purpose of including condition (4.25) to construct design matrices M is to reduce the variation of the magnitude of entries within the new search direction (4.24). To illustrate this concept, Figure 4.1 shows contour plots of the response function (4.14) with minimizer $x^* = (0,0)^T$ and an initial point $x^{(0)}$. Specifically, the application of two different search directions, $s^{(0)} = (-0.8, -0.7)^T$ and $s^{(0)} = (-80, -0.7)^T$, for steepest descent (4.5) are portrayed in Figures 4.1(a) and 4.1(b) to obtain different $x^{(1)}$. The signs of entries in both search directions $s^{(0)}$ are the same. However, the variation of the magnitude of entries within each $s^{(0)}$ differs significantly. Figure 4.1 highlights that applying steepest descent (4.5) with $s^{(0)} = (-80, -0.7)^T$ results in movement along the first coordinate $s_1^{(0)}$, but no movement along the second coordinate $s_2^{(0)}$. Furthermore, suppose the signs of some entries of $s^{(0)}$ are incorrect, and the magnitude of the entries vary greatly. In that case, minimal movement will be made along the search direction during an iteration of steepest descent (4.5).

Numerical investigations are performed to demonstrate that applying condition (4.25) for the construction of design matrices M reduces the variation of entries within the new search direction (4.24). That is, 100 response functions y_i ($i = 1, \dots, 100$) of the form (4.14) are generated, with smallest and largest eigenvalues λ_{min} and λ_{max} respectively. All other eigenvalues are sampled uniformly from $(\lambda_{min}, \lambda_{max})$. For investigations, $\lambda_{min} = 1$ and $\lambda_{max} = 1, 4, 8$. Recall that response functions (4.14) contain an error term $\epsilon \sim N(0, \sigma^2)$. The value of σ^2 is



(a) $s^{(0)} = (-0.8, -0.7)^T$ (b) $s^{(0)} = (-80, -0.7)^T$

Figure 4.1: Contour plots of response function (4.14) with minimizer $x^* = (0, 0)^T$, $A^T \Sigma A = I_d$ and $\sigma = 0.25$ are displayed, along with $x^{(0)}$ (red) and $x^{(1)}$ (blue), which is computed using an iteration of steepest descent (4.5) with the corresponding search direction $s^{(0)}$.

obtained by rearranging (4.17) for SNR $= 0.5, 2, 5$. For each response function y_i ($i = 1, \ldots, 100$), a random point $x^{(0)} \sim N(0, I_d)$ will also be generated. A small neighbourhood $R = (x_1^{(0)} \pm r, \ldots, x_d^{(0)} \pm r)^T$ centred at $x^{(0)}$ with $r = 0.1$ is explored. That is, ± 1 in the design matrix represents a change of ± 0.1 in the co-ordinates of the centre point $x^{(0)}$. To construct $s^{(0)}$ of the form (4.24), design matrices M will be generated with and without condition (4.25).

Figure 4.2 shows boxplots of $\mathrm{Var}(s^{(0)})$, where design matrices M are constructed with and without condition (4.25). If design matrices M are constructed without condition (4.25), then each column of M may contain an unequal number of ± 1's.

It can be observed in Figure 4.2 that the variation of entries of $s^{(0)}$ is largest for design matrices M that have been generated without condition (4.25). Also, $\mathrm{Var}(s^{(0)})$ increases as a function of λ_{max}. For design matrices M generated with condition (4.25), $\mathrm{Var}(s^{(0)})$ decreases as a function of SNR. However, such a trend is not apparent for design matrices M generated without condition (4.25).

In general, suppose the magnitude of entries vary greatly within $s^{(k)}$ and in the best case scenario, the entries of $s^{(k)}$ with the largest magnitude contain the correct sign. In that case, only movement along the entries of $s^{(k)}$ with the largest magnitude will be permitted to obtain $x^{(k+1)}$ in (4.5). Consequently, many iterations may be required before finding a subregion of the optimum. However, if some entries

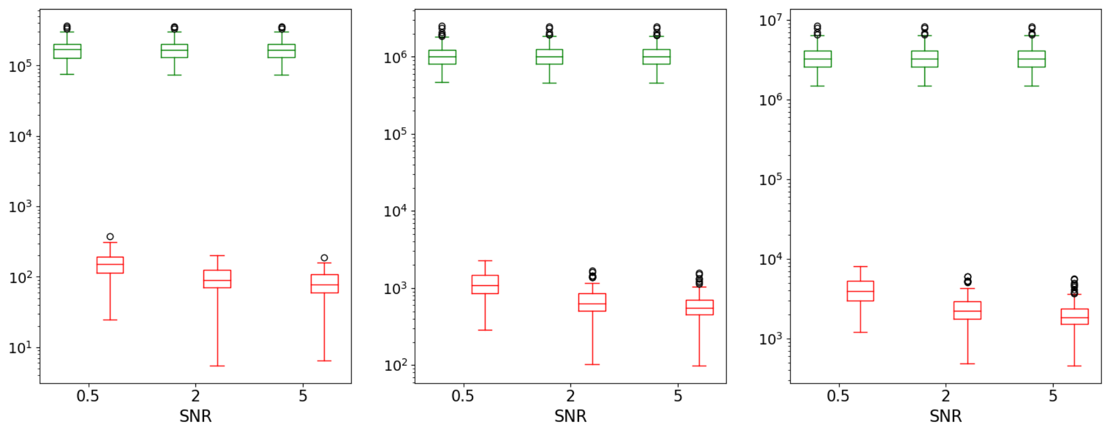


Figure 4.2: Boxplots of $\text{Var}(s^{(0)})$, where $s^{(0)}$ is computed using (4.24), with 100 functions of the form (4.14) with $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) where M has been constructed without (green) and with (red) condition (4.25), for $N = 16$, $d = 100$, $r = 0.1$ and $\text{SNR} = 0.5, 2, 5$.

of the search direction $s^{(k)}$ with the largest magnitude contain the incorrect sign, then minimal movement will be made along $s^{(k)}$. Hence, design matrices M will be generated with condition (4.25) when computing $s^{(k)}$ of the form (4.24) throughout this chapter.

4.4.4 Phase I of BW with new search direction $s^{(k)} = M^TY$ (PI-MY)

An alternative stopping criterion for Phase I of BW will need to be applied when $s^{(k)}$ is of the form (4.24). Recall that $\gamma^{(k)}$ and $s^{(k)}$ are computed for each iteration of steepest descent (4.5) in Phase I of BW. Let $c_{\gamma^{(k)}}$ be the total number of response function evaluations to compute step size $\gamma^{(k)}$, $c_{s^{(k)}}$ be the total number of response function evaluations to compute the search direction $s^{(k)}$ and c be a bound on the total number of response function evaluations used to compute $s^{(k)}$ and $\gamma^{(k)}$. Suppose K is the total number of iterations of Phase I of BW computed so far. Phase I of BW with $s^{(k)}$ of the form (4.24) terminates if

$$c \leq \sum_{k=0}^{K} (c_{\gamma^{(k)}} + c_{s^{(k)}} + N). \tag{4.27}$$

Consider the following algorithm for Phase I of BW with new search direction $s^{(k)}$ of form (4.24), abbreviated as PI-MY hereafter.

PI-MY algorithm

Initialization: Select an initial point $x^{(0)} \in \mathbb{R}^d$, a value for c and set $k \leftarrow 0$, $c_{\gamma^{(k)}} \leftarrow 0$ and $c_{s^{(k)}} \leftarrow 0$.

I.1 (a) If stopping condition (4.27) is not satisfied, go to Step (b). Otherwise, return $x^{(k)}$.

(b) Generate a design matrix M (see Subsection 4.4.3) and obtain $Y = (y(z_j))_{j=1}^{N}$ by evaluating the response function (4.1) at z_j defined in (4.3).

(c) Compute $s^{(k)}$ of the form (4.24) and go to Step I.2.

I.2 Apply an iteration of steepest descent (4.5) to obtain $x^{(k+1)}$. Set $k \leftarrow k+1$ and update the values of $c_{\gamma^{(k)}}$ and $c_{s^{(k)}}$. Repeat Step I.1 with $x^{(k)}$.

4.4.5 Summary

This section proposes a new search direction $s^{(k)}$ of the form (4.24), which can be used for steepest descent iterations (4.5) within Phase I of BW for large dimensions. The least-squares estimator $\hat{\theta}^{(k)}$ of the gradient $\nabla\eta(x^{(k)})$ is routinely suggested as the search direction $s^{(k)}$ for steepest descent iterations (4.5). However, applying the new search direction (4.24) provides two key advantages compared to the least-squares estimator $\hat{\theta}^{(k)}$. Firstly, the design matrix M can be of any size to compute the new search direction (4.24). As a result, there is an opportunity to reduce the number of observations N of a design matrix to allow more iterations of Phase I of BW. This is not the case for the computation of the least-squares estimator since a two-level fractional factorial design is required, where the number of observations N is fixed and greater than d. Flexibility on the choice of N will increase the likelihood of improving the accuracy and efficiency of Phase I of BW. Secondly, the computational complexity of constructing the new search directions (4.24) is far less than that of deriving the least-squares estimator. A strategy for computing design matrices M is proposed to exploit the discussed advantages, and analysis is conducted to justify the inclusion of condition (4.25). Furthermore, the algorithm for Phase I of BW with new search direction $s^{(k)}$ of form (4.24), denoted PI-MY, is also presented.

4.5 Modified least-squares search direction for Phase I of BW with large dimensions

4.5.1 Overview

The purpose of this section is to outline two modifications to the least-squares estimator $\hat{\theta}^{(k)}$ of $\nabla\eta(x^{(k)})$ as the choice of search direction for Phase I of BW with large dimensions. Recall from (4.2) that a design matrix M_+ consists of the original design matrix M with an additional first column of ones. A design matrix M_+ is used to compute the least-squares estimator, where M is typically a two-level fractional factorial design with a larger number of observations N than dimension d. As discussed in Subsection 4.2.2, a screening experiment can identify a subset of input variables that influence the response function value, thus reducing the dimension d. However, the new search direction (4.24) can be easily constructed for all input variables d and this can potentially improve the accuracy of search directions $s^{(k)}$ utilized in steepest descent iterations (4.5). Therefore, screening experiments will not be performed during investigations of the new search direction (4.24) within Phase I of BW.

It is desirable to compare the performance of Phase I of BW with the new search direction (4.24) and the least-squares estimator $\hat{\theta}^{(k)}$ as the choice of search direction. The following modifications to the least-squares estimator are proposed to ensure a fair investigation with the new search direction (4.24) when the dimension is large. Firstly, a small subset of input variables will be considered for the computation of the least-squares estimator instead of all input variables. Therefore, a two-level fractional factorial design with fewer variables and observations is required, and the non-zero entries of the search direction correspond to the input variables chosen. Hence, the discussed modification allows the application of Phase I of BW for large d and small N since only a small subset of coordinates of $x^{(k)}$ is updated at each iteration. Consequently, comparisons with the new search direction (4.24) can be performed with design matrices M, where $N < d$. Secondly, application of the least-squares estimator can be expanded by employing design matrices M, proposed in Subsection 4.4.3, to replace the two-level fractional factorial designs. However, the inverse $(M_+^T M_+)^{-1}$ will not exit if $N < q$, where $q = d + 1$. Therefore, the Moore-Penrose pseudoinverse can be used instead of the multiplicative inverse to construct the search direction within Phase I of BW. The performance of the proposed modification and the new search direction (4.24) can be compared with different N and d, since design matrices M are generated using the same strategy in Subsection 4.4.3.

4.5.2 Phase I of BW with adjusted least-squares search direction (PI-ALS)

The following adjustment is made to the BW algorithm in Subsection 4.2.2, to ensure the least-squares estimator can be used as the search direction when the dimension is large. Suppose d is considerably large (e.g. $d = 100$), then at k-th iteration of Phase I of BW it is natural to choose several random coordinates in which the updating (4.5) of $x^{(k)}$ is attempted. As a result, a two-level fractional factorial design matrix with fewer variables and observations is required. Hence, fewer response function evaluations (4.1) are used to construct the response vector $Y = (y(z_j))_{j=1}^{N}$ and the computational cost associated with deriving the multiplicative inverse is reduced. The least-squares estimator is used to estimate the unknown parameters of the linear model (4.4) corresponding to the chosen random coordinates of $x^{(k)}$. If the linear model is significant, then the non-zero entries of the search direction correspond to the estimated coefficients of the linear model, and the updating (4.5) of $x^{(k)}$ is performed. Otherwise, if the linear regression model is insignificant for the chosen coordinates, several other random coordinates are chosen, and the updating (4.5) of $x^{(k)}$ is attempted again. Phase I of BW terminates when all d coordinates have been explored at the k-th iteration, and no update has been made to $x^{(k)}$ due to insignificant linear regression models for various sets of random coordinates.

Phase I of BW with the adjusted least-squares method for computing the search direction $s^{(k)}$ will be denoted as PI-ALS hereafter. PI-ALS can be applied for large d since a small subset of coordinates of $x^{(k)}$ is selected at each iteration to compute the least-squares estimate. Furthermore, since N is chosen according to the number of selected random coordinates instead of the dimension d, the computational complexity of deriving the least-squares estimator is significantly reduced. Consequently, numerical experiments with PI-ALS and PI-MY can be compared for $N < d$. Consider the following algorithm for PI-ALS.

PI-ALS algorithm

Initialization: Select an initial point $x^{(0)} \in \mathbb{R}^d$ and a value L such that $L < d$. Set $k \leftarrow 0$

I.1 (a) Set $H = \{1, \dots, d\}$ and select a subset of random variables $\{p_1, \dots, p_L\} \subseteq H$.

(b) Select a fractional factorial design $\tilde{M} = (\tilde{m}_{j,l})_{j,l=1}^{N,L}$ and let

$$m_{j,i} = \begin{cases} \tilde{m}_{j,l}, & \text{if } i = p_l, \text{ where } l = 1, \dots, L \\ 0, & \text{otherwise,} \end{cases} \tag{4.28}$$

for $j = 1, \dots, N$ and $i = 1, \dots, d$. The response values $Y = (y(z_j))_{j=1}^N$ are computed using (4.1) and least-squares is used to obtain $\hat{\theta}_+ = (\hat{\theta}_0, ..., \hat{\theta}_L)^T$.

(c) The significance of the linear regression model is tested by computing, for example, the R^2-statistic or F-test. If the linear regression model is significant, let

$$s_i^{(k)} = \begin{cases} \hat{\theta}_l, & \text{if } i = p_l, \text{ where } l = 1, \dots, L \\ 0, & \text{otherwise,} \end{cases} \tag{4.29}$$

and go to Step I.2. If the linear regression model is insignificant, set

$$H \leftarrow H \setminus \{p_1, \dots, p_L\}.$$

If $|H| \geq L$, select another random subset of variables $\{p_1, \dots, p_L\} \subseteq H$ and go to Step (b). Otherwise, return $x^{(k)}$.

I.2 Apply steepest descent iteration (4.5) to obtain $x^{(k+1)}$ and set $k \leftarrow k + 1$. Repeat Step I.1 with $x^{(k)}$.

4.5.3 Phase I of BW with Moore-Penrose psuedoinverse for computation of the search direction (PI-MPI)

Recall from (4.2) that a design matrix M_+ consists of the original design matrix M with an additional first column of ones. Hence, the size of M_+ is $N \times q$, where $q = d + 1$. Typically, M is a two-level fractional factorial design with number of observations N greater than the dimension d. However, to expand the use of least-squares, design matrices M will be constructed according to the strategy in Subsection 4.4.3.

If $N < q$, then $(M_+^T M_+)$ will be of rank N and the inverse $(M_+^T M_+)$ will not exist. The Moore-Penrose pseudoinverse provides a generalized inverse when the matrix is singular. Therefore, it is proposed that the Moore-Penrose pseudoinverse is used to compute the least-squares estimator. Let $(.)^-$ denote the Moore-Penrose pseudoinverse and let $(.)^{-1}$ denote the standard multiplicative inverse. Suppose that $\text{rank}(M_+^T M_+) = q$ when $N \geq q$ and $\text{rank}(M_+ M_+^T) = N$ when $N < q$. Then the following equivalence forms for the Moore-Penrose pseudoinverse hold,

$$(M_+)^- = \begin{cases} (M_+^T M_+)^{-1} M_+^T, & \text{if } N \geq q \\ M_+^T (M_+ M_+^T)^{-1}, & \text{otherwise.} \end{cases}$$

Suppose that $\hat{\theta}_+^{(k)} = (\hat{\theta}_0^{(k)}, \hat{\theta}_1^{(k)}, ..., \hat{\theta}_d^{(k)})^T$. The least-squares estimator using the

Moore-Penrose pseudoinverse with $N < q$ finds

$$\min \|\hat{\theta}_+^{(k)}\|_2,$$

which is favourable since large coefficients of $\hat{\theta}_+^{(k)}$ magnify errors.

Throughout this chapter, the `numpy.linalg.pinv` function from the Numpy library [47] in Python will be used to compute the Moore-Penrose pseudoinverse. The `numpy.linalg.pinv` function returns the generalized inverse of the matrix using singular value decomposition (SVD), where relatively large singular values are selected for the computation of SVD. The following details explain the computation of the generalized inverse of a matrix using SVD. Suppose $\text{rank}(M_+) = p$, where $p = \min(N, q)$, and let $(\sigma_1, ..., \sigma_p)$ be the singular values of M_+. The SVD of M_+ is

$$M_+ = U\Sigma V^T,$$

where Σ is a diagonal matrix of size $p \times p$ containing the singular values, and U of size $N \times p$ and V of size $q \times p$ contains the left and right singular vectors, respectively. Suppose

$$\sigma_{max} = \max_{i=1,...,p} (\sigma_i)$$

and,

$$\tilde{\sigma}_i = \begin{cases} \sigma_i, & \text{if } \sigma_i > \sigma_{max} \times \tau \\ 0, & \text{otherwise,} \end{cases} \tag{4.30}$$

where $\tau > 0$ is a very small constant and $i = 1, \dots, p$. Then Σ^- is constructed as follows,

$$\Sigma^-_{i,i} = \begin{cases} 1/\tilde{\sigma}_i, & \text{if } \tilde{\sigma}_i > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{4.31}$$

The generalized inverse of M_+ is

$$(M_+)^- = V\Sigma^- U^T. \tag{4.32}$$

The least-squares estimator can be computed using the generalized inverse to obtain

$$\hat{\theta}_+^{(k)} = (M_+)^- Y, \tag{4.33}$$

where $Y = (y(z_j))_{j=1}^N$ is the response vector. The following search direction can be applied within Phase I of BW,

$$s^{(k)} = (\hat{\theta}_1^{(k)}, ..., \hat{\theta}_d^{(k)})^T, \tag{4.34}$$

using coefficients of (4.33).

Phase I of BW with search directions $s^{(k)}$ of the form (4.34) will be abbreviated as PI-MPI hereafter. An alternative stopping criterion is required for PI-MPI to that used in the original Phase I of BW in Subsection 4.2.2, since design matrices M_+ with $N < q$ may be used to derive $s^{(k)}$ in (4.34). Therefore, the same stopping criterion (4.27) as PI-MY is used for PI-MPI.

Numerical experiments with PI-MPI and PI-MY can be compared for various N and d since design matrices M are constructed according to the strategy in Subsection 4.4.3. Consider the following algorithm for PI-MPI.

PI-MPI algorithm

Initialization: Select an initial point $x^{(0)} \in \mathbb{R}^d$, a value for c and set $k \leftarrow 0$, $c_{\gamma^{(k)}} \leftarrow 0$ and $c_{s^{(k)}} \leftarrow 0$.

I.1 (a) If stopping condition (4.27) is not satisfied, go to Step (b). Otherwise, return $x^{(k)}$.

(b) Generate a design matrix M (see Subsection 4.4.3) and obtain $Y = (y(z_j))_{j=1}^{N}$ by evaluating the response function (4.1) at z_j in (4.3). Construct M_+ of the form (4.2) by adding an additional first column of ones to M.

(c) Obtain $s^{(k)}$ of the form (4.34) by computing (4.33) and go to Step I.2.

I.2 Apply steepest descent iteration (4.5) to obtain $x^{(k+1)}$. Set $k \leftarrow k + 1$ and update the values of $c_{\gamma^{(k)}}$ and $c_{s^{(k)}}$. Repeat Step I.1 with $x^{(k)}$.

Investigation of the use of the SVD to compute the generalized inverse

Recall that the generalized inverse is computed using (4.32), where the diagonal matrix Σ^- is constructed using (4.31) and entries $\Sigma^-_{i,i}$ $(i = 1, \dots, p)$ are determined by $\tilde{\sigma}_i$ in (4.30). The selection of τ may influence the assignment of $\tilde{\sigma}_i$ in (4.30). That is, if τ is too small, then some diagonal entries of Σ^- may be very large. As a consequence, large variation amongst the non-zero diagonal entries of Σ^- will increase the variance of the entries of $s^{(k)}$ in (4.34). Suppose that the magnitude of entries vary greatly within $s^{(k)}$ of the form (4.34) and in the best case scenario, the entries of $s^{(k)}$ with the largest magnitude contain the correct sign. In that case, only movement along the entries of $s^{(k)}$ with the largest magnitude will be permitted to obtain $x^{(k+1)}$ in (4.5). On the other hand, if τ is too large, then the pseudoinverse may be less accurate. The default is $\tau = 1e{-}15$ when the `numpy.linalg.pinv` function is employed to compute the Moore-Penrose pseudoinverse.

The influence of τ on the assignment of $\tilde{\sigma}_i$ $(i = 1, \dots, p)$ in (4.30) will be investigated for 100 design matrices M_+ of the form (4.2), where M is constructed with

$N = 50, 100, 200$ and $d = 100$. Consider

$$\tilde{\sigma}_{min} = \min_{\substack{i=1,\dots,p; \\ \tilde{\sigma}_i \neq 0}} (\tilde{\sigma}_i), \tag{4.35}$$

where $\tilde{\sigma}_i$ $(i = 1, \dots, p)$ is constructed in (4.30).

Figure 4.3 displays $\tilde{\sigma}_{min}$, where $\tau = 1e{-}15, 0.001, 0.01, 0.15$ has been applied in (4.30). Also, Table 4.1 shows the mean number of diagonal entries of Σ^- that are non-zero.

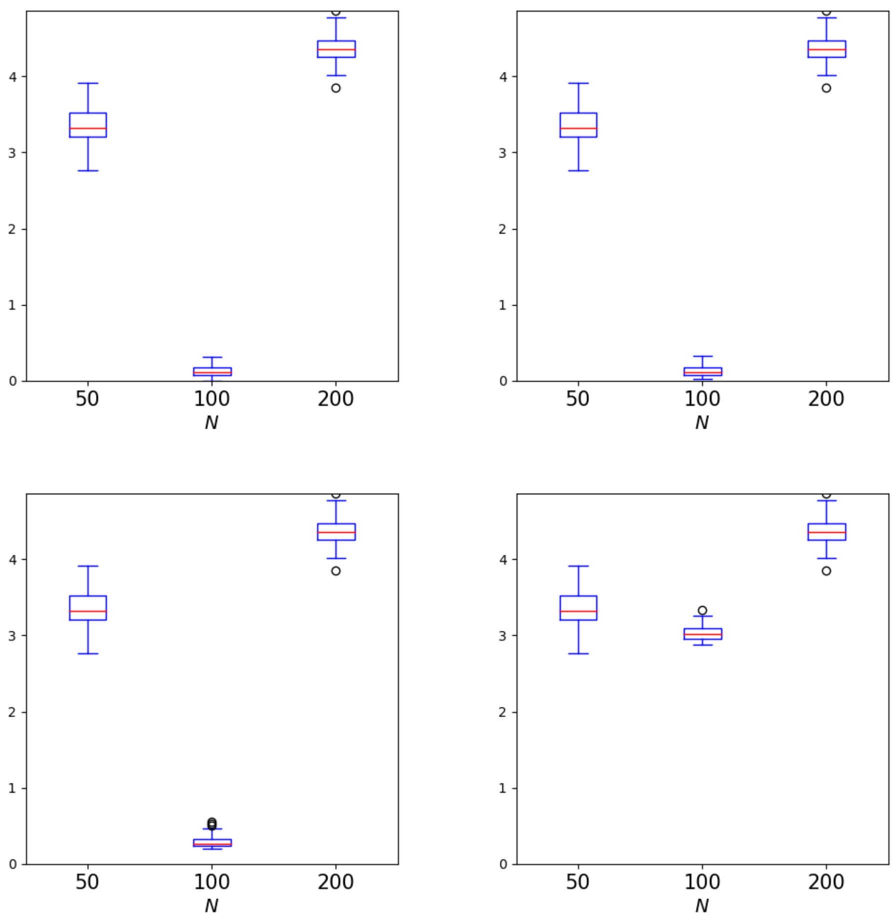


Figure 4.3: Boxplots of $\tilde{\sigma}_{min}$ (4.35), where $\tau = 1e{-}15$ (top left), $\tau = 0.001$ (top right), $\tau = 0.01$ (bottom left) and $\tau = 0.15$ (bottom right) has been applied in (4.30), with 100 design matrices M_+ of the form (4.2), where M is generated for $N = 50, 100, 200$ and $d = 100$.

Figure 4.3 shows that the value of τ has no effect on $\tilde{\sigma}_{min}$ when the number of observations of M_+ is $N = 50, 200$. Recall that Σ^- is of size $p \times p$, where $p = \min(N, q)$. Table 4.1 shows that for $N = 50$, the number of non-zero diagonal entries of Σ^- corresponds to $\text{rank}(M_+) = \min(N, q) = 50$. For $N = 200$, the number

N	τ			
	$1e-15$	0.001	0.01	0.15
50	50	50	50	50
100	100	99.98	99.12	82.06
200	101	101	101	101

Table 4.1: Mean number of diagonal entries of Σ^- that are non-zero in (4.31) for different values of τ with 100 design matrices M_+ of the from (4.2), where M is generated with $N = (50, 100, 200)$ and $d = 100$.

of non-zero diagonal entries of Σ^- is equivalent to $\text{rank}(M_+) = \min(N, q) = 101$. Hence, the number of non-zero diagonal entries of Σ^- is completely determined by $rank(M_+)$ when $N = 50, 200$ as opposed to different τ. For $N = 100$ and $d = 100$, Figure 4.3 shows that the value of $\tilde{\sigma}_{min}$ can differ depending on the value of τ. If τ is very small, then $\tilde{\sigma}_{min}$ will be minimal, and the corresponding diagonal entry of Σ^- will be very large. As a result, there will be large variation amongst the non-zero diagonal entries of Σ^-.

The impact of different τ on search directions (4.34) with design matrices M_+ of the form (4.2), where M is generated for $N = 100$ and $d = 100$ will be investigated. For numerical investigations, 100 response functions y_i $(i = 1, \dots, 100)$ of the form (4.14) are generated, with smallest and largest eigenvalues λ_{min} and λ_{max} respectively. All other eigenvalues are sampled uniformly from $(\lambda_{min}, \lambda_{max})$. For investigations, $\lambda_{min} = 1$ and $\lambda_{max} = 1, 4, 8$. Recall that response functions (4.14) contain an error term $\epsilon \sim N(0, \sigma^2)$. The value of σ^2 is obtained by rearranging (4.17) for SNR $= 0.5, 2, 5$. For each response function y_i $(i = 1, \dots, 100)$, a random point $x^{(0)} \sim N(0, I_d)$ will also be generated. A small neighbourhood $R = (x_1^{(0)} \pm r, \dots, x_d^{(0)} \pm r)^T$ centred at $x^{(0)}$ with $r = 0.1$ is explored. Different values of τ have been applied in (4.30) to determine the diagonal entries of Σ^- for computation of the generalized inverse for the least-squares estimator (4.33), and consequently, to obtain search directions $s^{(0)}$ of the form (4.34). Figure 4.4 shows boxplots of $\text{Var}(s^{(0)})$, where $s^{(0)}$ is of the form (4.34).

Figures 4.3 and 4.4 show that small values of τ increase the variation amongst the non-zero diagonal entries of Σ^-, which increases the variance of $s^{(0)}$ when design matrices M_+ are of the form (4.2), where M is generated with $N = 100$ and $d = 100$. Suppose the magnitude of entries vary greatly within $s^{(k)}$ and in the best case scenario, the entries of $s^{(k)}$ with the largest magnitude contain the correct sign. In that case, only movement along the entries of $s^{(k)}$ with the largest magnitude will be permitted to obtain $x^{(k+1)}$ in (4.5). Consequently, many iterations may

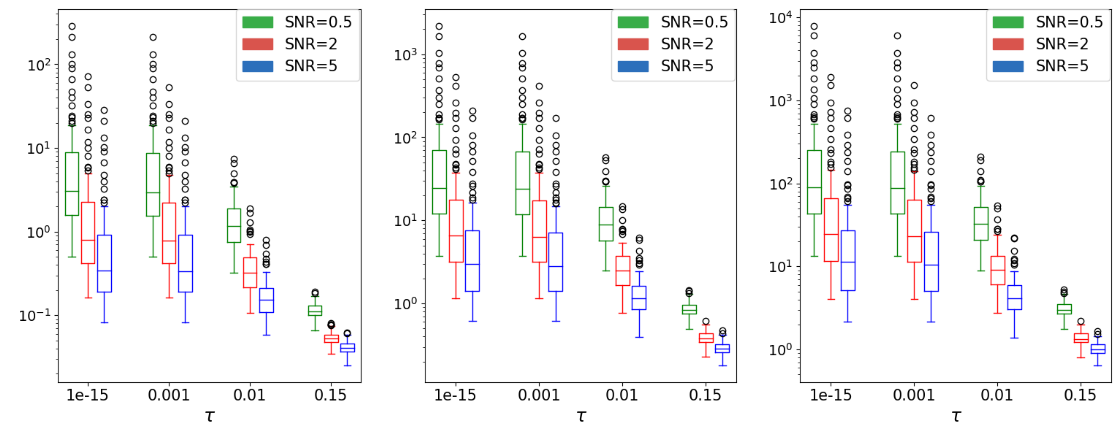


Figure 4.4: Boxplots of $\mathrm{Var}(s^{(0)})$ for different τ, where $s^{(0)}$ is of the form (4.34) and response functions (4.14) are used to compute $Y = (y(z_j))_{j=1}^{N}$ with $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (left) for various SNR with $r = 0.1$ and 100 design matrices M_+ of the form (4.2), where M is generated with $N = 100$ and $d = 100$.

be required before finding a subregion of the optimum. However, if some entries of the search direction $s^{(k)}$ with the largest magnitude contain the incorrect sign, then minimal movement will be made along $s^{(k)}$. Hence, $\tau = 0.15$ will be used for `numpy.linalg.pinv` when $N = d$ for numerical experiments in Section 4.7.3. Otherwise, the default $\tau = 1e - 15$ for `numpy.linalg.pinv` will be applied when $N \neq d$.

4.5.4 Summary

In this section, two modifications to the least-squares estimator $\hat{\theta}^{(k)}$ of $\nabla\eta(x^{(k)})$ as the choice of search direction are proposed for Phase I of BW. The purpose of modifications is to allow numerical comparisons with the new search direction proposed in (4.24) for large dimensions.

The first modification involves selecting a small subset of input variables for the computation of the least-squares estimator as opposed to selecting all input variables. Hence, a two-level fractional factorial design with fewer variables and observations is required, and non-zero entries of the search direction correspond to the input variables chosen. This method is denoted as the adjusted least-squares search direction. In addition, the algorithm for Phase I of BW with the adjusted least-squares search direction (PI-ALS) is presented in Subsection 4.5.2. Consequently, numerical experiments with PI-MY and PI-ALS will be conducted for large d and small N.

The second modification employs design matrices M proposed in Subsection 4.4.3

to replace the two-level fractional factorial designs for construction of M_+ in (4.2). In addition, the Moore-Penrose pseudoinverse is used instead of the multiplicative inverse to determine entries of the search direction. The algorithm for Phase I of BW with the Moore-Penrose psuedoinverse for computation of the search direction (PI-MPI) is presented in Subsection 4.5.3. Numerical experiments with PI-MY and PI-MPI will be conducted for large d and various N as design matrices M are generated according to the strategy in Subsection 4.4.3.

4.6 Selection of step length $\gamma^{(k)}$ for PI-MY, PI-ALS and PI-MPI

4.6.1 Overview

Throughout this chapter, a great deal of attention has been given to the choice of search direction $s^{(k)}$ for steepest descent iterations (4.5) within Phase I of BW for large dimensions. Recall that PI-MY, PI-ALS and PI-MPI are variations of Phase I of BW for large dimensions outlined in Sections 4.4 and 4.5, where different methods are used for computing the search direction $s^{(k)}$. A step length $\gamma^{(k)}$ is also required to compute steepest descent iterations (4.5). Therefore, the purpose of this section is to present the method that will be applied to compute $\gamma^{(k)}$ for steepest descent iterations (4.5) within PI-MY, PI-ALS and PI-MPI. The same method for computing $\gamma^{(k)}$ is applied to ensure a fair comparison with numerical experiments of PI-MY, PI-ALS and PI-MPI in Section 4.7.

4.6.2 Line search strategy to determine $\gamma^{(k)}$

The line search strategy for computing $\gamma^{(k)}$ consists of the following three steps

1. *Divide all entries of the search direction $s^{(k)}$ by the largest entry.*
   The magnitude of the entries of $s^{(k)}$ produced within PI-MY, PI-ALS and PI-MPI may differ significantly. Hence, the following update can be made to ensure the magnitude of entries of $s^{(k)}$ are similar for PI-MY, PI-ALS and PI-MPI,
   $$s^{(k)} = \frac{s^{(k)}}{\max\limits_{i=1,\ldots,d} |s_i^{(k)}|}. \tag{4.36}$$
   To initialize a line search strategy, an initial guess γ of the step length $\gamma^{(k)}$ is required. Since (4.36) is applied, the initial guess for the step length can be set as $\gamma = 1$ for PI-MY, PI-ALS and PI-MPI as all entries of $s^{(k)} \in [-1, 1]^d$.

If (4.36) is not applied, then the initial guess γ must be set according to the magnitude of entries of $s^{(k)}$. For example, if entries of $s^{(k)}$ are very large then a smaller value of γ is required. Otherwise, if entries of $s^{(k)}$ are very small, then a larger value of γ is necessary.

Recall that the response function evaluated with the initial guess γ is denoted as $y(\gamma)$ in (4.6), where y is a response function of the form (4.1). For all line search strategies, the first step usually involves evaluating the response function with $\gamma = 0$ and an initial guess $\gamma > 0$, to obtain $y(0)$ and $y(\gamma)$, respectively. If the magnitude of entries of $s^{(k)}$ are relatively small and the initial guess γ is also small, then the difference between response function values $y(0)$ and $y(\gamma)$ will be dictated by the error term in (4.1). As a consequence, the chosen step length $\gamma^{(k)}$ may result in limited movement along the search direction $s^{(k)}$, when in actual fact, additional movement along $s^{(k)}$ could improve the response function further. Updating $s^{(k)}$ according to (4.36) and choosing $\gamma = 1$ ensures that differences between response function values $y(0)$ and $y(\gamma)$ are influenced by the behaviour of the true function η and not just the error term in (4.1).

2. *Perform either forward or backward tracking*
   Forward and backward tracking is briefly discussed in Subsection 4.2.3, and involves repeatedly doubling or halving some initial guess γ of the step length until some stopping criterion is met. Specifically, forward tracking is applied if $y(\gamma) < y(0)$ and backward tracking is applied if $y(\gamma) \geq y(0)$. Forward or backward tracking is continued until two consecutive observations lead to an increase in the response function value. This stopping criterion is known as the two-in-a-row rule and is described in [89].

3. *Fit a second-order polynomial [34]*
   Suppose γ_{min} is the step length which produces the smallest response function value from applying forward or backward tracking, and consider,

$$\begin{aligned} \gamma_1 &= 0 \\ \gamma_2 &= \gamma_{min} \\ \gamma_3 &= 2\gamma_{min}. \end{aligned} \tag{4.37}$$

A second-order polynomial can be fitted with the following three points $(\gamma_1, y(\gamma_1))$, $(\gamma_2, y(\gamma_2))$ and $(\gamma_3, y(\gamma_3))$ (see [34]). Suppose γ^* is the minimizer of the second-order polynomial. Then the step length $\gamma^{(k)}$ is assigned as follows,

$$\gamma^{(k)} = \begin{cases} \gamma^*, & \text{if } y(\gamma^*) < y(\gamma_{min}) \\ \gamma_{min}, & \text{otherwise.} \end{cases}$$

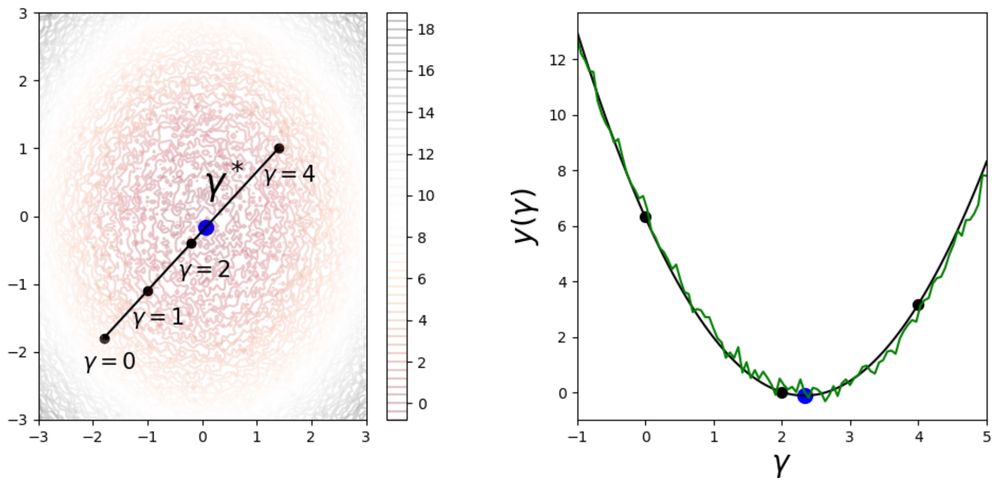


Figure 4.5: Contour plot of response function (4.14) (left) where $\epsilon \sim N(0, 0.0625)$, $\lambda_{min} = 1$ and $\lambda_{max} = 1$, with application of forward tracking to determine various γ (black). Optimal γ^* (blue) is computed by optimizing the fitted second-order polynomial. A plot of the fitted second-order polynomial with points (0, $y(0)$), (2, $y(2)$) and (4, $y(4)$) is presented (right). The minimizer of the second-order polynomial is highlighted in blue and the noisy response function values $y(\gamma)$ are plotted against various γ (green).

An example illustrating the proposed line search strategy for computing $\gamma^{(k)}$ is presented in Figure 4.5. Specifically, Figure 4.5 shows the application of forward tracking with a noisy response function (4.14) with $\epsilon \sim N(0, 0.0625)$, $\lambda_{min} = 1$ and $\lambda_{max} = 1$. Once forward tracking is terminated, a second-order polynomial is fitted with points $(\gamma_j, y(\gamma_j))$ for $j = 1, 2, 3$, where γ_j is assigned using (4.37). Figure 4.5 shows the second-order polynomial is fitted with points $(0, y(0))$, $(2, y(2))$ and $(4, y(4))$, and minimized to obtain γ^*. The purpose of fitting the second-order polynomial with points derived from forward or backward tracking is to allow the opportunity to reduce the response function value further. This is portrayed in Figure 4.5, where $\gamma^{(k)}$ is assigned the minimizer γ^* of the second-order polynomial.

4.7 Numerical experiments

4.7.1 Overview

Phase I of BW consists of iteratively applying steepest descent iterations (4.5) until a subregion of the optimum is reached. The search direction $s^{(k)}$ and step length $\gamma^{(k)}$ will need to be computed at each iteration k. Recall that PI-MY, PI-ALS and

PI-MPI are variations of Phase I of BW for large dimensions outlined in Sections 4.4 and 4.5, where different methods are used for computing the search direction $s^{(k)}$. The strategy for computing the step length $\gamma^{(k)}$ is outlined in Section 4.6 and applied for PI-MY, PI-ALS and PI-MPI. The purpose of this section is to investigate the performance of PI-MY, PI-ALS and PI-MPI.

To compare PI-MY, PI-ALS and PI-MPI, 100 response functions y_i $(i = 1, \dots, 100)$ of the form (4.14) and (4.15) are generated, with smallest and largest eigenvalues λ_{min} and λ_{max} respectively. All other eigenvalues are sampled uniformly from $(\lambda_{min}, \lambda_{max})$. For numerical experiments, $\lambda_{min} = 1$ and $\lambda_{max} = 1, 4, 8$. The effect on the performance of PI-MY, PI-ALS and PI-MPI with different λ_{max} is investigated. Recall that response functions (4.14) and (4.15) contain an error term $\epsilon \sim N(0, \sigma^2)$. The value of σ^2 is obtained by rearranging (4.17) for different values of SNR. For each response function y_i $(i = 1, \dots, 100)$, a random point $x^{(0)} \sim N(0, I_d)$ will also be generated. A small neighbourhood $R = (x_1^{(k)} \pm r, \dots, x_d^{(k)} \pm r)^T$ centred at $x^{(k)}$ with $r = 0.1$ is explored at each iteration $k = 0, \dots, K$.

In numerical experiments, $x_{MY}^{(K)}$, $x_{ALS}^{(K)}$, and $x_{MPI}^{(K)}$ are the final points found by applying PI-MY, PI-ALS and PI-MPI respectively. The performance of PI-MY, PI-ALS and PI-MPI is measured by observing the distances $\|x_{MY}^{(K)} - x^*\|$, $\|x_{ALS}^{(K)} - x^*\|$ and $\|x_{MPI}^{(K)} - x^*\|$. Furthermore, function values $\eta(x_{MY}^{(K)})$ and $\eta(x_{ALS}^{(K)})$ are also observed when comparing PI-MY and PI-ALS.

4.7.2 Comparison of PI-ALS and PI-MY

Design matrices are utilized at each iteration of PI-ALS and PI-MY to construct z_j in (4.3) and to obtain the response vector $Y = (y(z_j))_{j=1}^N$. For numerical experiments, the number of observations of a design matrix is set as $N = 16$. Thus, a 2^{10-6} fractional factorial design matrix (see [12, p.272] and [84, p.353]) will be used at each iteration of PI-ALS. To compute the least-squares estimator within PI-ALS, M_+ of the form (4.2) is constructed, where M is a 2^{10-6} fractional factorial design matrix. For each iteration of PI-MY, a design matrix M will be constructed according to the strategy outlined in Subsection 4.4.3.

PI-ALS will be applied for each response function y_i $(i = 1, \dots, 100)$ with random starting point $x^{(0)}$, and the total number of function evaluations to compute $\gamma^{(k)}$ and $s^{(k)}$ for all $k = 0, \dots, K$ will be observed for each y_i. Recall that PI-ALS involves selecting several random coordinates of $x^{(k)}$ and computing the least-squares estimator to determine the corresponding coefficients of a linear regression model. The significance of the linear regression model is tested by comparing the P-value of the F-test with a significance level of 0.1. If the linear regression model is insignificant then several other random coordinates are selected and the least-squares estimator is recomputed. PI-ALS terminates if all d co-ordinates have been explored and $x^{(k)}$ has not been updated due to insignificant linear regression models.

A value of c will need to be specified for the stopping criterion (4.27) of PI-MY. To ensure a fair comparison between PI-MY and PI-ALS, the value of c will be set as the average total number of function evaluations taken to compute $\gamma^{(k)}$ and $s^{(k)}$ for all $k = 0, \dots, K$ within PI-ALS, subtracted by the number of observations N of a design matrix. This ensures that an additional iteration of PI-MY is not performed if the number of function evaluations remaining is inadequate to compute the search direction $s^{(k)}$ in (4.24). Hence, function evaluations computed by PI-ALS and PI-MY will be similar, and consequently, the comparison of the performance of PI-MY and PI-ALS is fair. PI-MY will be applied with the same response functions y_i $(i = 1, \dots, 100)$ and random starting points $x^{(0)}$ as PI-ALS.

The performance of PI-MY and PI-ALS will be compared for different response functions y_i $(i = 1, \dots, 100)$ of the form (4.14) and (4.15), with $d = 10, 100$, $\lambda_{max} = 1, 4, 8$ and SNR$= 0.5, 1, 2, 3, 5, 10$.

Results for $d = 10$

In this example, the performance of PI-MY and PI-ALS is compared for $d = 10$, $N = 16$ and response functions y_i $(i = 1, \dots, 100)$ of the form (4.14) and (4.15), each with an initial random point $x^{(0)} \sim N(0, I_d)$. Since $d = 10$ and a 2^{10-6} fractional factorial design is used within PI-ALS, all coordinates of $x^{(k)}$ will be explored. In this case, PI-ALS is equivalent to the original Phase I of BW in Subsection 4.2.2. Specifically, search directions $s^{(k)} = (\hat{\theta}_1, ..., \hat{\theta}_d)^T$ consist of coefficients of the least-squares estimator.

		SNR					
Function	λ_{max}	0.5	1	2	3	5	10
(4.14)	1	0.66	0.47	0.33	0.27	0.21	0.15
	4	1.84	1.3	0.92	0.75	0.58	0.41
	8	3.51	2.48	1.76	1.43	1.11	0.79
(4.15)	1	0.1	0.07	0.05	0.04	0.03	0.02
	4	0.17	0.12	0.09	0.07	0.06	0.04
	8	0.25	0.17	0.12	0.1	0.08	0.06

Table 4.2: Corresponding σ for various SNR and λ_{max}, with $d = 10$ and y_i $(i = 1, \dots, 100)$ of the form (4.14) and (4.15).

Table 4.2 contains values of σ corresponding to various SNR and λ_{max}, with y_i

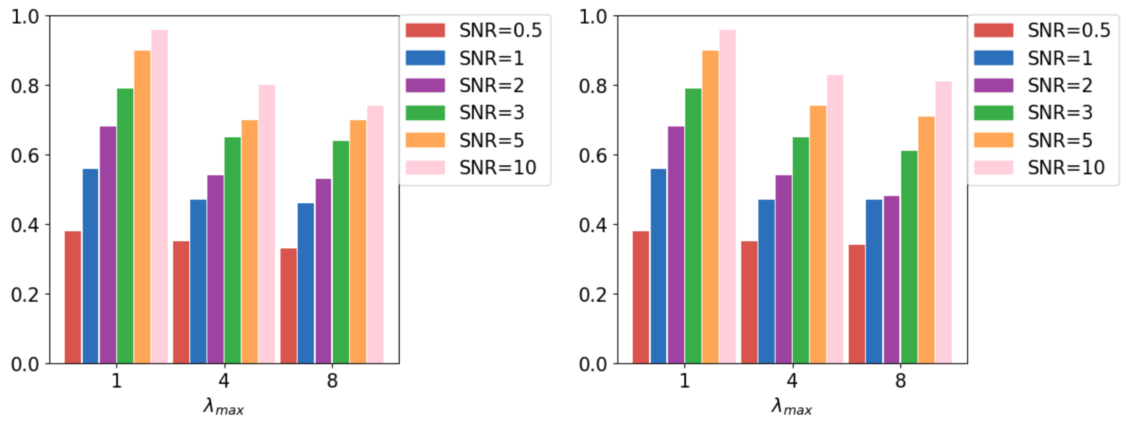


Figure 4.6: Proportion of points which satisfy $\|x_{MY}^{(K)} - x^*\| > \|x_{ALS}^{(K)} - x^*\|$ (left) and $\eta(x_{MY}^{(K)}) > \eta(x_{ALS}^{(K)})$ (right) for functions of the form (4.14) for various λ_{max} and SNR, with $\lambda_{min} = 1$, $N = 16$ and $d = 10$.

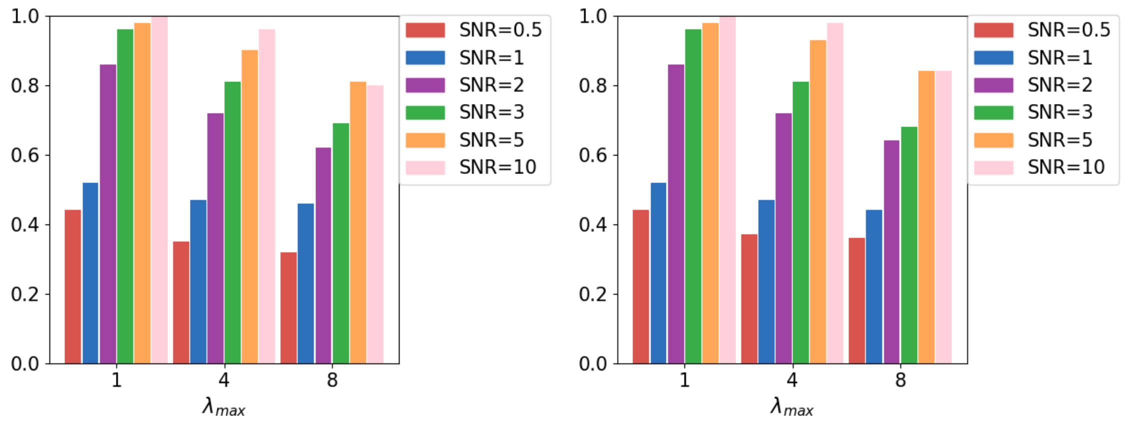


Figure 4.7: Proportion of points which satisfy $\|x_{MY}^{(K)} - x^*\| > \|x_{ALS}^{(K)} - x^*\|$ (left) and $\eta(x_{MY}^{(K)}) > \eta(x_{ALS}^{(K)})$ (right) for functions of the form (4.15) for various λ_{max} and SNR, with $\lambda_{min} = 1$, $N = 16$ and $d = 10$.

$(i = 1, \dots, 100)$ of the form (4.14) and (4.15) respectively. Table 4.2 shows that σ is largest for functions of the form (4.14) since values of $\eta(x^{(0)})$ are larger for functions (4.14) compared to functions (4.15). Figures 4.6 and 4.7 shows the proportion of points which satisfy $\eta(x_{MY}^{(K)}) > \eta(x_{ALS}^{(K)})$ and $\|x_{MY}^{(K)} - x^*\| > \|x_{ALS}^{(K)} - x^*\|$ for functions of the form (4.14) and (4.15), with $\lambda_{min} = 1$, $d = 10$, $N = 16$ for various λ_{max} and SNR.

Figures 4.6 and 4.7 show that for functions (4.14) and (4.15), the proportion of points which satisfy $\eta(x_{MY}^{(K)}) > \eta(x_{ALS}^{(K)})$ and $\|x_{MY}^{(K)} - x^*\| > \|x_{ALS}^{(K)} - x^*\|$ is greatest for large SNR and small λ_{max}. In this case, the least-squares estimator as the choice of search direction in PI-ALS is more effective than $s^{(k)}$ of the form (4.24) in PI-MY.

On the other hand, a smaller proportion of points satisfy $\eta(x_{MY}^{(K)}) > \eta(x_{ALS}^{(K)})$ and $\|x_{MY}^{(K)} - x^*\| > \|x_{ALS}^{(K)} - x^*\|$ when λ_{max} is large and SNR is small. However, search directions $s^{(k)}$ of the form (4.24) do not significantly enhance the performance of PI-MY in comparison to PI-ALS.

Results for $d = 100$

In this example, results of PI-MY and PI-ALS are compared with $d = 100$, $N = 16$ and response functions y_i ($i = 1, \dots, 100$) of the form (4.14) and (4.15), each with an initial random point $x^{(0)} \sim N(0, I_d)$. A subset containing $L = 10$ random coordinates of $x^{(k)}$ will be explored at the first stage of each iteration of PI-ALS since $d = 100$ and a 2^{10-6} fractional factorial design is used. If the linear regression model is significant, the search direction $s^{(k)}$ is of the form (4.29). Otherwise, if the linear regression model is insignificant, another subset containing $L = 10$ random coordinates of $x^{(k)}$ is selected and the process is repeated. Since $L = 10$ and $d = 100$, PI-ALS terminates if 10 subsets of random coordinates have been explored and no update has been made to $x^{(k)}$ due to insignificant linear regression models.

		SNR					
Function	λ_{max}	0.5	1	2	3	5	10
(4.14)	1	2.24	1.58	1.12	0.91	0.71	0.50
	4	6.22	4.40	3.11	2.54	1.97	1.39
	8	11.84	8.37	5.92	4.83	3.74	2.65
(4.15)	1	0.11	0.08	0.05	0.04	0.03	0.02
	4	0.19	0.14	0.10	0.08	0.06	0.04
	8	0.27	0.19	0.14	0.11	0.09	0.06

Table 4.3: Corresponding σ for various SNR and λ_{max}, with $d = 100$ and y_i ($i = 1, \dots, 100$) of the form (4.14) and (4.15).

Table 4.3 contains values of σ corresponding to various SNR and λ_{max}, with y_i ($i = 1, \dots, 100$) of the form (4.14) and (4.15) respectively. Similar to Table 4.2, values of σ are smallest for functions of the form (4.15).

Figures 4.8 and 4.10 show boxplots of the proportions $\|x_{ALS}^{(K)} - x^*\|/\|x^{(0)} - x^*\|$ and $\|x_{MY}^{(K)} - x^*\|/\|x^{(0)} - x^*\|$ for functions (4.14) and (4.15). Figures 4.9 and 4.11 show boxplots of the proportions $\eta(x_{ALS}^{(K)})/\eta(x^{(0)})$ and $\eta(x_{MY}^{(K)})/\eta(x^{(0)})$ for functions (4.14) and (4.15). Tables 4.4 and 4.6 show the average number of times $\eta(x^{(k)}) > \eta(x^{(k+1)})$ and the average total number of iterations K for PI-ALS and PI-MY with functions (4.14) and (4.15). Tables 4.5 and 4.7 portray the average total number of function evaluations and time taken (in seconds) by PI-ALS and PI-MY for functions

(4.14) and (4.15). The time taken is measured by monitoring the start and end time using `time.time()` within Python and taking the difference. Figures 4.8 - 4.11 and Tables 4.4 - 4.7 display results for various SNR and λ_{max}, with $\lambda_{min} = 1$, $N = 16$ and $d = 100$.

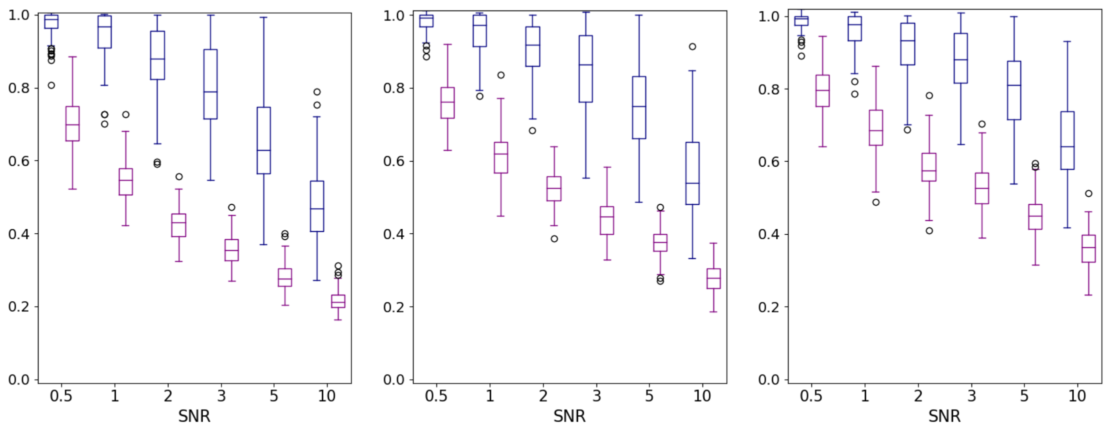


Figure 4.8: Boxplots of $\frac{\|x^{(K)}_{ALS}-x^*\|}{\|x^{(0)}-x^*\|}$ (blue) and $\frac{\|x^{(K)}_{MY}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various SNR, $\lambda_{min} = 1$, $N = 16$, $d = 100$ and y_i $(i = 1, \ldots, 100)$ of the form (4.14).

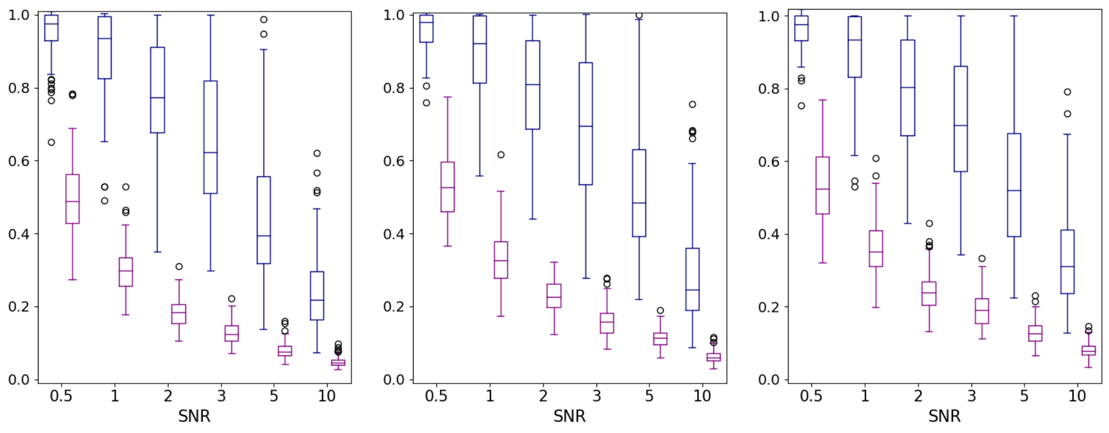


Figure 4.9: Boxplots of $\frac{\eta(x^{(K)}_{ALS})}{\eta(x^{(0)})}$ (blue) and $\frac{\eta(x^{(K)}_{MY})}{\eta(x^{(0)})}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various SNR, $\lambda_{min} = 1$, $N = 16$, $d = 100$ and y_i $(i = 1, \ldots, 100)$ of the form (4.14).

Figures 4.8 - 4.11 show that the median of proportions for $x^{(K)}_{ALS}$ is larger than those for $x^{(K)}_{MY}$ for all λ_{max} and SNR. This highlights that PI-MY outperforms PI-ALS for $d = 100$, $N = 16$ and response functions y_i $(i = 1, \ldots, 100)$ of the form (4.14) and (4.15).

The value of c for the stopping criterion (4.27) of PI-MY is set as the average

Metric	Type	λ_{max}	SNR					
			0.5	1	2	3	5	10
Average number of times $\eta(x^{(k)}) > \eta(x^{(k+1)})$	PI-ALS	1	1.84	3.28	5.91	8.23	12.66	18.96
		4	1.68	3.36	4.77	7.44	10.61	18.22
		8	1.66	2.94	4.80	6.17	9.86	15.03
	PI-MY	1	12.84	17.26	21.53	24.36	28.59	32.15
		4	11.79	17.41	19.02	22.98	25.72	32.46
		8	11.79	15.47	19.15	20.65	25.14	29.74
Average total number of iterations K	PI-ALS	1	3.70	5.17	7.52	9.74	14.41	21.25
		4	3.54	5.74	6.36	9.22	12.44	20.52
		8	3.73	5.05	6.50	7.65	11.36	17.11
	PI-MY	1	15.45	20.17	25.17	29.58	37.37	44.95
		4	14.55	21.61	22.38	28.79	33.35	46.74
		8	15.09	19.34	22.96	24.81	30.91	39.64

Table 4.4: Assess the quality of $s^{(k)}$ in PI-ALS and $s^{(k)}$ in PI-MY which improves the response function at each iteration k, for various λ_{max} and SNR, with $\lambda_{min} = 1$, $d = 100$, $N = 16$ and y_i $(i = 1, ..., 100)$ of the form (4.14).

Metric	Type	λ_{max}	SNR					
			0.5	1	2	3	5	10
Average total number of function evaluations	PI-ALS	1	371.45	479.09	603.41	725.21	955.88	1187.75
		4	358.99	536.28	537.82	717.70	847.99	1251.96
		8	378.14	478.95	558.14	605.84	768.87	1026.75
	PI-MY	1	371.87	479.93	602.34	725.84	957.32	1189.03
		4	356.46	534.45	536.70	717.70	849.20	1252.53
		8	378.08	478.51	558.16	603.57	768.26	1028.09
Average time taken (seconds)	PI-ALS	1	0.055	0.095	0.122	0.146	0.195	0.240
		4	0.071	0.107	0.107	0.145	0.170	0.267
		8	0.075	0.095	0.111	0.122	0.156	0.214
	PI-MY	1	0.065	0.074	0.098	0.113	0.153	0.176
		4	0.061	0.083	0.082	0.103	0.099	0.112
		8	0.062	0.073	0.093	0.093	0.116	0.135

Table 4.5: Average total number of function evaluations and time taken (seconds) by PI-ALS and PI-MY, for various λ_{max} and SNR, with $\lambda_{min} = 1$, $d = 100$, $N = 16$ and y_i $(i = 1, ..., 100)$ of the form (4.14).

number of function evaluations used within PI-ALS for each response function y_i $(i = 1, \dots, 100)$, subtracted by the number of observations N of a design matrix. As a result, it can be observed in Tables 4.5 and 4.7 that the average total number of function evaluations for PI-ALS and PI-MY is closely aligned. Furthermore, the average number of function evaluations decreases for small SNR values, which is due to the early termination of PI-ALS. As a consequence, the value of c will be small for the stopping criterion (4.27) and PI-MY will terminate early. Hence, an additional benefit of PI-MY is that the value of c can be increased, which will further

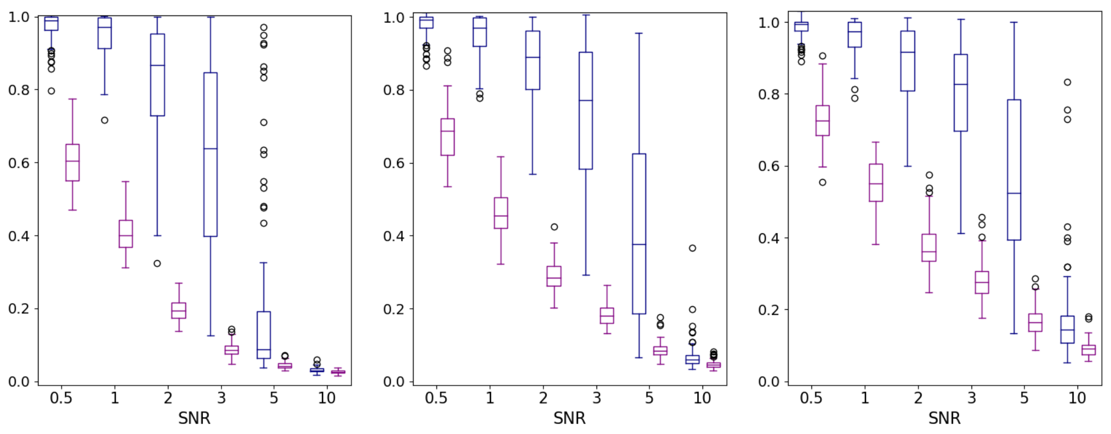


Figure 4.10: Boxplots of $\frac{\|x_{ALS}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (blue) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various SNR, $\lambda_{min} = 1$, $N = 16$, $d = 100$ and y_i $(i = 1, \dots, 100)$ of the form (4.15).

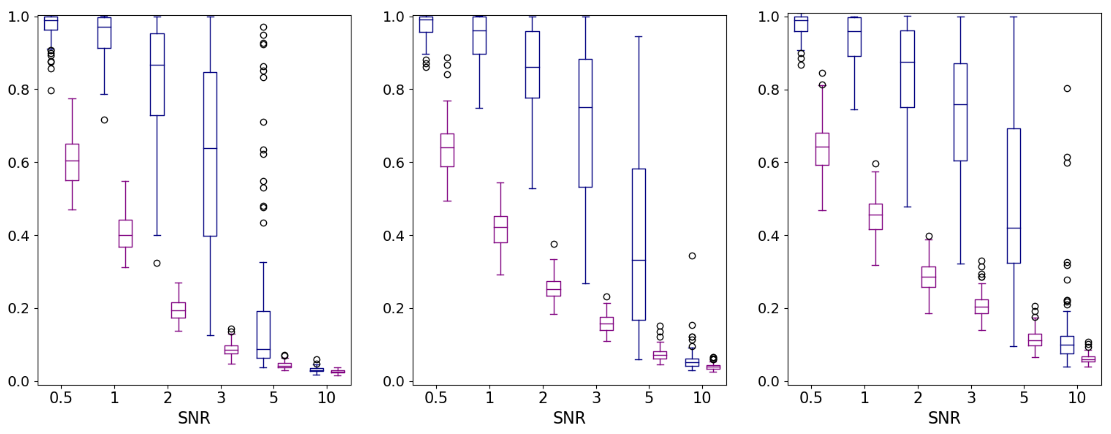


Figure 4.11: Boxplots of $\frac{\eta(x_{ALS}^{(K)})}{\eta(x^{(0)})}$ (blue) and $\frac{\eta(x_{MY}^{(K)})}{\eta(x^{(0)})}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various SNR, $\lambda_{min} = 1$, $N = 16$, $d = 100$ and y_i $(i = 1, \dots, 100)$ of the form (4.15).

enhance the performance of PI-MY. Tables 4.5 and 4.7 show that the average time taken by PI-MY is similar to PI-ALS for small SNR. However, the time taken by PI-ALS appears to be greater than PI-MY when SNR increases. This is due to the additional computations of the inverse within PI-ALS since the number of iterations of PI-ALS increases for large SNR, particularly for functions (4.15).

Tables 4.4 and 4.6 show that on average, the total number of iterations K is far larger for PI-MY than PI-ALS. This is due to the structure of the PI-ALS algorithm, which consists of selecting several random coordinates of $x^{(k)}$ and computing the least-squares estimator to obtain the coefficients of the linear regression model. If the linear regression model is insignificant, several other random coordinates are selected

Metric	Type	λ_{max}	SNR					
			0.5	1	2	3	5	10
Average number of times $\eta(x^{(k)}) > \eta(x^{(k+1)})$	PI-ALS	1	1.87	3.06	8.19	17.68	54.32	75.73
		4	1.76	3.18	6.74	12.49	32.65	75.30
		8	1.66	3.01	6.68	10.37	23.22	66.49
	PI-MY	1	15.17	20.09	31.09	45.65	61.62	70.26
		4	14.16	21.04	28.29	37.63	56.03	71.45
		8	14.02	19.02	28.11	34.25	49.31	70.96
Average total number of iterations K	PI-ALS	1	3.74	4.83	9.71	19.94	84.95	238.15
		4	3.66	5.25	8.38	14.06	38.88	172.69
		8	3.83	4.93	8.46	11.97	26.31	109.76
	PI-MY	1	16.52	20.72	32.37	52.94	152.75	369.31
		4	15.78	22.16	29.35	40.44	82.77	279.37
		8	16.12	20.51	29.48	36.39	62.03	187.04

Table 4.6: Assess the quality of $s^{(k)}$ in PI-ALS and $s^{(k)}$ in PI-MY which improves the response function at each iteration k, for various λ_{max} and SNR, with $\lambda_{min} = 1$, $d = 100$, $N = 16$ and y_i $(i = 1, ..., 100)$ of the form (4.15).

Metric	Type	λ_{max}	SNR					
			0.5	1	2	3	5	10
Average total number of function evaluations	PI-ALS	1	376.66	454.92	727.22	1304.55	4756.79	12494.55
		4	367.74	496.70	656.26	939.42	2276.14	9230.81
		8	381.03	463.06	667.11	837.33	1592.17	5905.33
	PI-MY	1	373.71	450.11	723.96	1303.98	4759.26	12496.18
		4	364.55	492.46	652.52	938.25	2277.32	9231.19
		8	379.98	461.52	665.34	834.82	1593.30	5906.27
Average time taken (seconds)	PI-ALS	1	0.054	0.078	0.111	0.257	1.045	2.659
		4	0.070	0.092	0.143	0.211	0.515	1.994
		8	0.082	0.100	0.150	0.189	0.367	1.316
	PI-MY	1	0.060	0.065	0.100	0.178	0.475	1.089
		4	0.068	0.080	0.107	0.154	0.301	1.136
		8	0.068	0.077	0.107	0.137	0.256	0.876

Table 4.7: Average total number of function evaluations and time taken (seconds) by PI-ALS and PI-MY, for various λ_{max} and SNR, with $\lambda_{min} = 1$, $d = 100$, $N = 16$ and y_i $(i = 1, ..., 100)$ of the form (4.15).

until either the linear regression model is significant, or until all d variables have been explored and no update has been made to $x^{(k)}$. Each time a linear regression model is constructed for a subset of random coordinates, N function evaluations are used. As a consequence, it is possible that a large number of function evaluations may be used at the k-th iteration of PI-ALS to obtain $s^{(k)}$ or to determine that $s^{(k)}$ cannot be constructed, which results in the termination of PI-ALS. For PI-MY, computation of $s^{(k)}$ of the form (4.24) uses N function evaluations at each iteration k. As a result, the total number of iterations K will be greater for PI-MY.

Tables 4.4 and 4.6 demonstrate that on average, $\eta(x^{(k)}) > \eta(x^{(k+1)})$ is greatest when PI-MY is applied for various λ_{max} and SNR with functions (4.14) and (4.15), with the exception of SNR $= 10$ and small λ_{max} for functions of the form (4.15). However, it can be observed in Figures 4.10 and 4.11 that PI-MY finds a reasonable approximation of the minimizer x^* for SNR $= 10$ and small λ_{max}. Since the total number of iterations K can be large for PI-MY, there may be some instances where $\eta(x^{(k)}) = \eta(x^{(k+1)})$. This can occur when a reasonable approximation $x^{(k)}$ of x^* has already been discovered, and as a consequence, it may be difficult to improve $\eta(x^{(k)})$ and $\|x^{(k)} - x^*\|$ any further.

4.7.3 Comparison of PI-MPI and PI-MY

The performance of PI-MPI and PI-MY will be compared for each response function y_i $(i = 1, \ldots, 100)$ of the form (4.14) and (4.15) with a random starting point $x^{(0)}$, for $N = 16, 32, 50, 100, 200$, $\lambda_{max} = 1, 4, 8$, SNR $= 0.5, 2$ and $d = 100$. To enhance the difficulty of numerical experiments, smaller values of SNR are used to increase the variance σ^2 of the error for response function values (4.14) and (4.15). Design matrices M are utilized at each iteration of PI-MPI and PI-MY to construct z_j in (4.3) and to obtain the response vector $Y = (y(z_j))_{j=1}^{N}$. For each iteration of PI-MPI and PI-MY, design matrices M will be constructed according to the strategy outlined in Subsection 4.4.3.

Recall that a value of c is required for the stopping criterion (4.27) of PI-MPI and PI-MY. The value of c can greatly effect the efficiency and accuracy of PI-MPI and PI-MY. If c is set to a small number, it may be difficult to find a subregion of the minimizer. Conversely, if c is set to a large value, progress may be limited after a particular number of iterations, and efficiency may be compromised. The value assigned to c may depend on the computational cost of response function evaluations. That is, if response function evaluations are computationally expensive, then a relatively small value for c will be selected. However, if the response function evaluations are inexpensive to compute, then c may be set to a larger value. To account for both scenarios, $c = 500, 2000$ is considered for numerical comparisons, and the effect of N on the performance of PI-MPI and PI-MY will be investigated for each value of c.

The PI-MPI algorithm, presented in Subsection 4.5.3, uses the Moore-Penrose pseudoinverse to compute the least-squares estimator. The application of the Moore-Penrose pseudoinverse is investigated for design matrices M_+ of the form (4.2), where M is constructed according to the strategy outlined in Subsection 4.4.3 with different N and d. Specifically, singular values of M_+ which are greater than some tolerance are selected in (4.30) for computation of the Moore-Penrose pseudoinverse. Figure 4.3 shows the singular values selected in (4.30) for computation of the Moore-Penrose pseudoinverse are not effected by the value of τ when $N \neq d$. Conversely, Figures 4.3 and 4.4 show that for $N = d$, the value of τ influences the selection of singular values in (4.30) used for computation of the Moore-Penrose pseudoinverse. Henceforth, it is proposed to set $\tau = 0.15$ for $N = d$ and $\tau = 1e - 15$ for $N \neq d$ when the Moore-Penrose pseudoinverse is used to compute $s^{(k)}$ of the form (4.34) within PI-MPI.

Figures 4.12 - 4.19 show boxplots of the proportions $\|x_{MPI}^{(K)} - x^*\| / \|x^{(0)} - x^*\|$ and $\|x_{MY}^{(K)} - x^*\| / \|x^{(0)} - x^*\|$ for various c, N, SNR and λ_{max}, with $d = 100$, $\lambda_{min} = 1$ and y_i $(i = 1, \dots, 100)$ of the form (4.14) and (4.15). Tables 4.8 - 4.15 show the average total number of function evaluations and time taken (seconds) by PI-MPI and PI-MY to obtain $x_{MPI}^{(K)}$ and $x_{MY}^{(K)}$ in Figures 4.12 - 4.19. The time taken is measured by monitoring the start and end time using `time.time()` within Python and taking the difference.

Results for different c with functions (4.14)

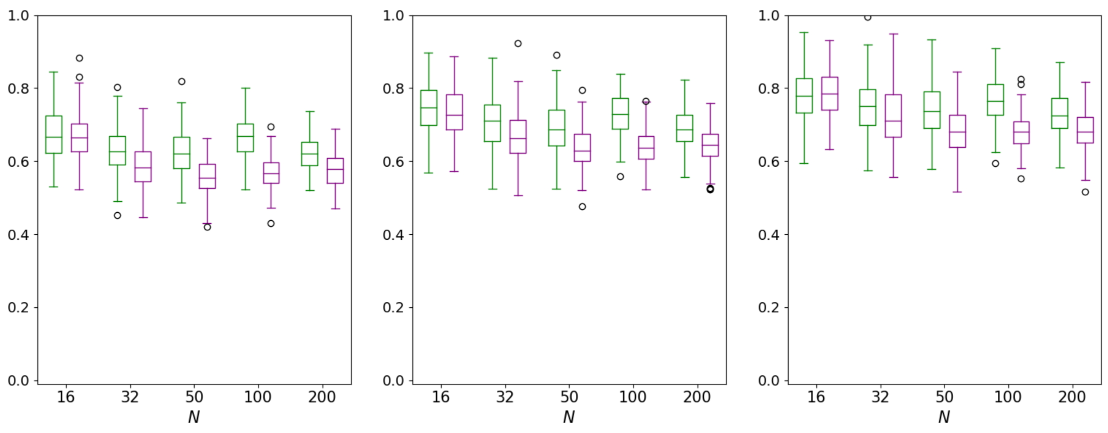


Figure 4.12: Boxplots of $\frac{\|x_{MPI}^{(K)} - x^*\|}{\|x^{(0)} - x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)} - x^*\|}{\|x^{(0)} - x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 0.5$ and y_i $(i = 1, \dots, 100)$ of the form (4.14), where $c = 500$.

Figures 4.12 - 4.15 show that PI-MY performs better than PI-MPI with the combinations, $N = 50$ with $c = 500$ and $N = 200$ with $c = 2000$, for all λ_{max} and SNR. Recall that N function evaluations are used to compute $s^{(k)}$ at each iteration of steepest descent (4.5) within PI-MY and PI-MPI. Therefore, only a small number of steepest descent iterations (4.5) will be permitted when N is large and c is small. However, the search directions $s^{(k)}$ are likely to be more accurate when N

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	498.99	489.79	484.30	417.12	408.99
		4	499.16	490.63	480.80	417.53	408.79
		8	498.59	490.85	481.05	417.76	408.78
	2	1	496.57	486.64	487.61	417.55	409.09
		4	497.58	486.47	487.22	417.58	409.02
		8	497.52	488.30	485.56	417.79	409.01
PI-MY	0.5	1	499.22	490.71	485.30	417.05	409.00
		4	498.88	490.09	483.82	417.73	408.89
		8	499.47	488.81	482.56	417.99	408.82
	2	1	498.37	485.81	490.87	417.12	409.12
		4	499.18	487.46	490.13	417.08	408.99
		8	497.91	487.62	490.32	417.21	408.96

Table 4.8: Average number of function evaluations for PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 500$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.14).

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	0.052	0.065	0.077	0.064	0.064
		4	0.102	0.089	0.086	0.069	0.054
		8	0.085	0.078	0.079	0.070	0.067
	2	1	0.078	0.076	0.078	0.083	0.069
		4	0.099	0.080	0.072	0.068	0.058
		8	0.084	0.079	0.079	0.068	0.064
PI-MY	0.5	1	0.090	0.072	0.067	0.050	0.047
		4	0.071	0.061	0.061	0.040	0.044
		8	0.077	0.068	0.068	0.053	0.040
	2	1	0.081	0.073	0.065	0.050	0.046
		4	0.078	0.065	0.060	0.049	0.044
		8	0.067	0.064	0.066	0.047	0.027

Table 4.9: Average time taken (seconds) by PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 500$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.14).

is large. Setting $N = d/2$ when $c = 500$ improves the performance of PI-MY since a suitable balance between the number of steepest descent iterations (4.5) performed and the quality of search directions $s^{(k)}$ is achieved. However, setting $N > d$ when $c = 500$ results in a similar or better performance of PI-MPI in comparison to setting

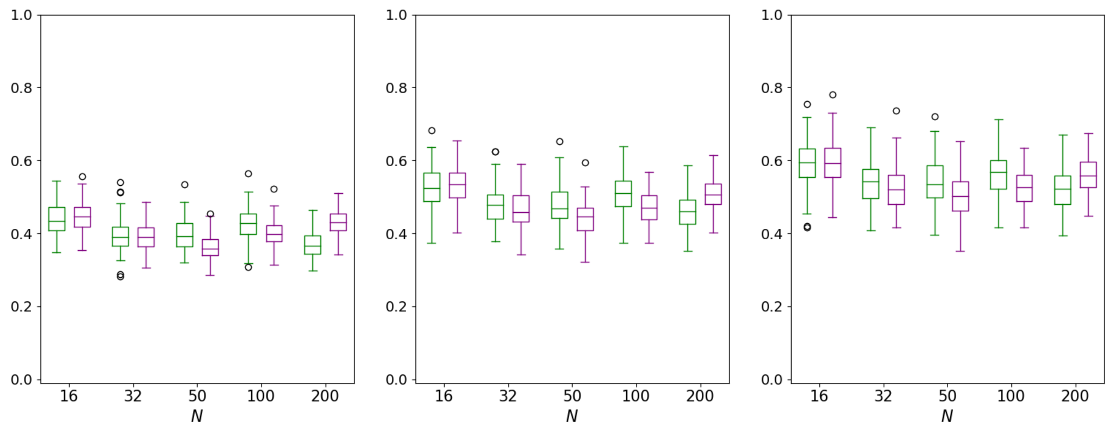


Figure 4.13: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 2$ and y_i $(i = 1, \ldots, 100)$ of the form (4.14), where $c = 500$.

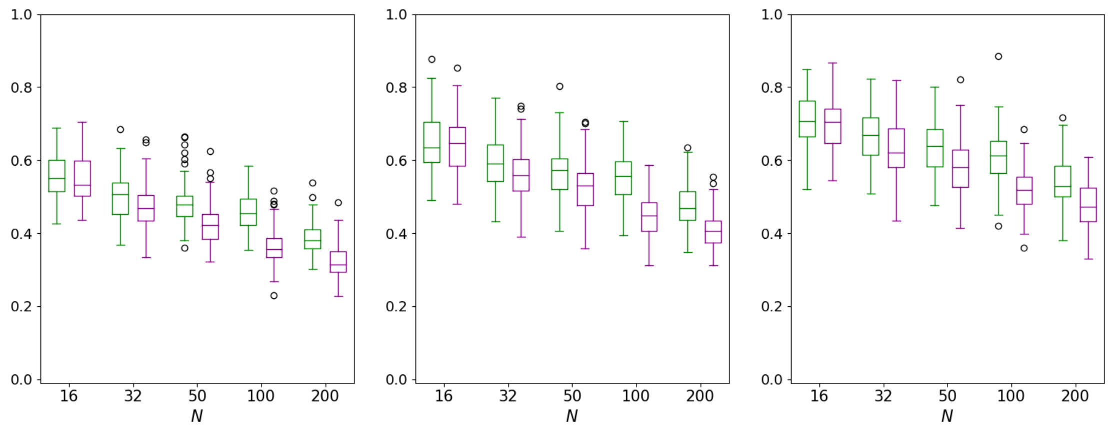


Figure 4.14: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 0.5$ and y_i $(i = 1, \ldots, 100)$ of the form (4.14), where $c = 2000$.

$N = d/2$. Generating design matrices M with $N > d$ improves the performance of PI-MY and PI-MPI when $c = 2000$. This is because a large number of steepest descent iterations (4.5) are permitted when $c = 2000$, even for large N, which allows the opportunity to find the subregion of the optimum.

Setting c to a small value provides a snapshot of the progress made during the early stages of PI-MY in finding the subregion of the optimum. Figures 4.12 and 4.13 show that setting $N = d/2$ allows more progress in finding the subregion of the optimum during the early stages of PI-MY. Therefore, it may be more effective to generate design matrices M with $N = d/2$ at the beginning of PI-MY, and increase N as the number of iterations grow. Adopting the discussed strategy appears to be

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	2000.93	1991.84	1982.98	1966.49	1862.13
		4	2000.17	1990.71	1982.29	1961.56	1863.85
		8	1998.75	1990.02	1984.10	1958.07	1866.52
	2	1	2002.29	1994.10	1983.16	1963.59	1876.29
		4	2000.55	1991.78	1983.14	1965.43	1871.92
		8	2000.91	1991.85	1981.86	1966.37	1871.99
PI-MY	0.5	1	2000.31	1993.55	1982.49	1961.80	1864.02
		4	2001.72	1994.49	1978.73	1969.12	1866.68
		8	2001.55	1993.98	1977.24	1968.96	1867.22
	2	1	2004.59	1993.76	1985.02	1966.92	1870.07
		4	2001.87	1991.42	1977.53	1966.25	1866.28
		8	2000.91	1992.83	1978.21	1966.59	1867.12

Table 4.10: Average number of function evaluations for PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 2000$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.14).

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	0.260	0.296	0.315	0.307	0.275
		4	0.322	0.315	0.328	0.357	0.260
		8	0.332	0.308	0.356	0.383	0.321
	2	1	0.329	0.309	0.307	0.315	0.271
		4	0.314	0.432	0.429	0.446	0.289
		8	0.319	0.301	0.326	0.338	0.287
PI-MY	0.5	1	0.305	0.274	0.260	0.253	0.232
		4	0.305	0.272	0.249	0.225	0.212
		8	0.287	0.266	0.253	0.231	0.218
	2	1	0.302	0.272	0.255	0.232	0.227
		4	0.256	0.255	0.236	0.220	0.201
		8	0.306	0.290	0.239	0.224	0.201

Table 4.11: Average time taken (seconds) by PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 2000$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.14).

the most effective use of response function evaluations to improve the performance of PI-MY.

Tables 4.8 and 4.10 show the close alignment of function evaluations used by PI-MPI and PI-MY for various N, SNR and λ_{max}. In addition, Tables 4.9 and 4.11

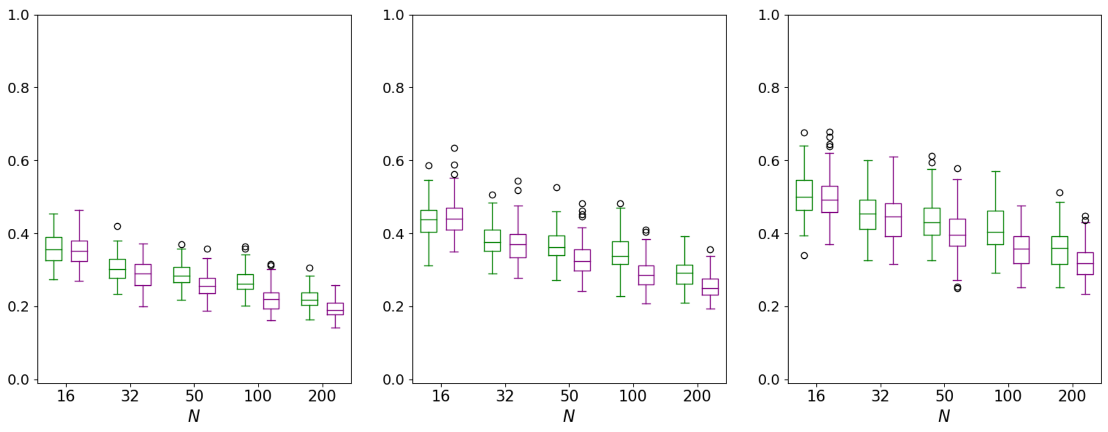


Figure 4.15: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 2$ and y_i $(i = 1, \dots, 100)$ of the form (4.14), where $c = 2000$.

show that the average time taken is similar for PI-MY and PI-MPI for various N, SNR and λ_{max}.

Results for different c with functions (4.15)

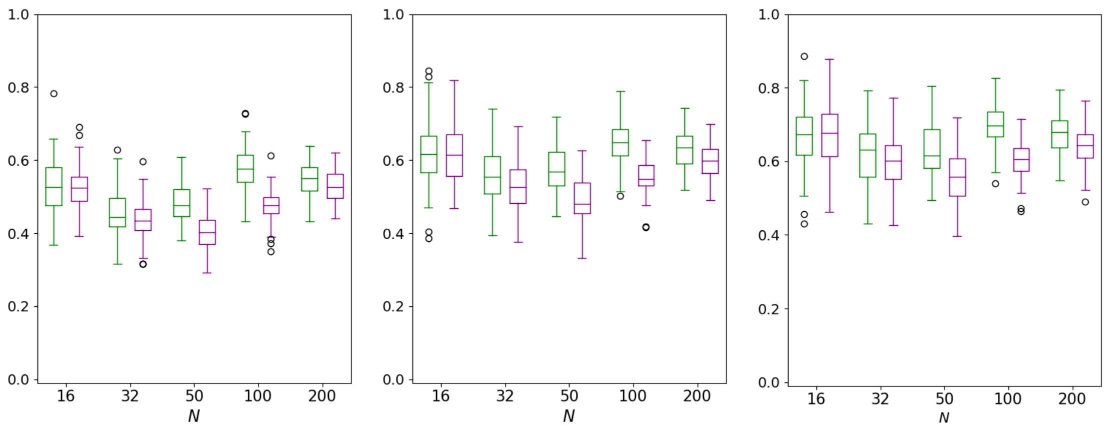


Figure 4.16: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 0.5$ and y_i $(i = 1, \dots, 100)$ of the form (4.15), where $c = 500$.

Figures 4.16 and 4.18 show that PI-MY performs better than PI-MPI with the combinations, $N = 50$ with $c = 500$ and $N = 200$ with $c = 2000$, for all λ_{max} and SNR $= 0.5$. However, a different trend is apparent for SNR $= 2$. That is, Figure 4.17 illustrates that PI-MPI performs better than PI-MY for all N and λ_{max} with $c = 500$. On the other hand, Figure 4.19 shows the performance of PI-MPI and PI-

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	497.12	478.14	489.37	416.51	409.24
		4	498.36	483.36	488.93	416.60	408.99
		8	497.10	484.14	487.31	416.69	408.90
	2	1	493.39	477.38	490.92	417.40	409.13
		4	495.13	477.92	490.56	417.15	409.07
		8	496.04	479.28	491.02	417.11	409.03
PI-MY	0.5	1	497.25	479.23	489.34	416.67	409.15
		4	496.27	481.80	489.03	416.63	408.96
		8	496.91	484.86	488.61	416.67	408.89
	2	1	493.46	476.29	489.85	417.10	409.20
		4	493.99	477.03	489.97	416.98	409.02
		8	494.23	478.43	490.51	416.96	408.97

Table 4.12: Average number of function evaluations for PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 500$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.15).

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	0.061	0.074	0.085	0.055	0.064
		4	0.104	0.084	0.101	0.106	0.054
		8	0.097	0.099	0.091	0.088	0.074
	2	1	0.087	0.082	0.083	0.084	0.070
		4	0.115	0.106	0.093	0.081	0.067
		8	0.103	0.085	0.079	0.086	0.070
PI-MY	0.5	1	0.100	0.081	0.078	0.066	0.059
		4	0.052	0.079	0.068	0.057	0.055
		8	0.088	0.079	0.078	0.056	0.054
	2	1	0.092	0.077	0.074	0.054	0.049
		4	0.051	0.078	0.069	0.056	0.051
		8	0.091	0.071	0.073	0.053	0.047

Table 4.13: Average time taken (seconds) by PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 500$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.15).

MY is very similar for all N when SNR $= 2$ and $c = 2000$. Therefore, the additional steepest descent iterations (4.5) permitted when $c = 2000$ are extremely effective during the latter stages of PI-MY, and ensures that the performance of PI-MY is closely aligned with PI-MPI when SNR $= 2$.

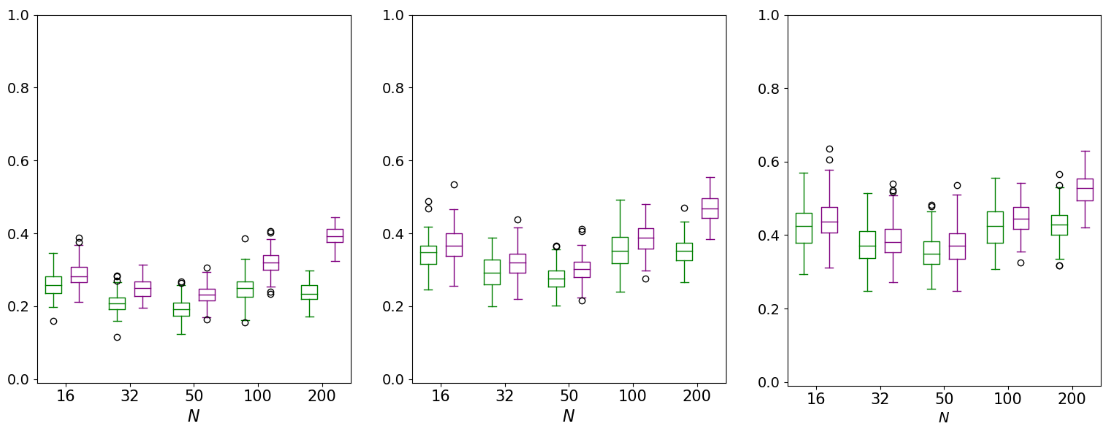


Figure 4.17: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 2$ and y_i $(i = 1, \ldots, 100)$ of the form (4.15), where $c = 500$.

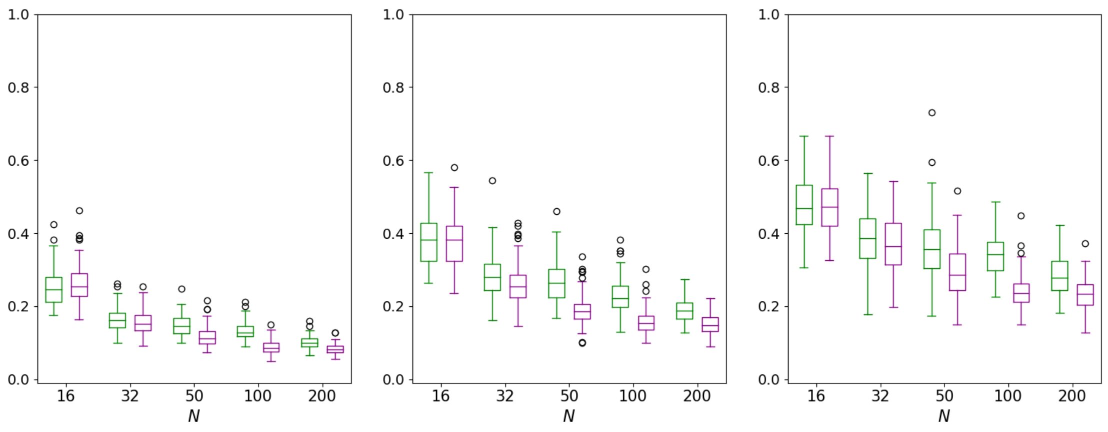


Figure 4.18: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 0.5$ and y_i $(i = 1, \ldots, 100)$ of the form (4.15), where $c = 2000$.

Figures 4.16 - 4.19 illustrate that the most effective use of response function evaluations is to construct design matrices M with number of observations $N = d/2$ during the early stages of PI-MY, and increase N as the number of iterations grow. This trend is also portrayed in Figures 4.12 - 4.15.

The key trends in Tables 4.12 - 4.15 are similar to Tables 4.8 - 4.11. That is, the total number of function evaluations and time taken by PI-MPI and PI-MY is similar for various N, SNR and λ_{max}.

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	2000.46	1992.48	1982.33	1940.54	1848.16
		4	2001.29	1993.54	1984.39	1939.84	1845.38
		8	2002.07	1995.04	1982.59	1940.00	1846.26
	2	1	2002.12	1994.16	1983.70	1971.93	1872.70
		4	2001.31	1992.28	1985.26	1948.29	1860.91
		8	2000.20	1993.15	1983.51	1941.33	1856.02
PI-MY	0.5	1	2000.57	1993.44	1983.53	1936.79	1849.63
		4	1999.86	1991.76	1981.97	1939.75	1848.36
		8	1999.19	1993.16	1982.09	1941.54	1848.20
	2	1	1998.94	1991.90	1986.20	1951.35	1855.72
		4	1999.81	1992.91	1983.66	1940.65	1851.91
		8	2000.60	1991.31	1980.76	1936.29	1850.79

Table 4.14: Average number of function evaluations for PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 2000$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.15).

Type	SNR	λ_{max}	N				
			16	32	50	100	200
PI-MPI	0.5	1	0.284	0.323	0.334	0.339	0.307
		4	0.369	0.309	0.261	0.341	0.284
		8	0.356	0.338	0.336	0.219	0.307
	2	1	0.366	0.347	0.354	0.340	0.293
		4	0.342	0.321	0.277	0.390	0.281
		8	0.357	0.322	0.326	0.199	0.283
PI-MY	0.5	1	0.351	0.315	0.299	0.238	0.216
		4	0.435	0.304	0.274	0.238	0.223
		8	0.270	0.309	0.277	0.151	0.227
	2	1	0.358	0.316	0.299	0.362	0.210
		4	0.281	0.280	0.268	0.235	0.231
		8	0.187	0.278	0.295	0.144	0.222

Table 4.15: Average time taken (seconds) by PI-MPI and PI-MY, with various N, λ_{max} and SNR for $c = 2000$, $d = 100$, $\lambda_{min} = 1$ and functions of the form (4.15).

4.7.4 Conclusions

Phase I of BW consists of iteratively applying steepest descent iterations (4.5) until a subregion of the optimum is reached. The search direction $s^{(k)}$ and step length $\gamma^{(k)}$ will need to be computed for each iteration of steepest descent (4.5). The strategy

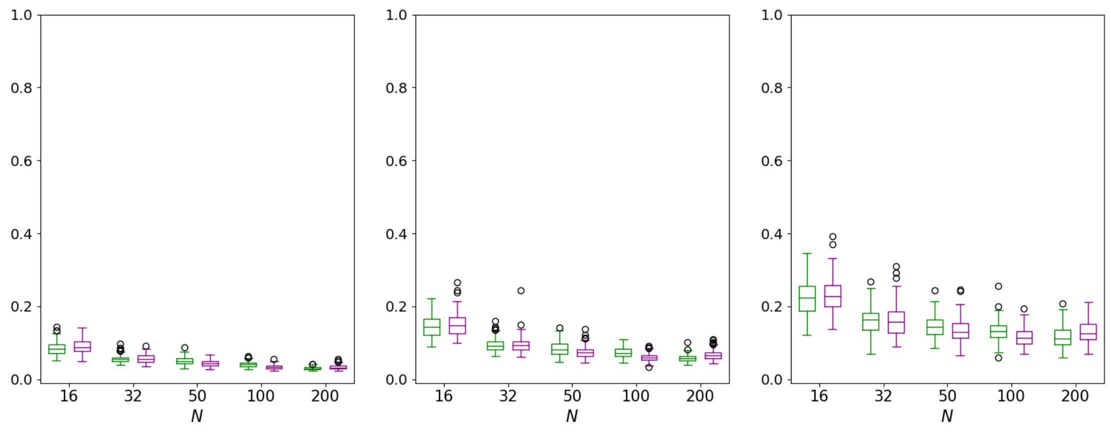


Figure 4.19: Boxplots of $\frac{\|x_{MPI}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (green) and $\frac{\|x_{MY}^{(K)}-x^*\|}{\|x^{(0)}-x^*\|}$ (purple), for $\lambda_{max} = 1$ (left), $\lambda_{max} = 4$ (centre) and $\lambda_{max} = 8$ (right) for various N, $\lambda_{min} = 1$, $d = 100$, SNR $= 2$ and y_i $(i = 1, \dots, 100)$ of the form (4.15), where $c = 2000$.

for computing the step length $\gamma^{(k)}$ in Phase I of BW is not challenged in this chapter. The purpose of numerical experiments is to measure the performance of Phase I of BW with different search directions $s^{(k)}$, namely, PI-MY, PI-ALS, and PI-MPI.

The performance of PI-MY and PI-ALS is compared with various SNR and λ_{max}, for functions (4.14) and (4.15). For numerical experiments with PI-MY and PI-ALS , the number of observations of a design matrix is set as $N = 16$ and the dimensions tested are $d = 10, 100$. Hence, a 2^{10-6} fractional factorial design matrix (see [12, p.272] and [84, p.353]) is used at each iteration of PI-ALS. To compute the least-squares estimator within PI-ALS, M_+ of the form (4.2) is constructed, where M is a 2^{10-6} fractional factorial design matrix. In addition, a design matrix M is constructed according to the strategy outlined in Subsection 4.4.3 with $N = 16$ and $d = 10, 100$ for each iteration of PI-MY. For $d = 10$, the performance of PI-ALS is either comparable or better than PI-MY for functions (4.14) and (4.15). However, the main focus throughout this chapter is to investigate the performance PI-MY for large dimensions. Figures 4.12 - 4.19 and Tables 4.8 - 4.14 clearly show that PI-MY outperforms PI-ALS for all SNR and λ_{max} when $d = 100$. Hence, the choice of search direction $s^{(k)}$ in (4.24) for PI-MY is far more effective than $s^{(k)}$ in (4.29) for PI-ALS when the dimension is large.

The performance of PI-MY and PI-MPI is also compared for functions (4.14) and (4.15), with various SNR and λ_{max}. To enhance the difficulty of numerical experiments, smaller values of SNR are used to increase the variance σ^2 of the error for response function values (4.14) and (4.15). Design matrices M for PI-MY and PI-MPI are constructed according to the strategy outlined in Subsection 4.4.3 for $d = 100$ and different values of N. For PI-MPI, an additional first column of ones is added to design matrices M to obtain M_+ (4.2). Recall that a value c is required for the stopping criterion (4.27) of PI-MY and PI-MPI. For numerical experiments, $c = 500, 2000$ is considered and the effect of N on the performance of PI-MY and PI-MPI is investigated for each value of c.

Two key trends are observed from the performance of PI-MY and PI-MPI for different c and N. Firstly, if c is set to a relatively small value then selecting $N = d/2$ improves the performance of PI-MY. Secondly, if c is set to a large value then selecting $N \geq d$ enhances the performance of PI-MY and PI-MPI. Both trends are observed for functions (4.14) and (4.15) with various SNR and λ_{max}. The value of N which improves the performance of PI-MPI for relatively small c depends on the type of function (4.14) and (4.15). In addition, PI-MY with suitably chosen N outperforms PI-MPI for functions (4.14) with different values of SNR, λ_{max} and c. Furthermore, PI-MY outperforms PI-MPI for functions (4.15) when N is chosen appropriately for SNR $= 0.5$ with various λ_{max} and c. A different trend is apparent for functions (4.15) with SNR $= 2$. Nevertheless, PI-MPI and PI-MY perform similarly provided c is relatively large when SNR $= 2$.

In conclusion, search directions of the form (4.24) within PI-MY can significantly enhance the performance of Phase I of BW for large dimensions, particularly when errors of the response are large.

4.8 Summary

RSM is a collection of methods for approximating a minimizer of a regression function. The most often used and cited RSM strategy is the so-called Box-Wilson (BW) algorithm. The BW algorithm consists of two phases, and the focus of this chapter is on Phase I of BW. Phase I of BW consists of iteratively applying steepest descent iterations (4.5) until a subregion of the optimum is reached. The search direction $s^{(k)}$ and step length $\gamma^{(k)}$ will need to be computed for each iteration of steepest descent (4.5). In the RSM-related literature, the least-square estimator of the gradient of the response function at a current point is routinely suggested as $s^{(k)}$. However, this chapter presents a new search direction $s^{(k)}$ of the form (4.24). Two modifications to the least-squares estimator as the choice of search direction are proposed for Phase I of BW, to allow numerical comparisons with the new search direction (4.24) for large dimensions. Numerical comparisons show that the new search direction (4.24) can significantly improve Phase I of BW and RSM, in general, for high-dimensional problems, especially when errors of the response are large.

Chapter 5

Software Development

This chapter is summarized as follows.

- Section 5.1 demonstrates the importance of producing high-quality software. In addition, an overview of the measures taken to ensure the development of high-quality software for Chapters 3 and 4 are presented, where further discussion of each measure is outlined in the following sections.
- Section 5.2 verifies that software employs a version control system and is accessible.
- In Section 5.3, measures taken to ensure reproducibility of results are presented.
- Section 5.4 outlines the purpose of testing software.
- In Section 5.5, the test coverage for software is presented.
- Section 5.6 describes how continuous integration has been employed for software.
- In Section 5.7, the purpose of documentation is discussed, along with a variety of rules to consider during the creation and development of documentation. Full documentation is available for all software developed.

5.1 Overview

There have been significant advancements in technology and storage capabilities in recent times, which has enabled considerable progression in a wide variety of research fields. As a result, it has become increasingly important to use software as a tool to carry out cumbersome operations accurately and promptly, to investigate developments in a particular research area. Throughout this chapter, code refers to a collection of executable instructions, a program refers to a piece of code used to carry out a particular task, and software refers to a collection of programs used together to obtain outputs. The terms software, programs and code will be used throughout when discussing best practices for software development.

It has become customary for researchers to develop research specific software that involves high-level expertise and knowledge since available software may be irrelevant, impractical to apply or non-existent. However, many researchers who develop software may have minimal programming experience and may not be aware of software development best practices. Furthermore, if research outputs are required to meet a deadline, time may not be spent ensuring accurate or well-documented software. However, producing high-quality software provides many benefits, which are outlined as follows. Software that can be successfully executed and tested provides confidence that outputs generated are accurate. Furthermore, other researchers may be inclined to use the software, which may present opportunities for collaboration, resulting in possible improvements in the capabilities and efficiency of the software. The inclusion of comments and documentation allows simple modifications to be made by the developer and other researchers. As a result, more time can be spent on the inspection and analysis of results to facilitate advancements in a specific research area.

A great deal of literature is devoted to providing advice and recommendations to improve software [7, 55, 69, 100, 111, 136]. Specifically, [7] highlights the following key requirements which should be adopted for software to contribute towards science.

- *Re-runnable*
  Developers must be able to repeatedly run code, which may not be possible if upgrades are made to the system, programming language, libraries and software.
- *Repeatable*
  The developer should be able to regenerate the same results each time the code is executed.
- *Reproducible*
  Other researchers should be able to run the code and reproduce the same results.
- *Reusable*
  If programs are readable, structured and documented, other researchers are able to inspect, run and modify programs.
- *Replicable*
  The description of software within literature should be complemented by the

code, documentation and implementation details to replicate results. This provides other researchers with enough information to re-write the software and obtain the same results.

Measures	Description
Version control and accessibility [7, 55, 100, 111, 136]	A repository is a location where software can be developed, maintained and made publicly available through a hosting website. A version control system allows changes to be tracked and displayed on a hosting website.
Reproduce results [7, 111]	The ability to repeatedly run code and obtain the same outputs provides confidence in the software's functionality.
Testing [7, 100, 136]	It is vital that testing is undertaken to confirm the accuracy of code.
Coverage [100]	Although testing is an essential part of developing software, it must be ensured that tests check all lines of code used to generate outputs. The number of lines of code executed by tests is checked using a coverage tool.
Continuous integration [100]	A continuous integration engine verifies that software can be built successfully from a list of requirements. This is essential for other researchers to be able to install and use the software. Furthermore, continuous integration can be integrated with other services to display a variety of key benchmarks that evaluate aspects of the software, including the percentage of test coverage and status of documentation.
Documentation [7, 69, 100, 136]	Clear and descriptive documentation explaining how the software works will encourage other people to inspect, apply and even modify the software. This can be in the form of comments within the code or embedded documentation explaining full details on installing and running the software.

Table 5.1: Measures taken during the development of software.

Table 5.1 contains a hybrid of recommendations from [7, 55, 69, 100, 111, 136] that have been implemented during the development of software to meet the re-runnable, repeatable, reproducible, reusable and replicable requirements in [7]. Further details on measures taken to meet each consideration in Table 5.1 are presented in Sections 5.2 - 5.7.

All software has been developed using Python, which is a clear and powerful programming language [130]. The following Python libraries are widely used within the software.

- NumPy [47] facilitates efficient scientific computations with multidimensional arrays.
- SciPy [134] is a scientific computing library containing a variety of algorithms, data structures and mathematical formulas.
- Matplotlib [53] enables the creation of high-quality static or animated plots.
- SALib [50] contains implementations of sensitivity analysis methods.
- statsmodels [114] is used to conduct statistical tests and to estimate a variety of statistical models.
- pandas [82] is utilized to perform data analysis.

PEP8 [133] guidelines should be adopted during the development of Python code to improve the readability, structure and consistency of code. Areas where PEP8 guidelines are applicable include the layout of code, the use of comments and naming conventions of variables, functions, modules and classes [133]. In some cases, a particular guideline may not be adopted if the code's readability is reduced or the code's functionality is affected. Flake8 [142] is a Python library that checks code for errors and compliance with various PEP8 guidelines. Throughout the software development stage, Flake8 has been invoked to identify errors and ensure compliance with various PEP8 guidelines where possible.

Software has been developed for multistart with early termination of descents (METOD) in Chapter 3 and the choice of direction in high-dimensional response surface methodology (RSM) within Chapter 4. All software is contained in publicly accessible repositories hosted on GitHub [41], and the locations of each repository are presented in Table 5.2. The motivation for developing software for Chapters 3 and 4 is twofold. Firstly, ensuring high-quality software will instil confidence that numerical comparisons in Chapters 3 and 4 are accurate and valid. Secondly, examples applying the software are included within each repository to allow other researchers to implement and use the software for a specified problem. Furthermore, the software can be modified to provide a foundation for further research.

5.2 Version control and accessibility

During the course of research, it is vital that different conjectures are explored and tested. For example, modified versions of a program may be applied, and

Repository name	URL
METOD algorithm [112] (Chapter 3)	https://github.com/Megscammell/METOD-Algorithm
Estimate of direction in RSM [113] (Chapter 4)	https://github.com/Megscammell/Estimate-of-direction-in-RSM

Table 5.2: Name and location of the repositories containing the software used for Chapters 3 and 4. Repositories contain source code, tests, examples and numerical experiments.

outputs may be stored to investigate the influence of the modifications on results. However, different versions of programs may be saved under other names within a directory, which may result in programs being deleted by mistake. Also, the differences between various versions of a program may be forgotten. Fortunately, a version control system can be used to combat these problems. A version control system tracks changes of files from a repository and ensures that previous versions of files are accessible. Also, a message explaining the reason for a change within a file can be specified to identify the key differences between each program version. Using a version control system is an important requirement for software development within [7, 55, 100, 111, 136].

The version control system for all software is Git [40] and the online hosting site is GitHub [41]. GitHub is a platform that enables developers to explore a broad range of repositories, share software publicly and collaborate with other researchers, all of which provide the opportunity to enhance software quality. In order to use Git and GitHub, a directory is initialized as a Git repository. Files within the directory can be altered, and once the developer is satisfied with the alternations, files are added, committed and pushed to the remote repository on GitHub. A valuable aspect of GitHub is that a screenshot of each file is available after each commit, and the deletions and additions made are highlighted in red and green. Therefore, changes made to files are transparent and can be easily observed by other researchers inspecting the repository. Further details explaining how to use Git and GitHub are provided in [9].

In addition to version control, GitHub also provides the option for repositories to be made publicly accessible. Various advantages of publicly accessible software are outlined in [55] and include, the ability to reproduce results since all software versions are made available and promotion of software. Both advantages support the reproducibility and reusable requirements in [7]. All code, data, plots and documentation for software are available in GitHub repositories within Table 5.2. In addition, the description of parameters and respective values used to execute the software are provided within comments of the code or documentation. Therefore, other researchers will have enough information to re-write the software and obtain the same results, which supports the replicable requirement in [7].

A licence is required to instruct other researchers on how to use the software. The importance of software licences is discussed in [85] and include ensuring owners have control over how software is used, mitigate the risk of any legal claims and encouraging other researchers to use the software. An MIT licence is employed, which allows the use, modification and distribution of code whilst ensuring the owner receives recognition [85]. Providing an open-source licence is recommended in [55] to promote best practices in research software. Since an open-source licence encourages other researchers to use and modify the software, opportunities for collaboration and further development of the software may arise.

Both repositories in Table 5.2 contain a Digital Object Identifier (DOI), produced using Zenodo [30]. The purpose of a DOI is to allow other researchers to cite the software easily. By [100], the benefits of citable software are that developers can receive recognition for their code, and results can be reproduced since previous versions of the software are available. In addition, [55] states that if the developer and other researchers have utilized the software to publish research, then acknowledgement from citations will increase visibility, and other researchers may be inclined to use the software. Releases within GitHub capture a particular instance of the repository. Each time a new release is made, a new DOI link is produced, and copies of each released version are available in Zenodo. Hence, any results produced using different software versions can be easily reproduced. Therefore, providing a DOI link supports the reproducible and reusable requirements in [7].

5.3 Reproducible results

A variety of measures are carried out to ensure that results can be reproduced. Some measures include environments to run code, initializing the pseudo-random number generator, and storing results used to generate summary plots. All three measures will be discussed, focusing on how each measure has been considered for software development.

The environment used to run any code should be stored and contain dependencies of the system, software and libraries. An environment is required to meet the re-runnable and reproducible requirements in [7]. Furthermore, ensuring dependencies of the software are documented is outlined as a rule for reproducible research in [111]. Code may not run if an environment is not stored since the system, software, or libraries may be updated. In addition, results may differ from those previously obtained. Therefore, environments are contained within the GitHub repositories in Table 5.2 and can be easily accessed, which allows other researchers the opportunity to re-run the code and obtain the same results.

A pseudo-random number generator can be used to produce pseudo-random numbers. A simulation may be used to investigate the performance of the software with a variety of different input parameters and starting points, which may be generated using pseudo-random numbers. Seeds for the pseudo-random number generator are used to generate the same pseudo-random numbers. Therefore, simulations should be repeated with different pseudo-random numbers to ensure that the initialization of a seed does not influence the results. This can be done by either initializing a seed for several simulations or changing the seed for each simulation. The pseudo-random number generator seed must be recorded to acquire the same results. Otherwise, results will differ each time the software is executed. Hence, the importance of initializing and recording seeds of the pseudo-random number generator is discussed in [7, 111] to ensure repeatability and reproducibility of results. All numerical experiments in Chapters 3 and 4 involve generating random starting points and function parameters to compare existing and proposed methods. Therefore, a seed for the pseudo-random number generator is specified for all programs used to generate numerical experiments to reproduce results.

Plots are used to describe the general trends of results. Storing data and programs to generate plots is considered an essential aspect of reproducibility in [7, 111]. The availability of data and programs to create plots provides confidence that the main trends of the data are captured accurately. The advantages of storing data and programs to generate plots are discussed in [111]. They include the ability to improve plots by making simple modifications to programs rather than re-running all software, and the opportunity for closer inspection of results by allowing all data to be accessible. All results of numerical experiments and analysis have been stored within each repository in Table 5.2, along with programs used to generate various plots to summarize the main trends of results.

5.4 Testing

During the development of software, it is common that some errors occur. However, understanding how errors can be addressed is essential for developing software. Therefore, it is vital that testing is undertaken to identify elements of the code where errors or unexpected outputs may occur. Testing code is considered a best practice for developing software [100, 136] and ensuring reproducibility of research [7].

The following layers of defence, outlined in [136], are effective actions to reduce the risk of mistakes within programs. The first layer of defence against mistakes is defensive programming. To ensure simple errors are caught early, assertions can be used. Assertions check whether a particular statement holds within a program. If the assertion is not true, then an error will be produced. For example, an assertion within code may check that the shape of an array is of the expected dimension. By [136], assertions simplify debugging and explain the functionality of a program by providing information on the expected behaviour. Assertions are used within all developed software to identify any mistakes early on.

The second layer of defence outlined in [136] is automated testing. A wide range of testing needs to be conducted to ensure the accuracy of software and outputs. Python contains libraries that can be used for automated testing, and the testing framework used for all source code is `pytest` [67]. Advantages of `pytest` over other testing frameworks can be found in [95] and include the simplicity of creating and reading tests. Testing can be split into two parts: example based and property based testing. Example based testing asserts that outputs match a given solution. For instance, if an example based test was applied for a function that calculates the sum of two numbers, we could assert that $4 + 5 = 9$. Alternatively, property based testing ensures that outputs have a particular form. For instance, if a property based test was applied for the previous example discussed, a function that calculates the sum of two numbers, the test will assert that a number is returned. The `hypothesis` library [79] can be used for property based testing and works by generating a wide range of input parameters used by the code to ensure outputs are of a particular form. In addition, the `hypothesis` library can also be useful in generating a set of input parameters that may cause errors in the software. This is beneficial since the software can be amended accordingly to rectify errors.

All tests pass for the source code in each repository within Table 5.2, which can be observed in Figures 5.1 and 5.2. The warnings which appear in Figure 5.1 are inbuilt warning messages within the METOD algorithm and are not a concern. The warning presented in Figure 5.2 is generated from a program used within the statsmodels library. The warning message is not a concern provided that the environment containing all dependencies of the system, software and libraries is used.

```
$ pytest
collected 268 items

tests/test_all_comparisons_both.py ..                                                           [  0%]
tests/test_apply_sd_until_stopping_criteria.py ......                                           [  2%]
tests/test_apply_sd_until_warm_up.py ...                                                        [  4%]
tests/test_calc_minimizer_qing.py ..                                                            [  4%]
tests/test_calc_minimizer_shekel.py ..                                                          [  5%]
tests/test_calc_minimizer_styb.py ..                                                            [  6%]
tests/test_check_alg_cond.py ...                                                                [  7%]
tests/test_check_alg_cond_all_possibilities.py ....                                             [  8%]
tests/test_check_classification_sd_metod.py .                                                   [  9%]
tests/test_check_des_points.py ....                                                             [ 10%]
tests/test_check_grad_starting_point.py ....                                                    [ 12%]
tests/test_check_if_new_minimizer.py ..                                                         [ 13%]
tests/test_check_matchings.py ...                                                               [ 14%]
tests/test_check_non_matchings.py ...                                                           [ 15%]
tests/test_check_outputs.py ..                                                                  [ 16%]
tests/test_check_quantities.py .                                                                [ 16%]
tests/test_check_unique_minimizers.py ...                                                       [ 17%]
tests/test_compute_trajectories.py ..                                                           [ 18%]
tests/test_create_sobol_sequence_points.py ..                                                   [ 19%]
tests/test_distances.py .......                                                                 [ 21%]
tests/test_evaluate_quantities_with_points.py ..                                                [ 22%]
tests/test_forward_backward_tracking.py ..............                                          [ 27%]
tests/test_function_parameters_several_quad.py ......                                           [ 29%]
tests/test_function_parameters_shekel.py ......                                                 [ 32%]
tests/test_function_parameters_sog.py .....                                                     [ 33%]
tests/test_griewank.py ...                                                                      [ 35%]
tests/test_hartmann6.py ......                                                                  [ 37%]
tests/test_individual_comparisons.py ......                                                     [ 39%]
tests/test_main_analysis_other.py ...                                                           [ 40%]
tests/test_main_analysis_quad.py ....                                                           [ 42%]
tests/test_main_analysis_shekel.py ...                                                          [ 43%]
tests/test_main_analysis_sog.py ...                                                             [ 44%]
tests/test_metod.py .............................                                               [ 55%]
tests/test_metod_class.py ..........................                                            [ 64%]
tests/test_metod_without_class.py ..........................                                    [ 74%]
tests/test_minimize_function.py .                                                               [ 75%]
tests/test_multistart.py ..                                                                     [ 75%]
tests/test_partner_point_each_sd.py ...                                                         [ 76%]
tests/test_qing_function.py ..                                                                  [ 77%]
tests/test_qing_gradient.py ..                                                                  [ 78%]
tests/test_quad_function.py ......                                                              [ 80%]
tests/test_quantities.py .                                                                      [ 80%]
tests/test_regions_greater_than_2.py ..                                                         [ 81%]
tests/test_rotation.py ..                                                                       [ 82%]
tests/test_sd_iteration.py ............                                                         [ 86%]
tests/test_several_quad_func.py ..                                                              [ 87%]
tests/test_several_quad_grad.py ..                                                              [ 88%]
tests/test_shekel_func.py .                                                                     [ 88%]
tests/test_shekel_grad.py .                                                                     [ 89%]
tests/test_single_quad_func.py ..                                                               [ 89%]
tests/test_single_quad_grad.py ..                                                               [ 90%]
tests/test_sog_function.py ......                                                               [ 92%]
tests/test_sog_gradient.py ........                                                             [ 95%]
tests/test_styblinski_tang_function.py ..                                                       [ 96%]
tests/test_styblinski_tang_gradient.py ..                                                       [ 97%]
tests/test_trid.py ...                                                                          [ 98%]
tests/test_version.py .                                                                         [ 98%]
tests/test_zakharov.py ...                                                                      [100%]

=========================================== warnings summary ===========================================
tests/test_check_grad_starting_point.py::test_1
tests/test_metod.py::test_22
tests/test_metod.py::test_23
tests/test_metod.py::test_28
  /Users/megscammell/OneDrive - Cardiff University/Megs Work/Python/METOD Algorithm Project/GitHub/METOD/METOD-
       Algorithm/src/metod_alg/metod_algorithm_functions/check_grad_starting_point.py:74: RuntimeWarning: Norm
       of gradient at starting point is too small. A new starting point will be used, RuntimeWarning)

-- Docs: https://docs.pytest.org/en/stable/warnings.html
============================= 268 passed, 4 warnings in 3352.25s (0:55:52) =============================
```

Figure 5.1: Applying `pytest` for the METOD algorithm source code.

```
$ pytest
collected 98 items

tests/test_alternative_search_direction.py ........                            [  9%]
tests/test_calc_first_phase_RSM.py ..............                              [ 24%]
tests/test_calc_its_until_sc_LS.py ...                                         [ 27%]
tests/test_calc_its_until_sc_MP.py ...                                         [ 30%]
tests/test_calc_its_until_sc_XY.py ...                                         [ 33%]
tests/test_create_design_matrix.py .......                                     [ 40%]
tests/test_forward_backward_tracking.py ......................                 [ 64%]
tests/test_num_exp_SNR.py .........................                            [ 90%]
tests/test_quad_func_params.py ..                                              [ 92%]
tests/test_quad_function.py ..                                                 [ 94%]
tests/test_sqr_quad_function.py ..                                             [ 96%]
tests/test_sqrt_quad_function.py ..                                            [ 98%]
tests/test_version.py .                                                        [100%]

============================== warnings summary ====================================
../../../../../../../../opt/anaconda3/envs/est_dir_env/lib/python3.8/site-packages/patsy/constraint.py:13
  /Users/megscammell/opt/anaconda3/envs/est_dir_env/lib/python3.8/site-packages/patsy/constraint.py:13:
       DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is
       deprecated since Python 3.3, and in 3.9 it will stop working
    from collections import Mapping

-- Docs: https://docs.pytest.org/en/stable/warnings.html
======================= 98 passed, 1 warning in 53.58s =============================
```

Figure 5.2: Applying `pytest` for the Estimate of Direction in RSM source code.

5.5 Coverage

Testing is an essential aspect of software development when implemented correctly. In some cases, tests may not check all code within a program, and as a result, errors and unexpected outputs may still occur when programs are executed. Therefore, the developer must be able to assess the quality of testing. This can be achieved by observing test coverage.

A test coverage tool works by monitoring which lines of code within a program have been executed by a test. The percentage of test coverage will reduce according to the number of lines of code that have not been executed. Test coverage is proposed as best practice in [100] since it can give general information on whether code outputs depict expected behaviour. Also, information from the test coverage can highlight where improvements are needed to ensure the accuracy and reliability of code. It is desirable to have a high percentage of coverage since this indicates that a test has executed each line within a program. The `pytest-cov` tool [103] is used to check the coverage percentage of source code. The coverage is 100% for source code in each repository within Table 5.2, and details of the coverage can be observed in Figures 5.3 and 5.4.

```
$ pytest --cov-report term --cov=src
collected 268 items
---------- coverage: platform darwin, python 3.6.8-final-0 -----------
Name                                                                          Stmts   Miss  Cover
-------------------------------------------------------------------------------------------------
src/metod_alg/__init__.py                                                         3      0   100%
src/metod_alg/check_metod_class/__init__.py                                      12      0   100%
src/metod_alg/check_metod_class/check_alg_cond_all_possibilities.py              15      0   100%
src/metod_alg/check_metod_class/check_classification_sd_metod.py                  7      0   100%
src/metod_alg/check_metod_class/check_des_points.py                              17      0   100%
src/metod_alg/check_metod_class/check_if_new_minimizer.py                         8      0   100%
src/metod_alg/check_metod_class/metod_with_classification.py                    135      0   100%
src/metod_alg/check_metod_class/metod_without_classification.py                 110      0   100%
src/metod_alg/check_metod_class/quad_obj_func.py                                 19      0   100%
src/metod_alg/check_metod_class/regions_greater_than_2.py                         8      0   100%
src/metod_alg/check_metod_class/sog_obj_func.py                                  13      0   100%
src/metod_alg/metod.py                                                          100      0   100%
src/metod_alg/metod_algorithm_functions/__init__.py                              16      0   100%
src/metod_alg/metod_algorithm_functions/apply_sd_until_stopping_criteria.py      32      0   100%
src/metod_alg/metod_algorithm_functions/apply_sd_until_warm_up.py                18      0   100%
src/metod_alg/metod_algorithm_functions/check_alg_cond.py                        16      0   100%
src/metod_alg/metod_algorithm_functions/check_grad_starting_point.py             14      0   100%
src/metod_alg/metod_algorithm_functions/check_unique_minimizers.py               12      0   100%
src/metod_alg/metod_algorithm_functions/create_sobol_sequence_points.py           9      0   100%
src/metod_alg/metod_algorithm_functions/distances.py                             10      0   100%
src/metod_alg/metod_algorithm_functions/forward_backward_tracking.py             70      0   100%
src/metod_alg/metod_algorithm_functions/minimize_function.py                      3      0   100%
src/metod_alg/metod_algorithm_functions/partner_point_each_sd.py                  3      0   100%
src/metod_alg/metod_algorithm_functions/sd_iteration.py                          38      0   100%
src/metod_alg/metod_analysis/__init__.py                                         16      0   100%
src/metod_alg/metod_analysis/all_comparisons_matches_both.py                     33      0   100%
src/metod_alg/metod_analysis/check_matchings.py                                   7      0   100%
src/metod_alg/metod_analysis/check_non_matchings.py                               7      0   100%
src/metod_alg/metod_analysis/check_quantities.py                                  5      0   100%
src/metod_alg/metod_analysis/compute_trajectories.py                             25      0   100%
src/metod_alg/metod_analysis/evaluate_quantities_with_points.py                  17      0   100%
src/metod_alg/metod_analysis/evaluate_quantities_with_points_quad.py             15      0   100%
src/metod_alg/metod_analysis/individual_comparisons.py                           27      0   100%
src/metod_alg/metod_analysis/main_analysis_other.py                              46      0   100%
src/metod_alg/metod_analysis/main_analysis_quad.py                               55      0   100%
src/metod_alg/metod_analysis/main_analysis_shekel.py                             44      0   100%
src/metod_alg/metod_analysis/main_analysis_sog.py                                44      0   100%
src/metod_alg/metod_analysis/quantities.py                                       11      0   100%
src/metod_alg/multistart.py                                                      20      0   100%
src/metod_alg/objective_functions/__init__.py                                    33      0   100%
src/metod_alg/objective_functions/calc_minimizer_hartmann6.py                     9      0   100%
src/metod_alg/objective_functions/calc_minimizer_qing.py                         13      0   100%
src/metod_alg/objective_functions/calc_minimizer_sev_quad.py                      9      0   100%
src/metod_alg/objective_functions/calc_minimizer_shekel.py                        8      0   100%
src/metod_alg/objective_functions/calc_minimizer_sog.py                           8      0   100%
src/metod_alg/objective_functions/calc_minimizer_styb.py                         17      0   100%
src/metod_alg/objective_functions/check_outputs.py                               16      0   100%
src/metod_alg/objective_functions/function_parameters_several_quad.py            23      0   100%
src/metod_alg/objective_functions/function_parameters_shekel.py                  22      0   100%
src/metod_alg/objective_functions/function_parameters_sog.py                     25      0   100%
src/metod_alg/objective_functions/griewank.py                                    17      0   100%
src/metod_alg/objective_functions/hartmann6.py                                   16      0   100%
src/metod_alg/objective_functions/qing.py                                         5      0   100%
src/metod_alg/objective_functions/rotation.py                                    27      0   100%
src/metod_alg/objective_functions/several_quad.py                                12      0   100%
src/metod_alg/objective_functions/shekel.py                                      12      0   100%
src/metod_alg/objective_functions/single_quad.py                                  4      0   100%
src/metod_alg/objective_functions/sog.py                                         13      0   100%
src/metod_alg/objective_functions/styblinski_tang.py                              6      0   100%
src/metod_alg/objective_functions/trid.py                                        15      0   100%
src/metod_alg/objective_functions/zakharov.py                                     9      0   100%
src/metod_alg/version.py                                                          1      0   100%
-------------------------------------------------------------------------------------------------
TOTAL                                                                          1380      0   100%
================= 268 passed, 4 warnings in 3308.13s (0:55:08) =================
```

Figure 5.3: Test coverage for the METOD algorithm.

```
$ pytest --cov-report term --cov=src
collected 98 items
---------- coverage: platform darwin, python 3.8.5-final-0 -----------
Name                                                                     Stmts   Miss  Cover
--------------------------------------------------------------------------------------------
src/est_dir/PI_LS.py                                                        82      0   100%
src/est_dir/PI_MPI.py                                                       52      0   100%
src/est_dir/PI_XY.py                                                        47      0   100%
src/est_dir/__init__.py                                                     38      0   100%
src/est_dir/alternative_search_direction.py                                 38      0   100%
src/est_dir/compute_random_design.py                                        18      0   100%
src/est_dir/compute_y.py                                                    10      0   100%
src/est_dir/divide_abs_max_value.py                                          4      0   100%
src/est_dir/forward_backward_tracking.py                                   123      0   100%
src/est_dir/numerical_exp_SNR.py                                           222      0   100%
src/est_dir/quad_f.py                                                        7      0   100%
src/est_dir/quad_func_params.py                                             13      0   100%
src/est_dir/sqrt_quad_f.py                                                   7      0   100%
src/est_dir/square_quad_f.py                                                 7      0   100%
src/est_dir/version.py                                                       1      0   100%
--------------------------------------------------------------------------------------------
TOTAL                                                                      669      0   100%

========================= 98 passed, 1 warning in 76.03s (0:01:16) ===========================
```

Figure 5.4: Test coverage for the Estimate of Direction in RSM.

5.6 Continuous integration

Continuous integration engines check that software can be built successfully from a list of requirements. The list of requirements may include dependencies of the libraries needed for the software to run. GitHub Actions [42] incorporate continuous integration for a repository and continuously checks the build and testing of the software after each committed change. The following objectives of continuous integration are outlined in [100]. Firstly, continuous integration ensures software is executable on different machines and configurations. Secondly, other services can be integrated with continuous integration to run performance metrics such as test coverage and status of documentation, which are reported to GitHub. Continuous integration is considered as best practice in [100]. Therefore, GitHub Actions has been set up for each repository in Table 5.2 to ensure that the build and test coverage remains successful after updates have been made to files within each GitHub repository. The testing coverage is performed using Codevcov [21], which reports the percentage of test coverage after each committed change within a repository and the status of documentation is checked using Read the Docs [106].

5.7 Documentation

Providing detailed documentation enables the developer and other researchers to understand, apply and modify the software. Including documentation for software is considered as best practice in [100, 136], and suggested to meet the reusable requirement in [7]. Consider the following benefits of documentation, which are discussed in [7].

- ✓ The developer of the software understands and remembers the purpose of creating and updating the software. For example, the functionality of a program can easily be forgotten if details are not documented, and it may take a considerable amount of time to regain understanding on the purpose of the program. Therefore, including documentation ensures that key information is captured and the developer can quickly refamiliarise themselves with aspects of the software.
- ✓ Other researchers can apply and modify the software which may aid further study in a research area. Furthermore, recommendations may be made to enhance the software's adaptability and efficiency.

The full documentation for each repository in Table 5.2 is hosted on Read the Docs [106]. The full documentation for the METOD algorithm can be found at https://metod-algorithm.readthedocs.io/ and full documentation for the Estimate of Direction in RSM can be found at https://estimate-of-direction-in-rsm.readthedocs.io/. To ensure software documentation is of a high standard, the following contains a selection of rules from [69] that have been considered during the creation and development of documentation.

- *Write comments as you code*
  It is essential to write comments explaining the purpose and functionality of code. This ensures that code can be readily understood and improves readability for the developer and other researchers. Source code for each repository in Table 5.2 contains detailed comments on code functionality, various input parameters and expected outputs. In addition, all tests contain comments explaining the purpose of each test.
- *Include examples*
  Including informative and coherent examples will encourage other researchers to use the software for a specified problem. Examples are included for each repository in Table 5.2 and have an intuitive layout and structure, allowing other researchers to understand and modify the application of the software.
- *Include a README file*
  An online hosting site such as GitHub includes a README file within a repository. A README file should be self-contained and provide brief details on using the software. It is recommended in [69] that a README file should

include information on the installation process, the location of complete documentation, the licence of the software, the process of testing the functionality of the software and a quickstart example that can be easily followed.

Badges can also be displayed on README files, summarising various key benchmarks to evaluate the software. Badges include the status of the build, which can be obtained by GitHub Actions [42], the percentage of coverage by integrating GitHub Actions with Codecov [21], the status of documentation within Read the Docs [106] and the DOI of the software.

- *Include a quickstart guide*
  The process for installing and running software must be clear and straightforward. If the process is complicated, other researchers may not use the software or make mistakes during the installation or execution stage. Hence, an informative quickstart guide explaining the installation and application of the software is provided for both repositories in Table 5.2. Furthermore, additional details on the quickstart guide are provided in the full documentation.

- *Version control documentation*
  As discussed in Section 5.2, the version control system used is Git [40] and the online hosting site is GitHub [41]. Consequently, changes made to the documentation are tracked, and details explaining the changes are provided.

- *Use automated tools*
  Documentation is written using Sphinx [122] which is a Python documentation generator that uses reStructuredText to create clear and informative HTML web pages. Each GitHub repository within Table 5.2 is coupled with Read the Docs, which allows any committed changes in the documentation to be automatically reflected in Read the Docs.

- *Write error messages that provide solutions or point to documentation*
  Various errors can occur during the development and execution of software. It can be beneficial to embed informative messages explaining the reason for errors. For example, a function may require a float variable to run successfully. However, if a boolean variable is given, it would be helpful to provide an error message explaining that a float variable is required for the function to run successfully. A variety of informative error and warning messages have been embedded within some source code programs, to explain errors and possible solutions to consider for the software to run successfully.

5.8 Summary

Various measures from [7, 55, 69, 100, 111, 136] have been followed to ensure that all software for Chapters 3 and 4 incorporate the re-runnable, repeatable, reproducible, reusable and replicable requirements from [7]. The purpose of abiding by the requirements is to create, develop and maintain high-quality software, which is essential for the following reasons. Firstly, to reduce the risk of errors occurring during the execution of the software, which may result in the termination of software or inaccurate results. Secondly, to produce accurate research outputs efficiently. Finally, it is extremely important that other researchers can apply and modify the software, as this can allow the opportunity for discussion on enhancements of the software and further developments in a research area. Throughout this section, it is evident that the implementation of all measures discussed ensure high-quality software is used to generate results for Chapters 3 and 4.

Chapter 6

Conclusion

This chapter summarises the research contributions within this thesis and outlines potential further research.

6.1 Overview of research contributions

A summary of the main research contributions of this thesis are outlined in Section 1.3. The first research contribution is addressed in Chapter 3, where multistart with early termination of descents (METOD) [144] is presented, which reduces the number of repeated local descents to the same local minimizer. The early termination of descents in METOD is achieved through repeatedly evaluating the fundamental inequality (3.17). A variety of enhancements to the original METOD algorithm in [144] have been implemented to improve the performance. In addition, numerical examples show the high efficiency and accuracy of the METOD algorithm compared to multistart with various test functions.

The second research contribution is discussed in Chapter 4, where the choice of search direction $s^{(k)}$ for the iterative update (4.5) within the first phase of BW is investigated. In the RSM-related literature, the least-square estimator of the gradient of the response function at a current point is routinely suggested as $s^{(k)}$. However, a new search direction $s^{(k)}$ of the form (4.24) is presented, which is inspired by the work in [141, Chpt. 8] and [38, 39]. The new search direction (4.24) can significantly improve the first phase of BW and RSM, in general, for high-dimensional problems, especially when errors of the response are large.

High-quality software used for analysis and results in Chapters 3 and 4 has been made available on GitHub, which addresses the third research contribution. Chapter 5 illustrates the best practices adopted during the development stage to ensure reliability and accuracy of software. Examples are also available to demonstrate the application of the software, allowing other researchers to employ the software for high-dimensional optimization problems.

General applications of multistart and RSM for machine learning have been outlined in Chapters 3 and 4, which supports the final research contribution. In addition, potential enhancements to stochastic gradient descent (SGD) are outlined in Section 6.2, which is a celebrated optimization method utilized for various optimization problems within machine learning.

6.2 Potential further research

The purpose of this section is to discuss potential further research areas for the contributions outlined in Chapters 3 and 4. That is, multistart with early termination of descents (METOD) and the choice of direction in high-dimensional response surface optimization.

6.2.1 Multistart with early termination of descents (METOD)

The METOD algorithm is presented in Chapter 3, where the early termination of descents is achieved through repeatedly evaluating the fundamental inequality (3.17). As a result, the efficiency of multistart can be significantly improved, especially when the dimension of the problem is very large. Recall that the fundamental inequality (3.17) uses partner points $\tilde{x}_i = x_i - \beta\nabla f(x_i)$ $(i = 1, 2)$, where $\nabla f(x_i)$ is the gradient of the objective function at point x_i. Hence, an investigation into different first-order search directions (see Section 2.3) may be conducted to observe whether the efficiency of the METOD algorithm can be further improved whilst maintaining high accuracy.

The computation of function evaluations and derivatives is assumed to be inexpensive in Chapter 3. However, computing function evaluations and derivatives can be expensive in many practical high-dimensional global optimization problems. In that case, [144, Sect. 2.3] illustrates that inequality (3.17) can be altered so that the full gradient $\nabla f(x_i)$ is replaced with derivatives computed for a few directions only. Suppose $I = \{i_1, ..., i_p\}$ is the set of indices representing the direction index to compute the derivative, where p is the total number of directions explored. As discussed in [144, Sect. 2.3], the same set of indices $I = \{i_1, ..., i_p\}$ must be selected to compute the derivatives of f at points x_i $(i = 1, 2)$ when applying inequality (3.17). This approach can further improve the efficiency of the METOD algorithm by reducing the computational cost since full gradients are not required. In addition, this approach may improve the efficiency of stochastic gradient descent (SGD) when applied with highly non-convex objective functions (i.e. deep neural networks).

6.2.2 The choice of direction in high-dimensional response surface optimization

A new search direction (4.24) is presented in Chapter 4 for the first phase of the BW algorithm, which is the most often used and cited RSM strategy. Numerical comparisons show that the new search direction (4.24) can significantly improve the first phase of BW for high-dimensional problems, especially when errors of the response are large. Recall that design matrices for the new search direction are constructed according to the strategy outlined in Section 4.4.3. Hence, investigations on different design matrices may be conducted to determine whether the efficiency and accuracy of the new search directions (4.24) can be further improved.

The total number of function evaluations permitted during the first phase of BW with search directions (4.24) is required by the stopping condition (4.27) and must be set before applying the first phase of BW. However, it can be difficult to determine the total number of function evaluations required to find the subregion of the optimum in practical situations. Hence, an alternative stopping condition that considers the progress made in finding a subregion of the optimum would be more beneficial. For example, the first phase of BW may be stopped if the movement toward a subregion is minimal or the difference between response function values is marginal for a predefined number of iterations.

As discussed in [140], SGD may be improved by utilizing search directions of the form (4.24), where M represents the design matrix at chosen observations, which are updated at each iteration. Since search directions (4.24) can significantly improve the first Phase of BW and RSM, in general, it will be beneficial to investigate the use of search directions (4.24) within SGD.

Bibliography

[1] C. J. Alves, P. M. Pardalos, and L. N. Vicente, editors. *Optimization in Medicine*. Springer, 2008.

[2] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury. Simulation optimization: A review of algorithms and applications. *Annals of Operations Research*, 240(1):351–380, 2016.

[3] A. Y. Aydar. Utilization of Response Surface Methodology in Optimization of Extraction of Plant Materials. *Statistical Approaches with Emphasis on Design of Experiments Applied to Chemical Processes*, pages 157–169, 2018.

[4] R. R. Barton. Simulation optimization using metamodels. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 230–238. IEEE, 2009.

[5] Y. Bengio, A. Lodi, and A. Prouvost. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[6] K. P. Bennett and E. Parrado-Hernández. The Interplay of Optimization and Machine Learning Research. *The Journal of Machine Learning Research*, 7 (46):1265–1281, 2006.

[7] F. C. Benureau and N. P. Rougier. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *Frontiers in Neuroinformatics*, 11, 2018.

[8] M. A. Bezerra, R. E. Santelli, E. P. Oliveira, L. S. Villar, and L. A. Escaleira. Response surface methodology (RSM) as a tool for optimization in analytical chemistry. *Talanta*, 76(5):965–977, 2008.

[9] J. D. Blischak, E. R. Davenport, and G. Wilson. A Quick Introduction to Version Control with Git and GitHub. *PLoS Computational Biology*, 12(1): e1004668, 2016.

[10] A. Blum, J. Hopcroft, and R. Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

[11] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, 60(2):223–311, 2018.

[12] G. Box, J. Hunter, and W. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley Series in Probability and Statistics. Wiley, 2005. ISBN 9780471718130. URL `https://books.google.co.uk/books?id=oYUpAQAAMAAJ`.

[13] G. E. P. Box and K. B. Wilson. On the Experimental Attainment of Optimum Conditions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(1):1–45, 1951.

[14] R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973.

[15] C. G. Broyden. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[16] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone. Machine Learning for Global Optimization. *Computational Optimization and Applications*, 51(1):279–303, 2010.

[17] K. H. 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.

[18] K. H. Chang, M. K. Li, and H. Wan. Combining STRONG with screening designs for large-scale simulation optimization. *IIE Transactions*, 46(4):357–373, 2014.

[19] M. Chau, M. C. Fu, H. Qu, and I. O. Ryzhov. Simulation optimization: a tutorial overview and recent developments in gradient-based methods. In *Proceedings of the Winter Simulation Conference 2014*, pages 21–35. IEEE, 2014.

[20] E. K. P. Chong and S. H. Żak. *An Introduction to Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, Hoboken, New Jersey, Fourth edition, 2013. ISBN 1-118-52369-5.

[21] Codecov. URL `https://about.codecov.io/`.

[22] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[23] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 13(3):262–280, 1987.

[24] N. A. C. Cressie. *Statistics for Spatial Data*. Wiley Series in Probability and Statistics. Wiley-Interscience Publication, New York, revised edition, 1993.

[25] F. E. Curtis and K. Scheinberg. Optimization Methods for Supervised Machine Learning: From Linear Models to Deep Learning. In *Leading Developments from INFORMS Communities*, pages 89–114. INFORMS, 2017.

[26] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv preprint arXiv:1406.2572*, 2014.

[27] A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives. In *Advances in Neural Information Processing Systems*, volume 27, pages 1646–1654. Curran Associates, Inc, 2014.

[28] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[29] O. A. Elwakeil and J. S. Arora. Two algorithms for global optimization of general NLP problems. *International Journal for Numerical Methods in Engineering*, 39(19):3305–3325, 1996.

[30] European Organization For Nuclear Research and OpenAIRE. Zenodo, 2013. URL `https://www.zenodo.org/`.

[31] S. K. S. Fan and K. N. Huang. A new search procedure of steepest ascent in response surface exploration. *Journal of Statistical Computation and Simulation*, 81(6):661–678, 2011.

[32] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[33] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[34] M. C. Fu. Optimization via Simulation: A Review. *Annals of Operations Research*, 53(1):199–247, 1994.

[35] M. C. Fu, editor. *Handbook of Simulation Optimization*. Springer, 2014.

[36] M. C. Fu, F. W. Glover, and J. April. Simulation optimization: a review, new developments, and applications. In *Proceedings of the 2005 Winter Simulation Conference*, pages 83–95. IEEE, 2005.

[37] C. Gambella, B. Ghaddar, and J. Naoum-Sawaya. Optimization Problems for Machine Learning: A Survey. *European Journal of Operational Research*, 290 (3):807–828, 2021.

[38] J. Gillard and A. Zhigljavsky. Optimal directional statistic for general regression. *Statistics & Probability Letters*, 143:74–80, 2018.

[39] J. Gillard and A. Zhigljavsky. Optimal estimation of direction in regression models with large number of parameters. *Applied Mathematics and Computation*, 318:281–289, 2018.

[40] Git. URL `https://git-scm.com/`.

[41] GitHub. URL `https://github.com`.

[42] GitHub Actions. URL `https://github.com/features/actions`.

[43] D. Goldfarb. A Family of Variable-Metric Methods Derived by Variational Means. *Mathematics of Computation*, 24(109):23–26, 1970.

[44] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT press, 2016.

[45] I. E. Grossmann, editor. *Global Optimization in Engineering Design*, volume 9 of *Nonconvex Optimization and Its Applications*. Springer US, New York, NY, 1st edition, 1996.

[46] W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2(1):35–58, 2006.

[47] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[48] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, New York, NY, second edition, 2009.

[49] R. Haycroft. Studying the Rate of Convergence of the Steepest Descent Optimisation Algorithm with Relaxation. In *Computer Aided Methods In Optimal Design And Operations*, pages 49–58. World Scientific, 2006.

[50] J. Herman and W. Usher. SALib: An open-source Python library for Sensitivity Analysis. *The Journal of Open Source Software*, 2(9), 2017. doi: 10.21105/joss.00097. URL `https://doi.org/10.21105/joss.00097`.

[51] W. J. Hill and W. G. Hunter. A Review of Response Surface Methodology: A Literature Survey. *Technometrics*, 8(4):571–590, 1966.

[52] G. Hinton. Lecture 6e - rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, pages 26–31, 2012. URL `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[53] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: https://doi.org/10.5281/zenodo.4475376.

[54] M. Jamil and X. S. Yang. A Literature Survey of Benchmark Functions For Global Optimisation Problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[55] R. C. Jiménez, M. Kuzak, M. Alhamdoosh, M. Barker, B. Batut, M. Borg, S. Capella-Gutierrez, N. C. Hong, M. Cook, M. Corpas, et al. Four simple recommendations to encourage best practices in research software. *F1000 Research*, 6:876, 2017.

[56] R. Johnson and T. Zhang. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. *Advances in Neural Information Processing Systems*, 26:315–323, 2013.

[57] S. Joshi, H. D. Sherali, and J. D. Tew. An enhanced response surface methodology (RSM) algorithm using gradient deflection and second-order search strategies. *Computers & Operations Research*, 25(7-8):531–541, 1998.

[58] A. R. Kan and G. T. Timmer. Stochastic Global Optimization Methods Part I: Clustering Methods. *Mathematical Programming*, 39(1):27–56, 1987.

[59] A. I. Khuri and S. Mukhopadhyay. Response surface methodology. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):128–149, 2010.

[60] J. Kiefer. Sequential Minimax Search for a Maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953.

[61] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[62] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[63] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[64] J. P. Kleijnen, D. Den Hertog, and E. Angün. Response surface methodology's steepest ascent and step size revisited. *European Journal of Operational Research*, 159(1):121–131, 2004.

[65] J. P. C. Kleijnen. Kriging Metamodeling in Simulation: A Review. *European Journal of Operational Research*, 192(3):707–716, 2009.

[66] J. P. C. Kleijnen. Response Surface Methodology. In *Handbook of Simulation Optimization*, pages 81–104. Springer, 2015.

[67] H. Krekel et al. Pytest. URL `https://docs.pytest.org/en/latest/`.

[68] T. Krityakierne and C. A. Shoemaker. SOMS: SurrOgate MultiStart algorithm for use with nonlinear programming for global optimization. *International Transactions in Operational Research*, 24(5):1139–1172, 2017.

[69] B. D. Lee. Ten simple rules for documenting scientific software. *PLoS Computational Biology*, 14(12), 2018.

[70] Y. Li and M. Fu. Sequential first-order response surface methodology augmented with direct gradients. In *2018 Winter Simulation Conference (WSC)*, pages 2143–2154. IEEE, 2018.

[71] M. Locatelli. Relaxing the Assumptions of the Multilevel Single Linkage Algorithm. *Journal of Global Optimization*, 13(1):25–42, 1998.

[72] M. Locatelli and F. Schoen. Random Linkage: a family of acceptance/rejection algorithms for global optimisation. *Mathematical Programming*, 85(2):379–396, 1999.

[73] M. Locatelli and F. Schoen. Global optimization based on local searches. *Annals of Operations Research*, 240(1):251–270, 2015.

[74] W. Long-Fei and S. Le-Yuan. Simulation Optimization: A Review on Theory and Applications. *Acta Automatica Sinica*, 39(11):1957–1968, 2013.

[75] D. López-Soto, F. Angel-Bello, S. Yacout, and A. Alvarez. A multi-start algorithm to design a multi-class classifier for a multi-criteria ABC inventory classification problem. *Expert Systems with Applications*, 81:12–21, 2017.

[76] M. Lozano, F. J. Rodriguez, D. Peralta, and C. García-Martínez. Randomized greedy multi-start algorithm for the minimum common integer partition problem. *Engineering Applications of Artificial Intelligence*, 50:226–235, 2016.

[77] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer, 2015.

[78] G. A. Lujan-Moreno, P. R. Howard, O. G. Rojas, and D. C. Montgomery. Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study. *Expert Systems with Applications*, 109:195–205, 2018.

[79] D. MacIver. Hypothesis. URL `https://hypothesis.readthedocs.org/`.

[80] M. Makela. Experimental design and response surface methodology in energy applications: A tutorial review. *Energy Conversion and Management*, 151: 630–640, 2017.

[81] R. Martí, M. G. C. Resende, and C. C. Ribeiro. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 226 (1):1–8, 2013.

[82] W. McKinney. Data Structures for Statistical Computing in Python. In S. van der Walt and J. Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[83] G. Miró-Quesada and E. Del Castillo. An enhanced recursive stopping rule for steepest ascent searches in response surface methodology. *Communications in Statistics-Simulation and Computation*, 33(1):201–228, 2004.

[84] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2017.

[85] A. Morin, J. Urban, and P. Sliz. A Quick Guide to Software Licensing for the Scientist-Programmer. *PLoS Computational Biology*, 8(7):e1002598–e1002598, 2012.

[86] R. H. Myers and A. I. Khuri. A new procedure for steepest ascent. *Communications in Statistics-Theory and Methods*, 8(14):1359–1376, 1979.

[87] R. H. Myers, A. I. Khuri, and W. H. Carter. Response Surface Methodology: 1966-1988. *Technometrics*, 31(2):137–157, 1989.

[88] R. H. Myers, D. C. Montgomery, G. G. Vining, C. M. Borror, and S. M. Kowalski. Response Surface Methodology: A Retrospective and Literature Survey. *Journal of Quality Technology*, 36(1):53–77, 2004.

[89] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments.* Wiley, Fourth edition, 2016.

[90] H. G. Neddermeijer, G. J. Van Oortmarssen, N. Piersma, and R. Dekker. A framework for response surface methodology for simulation optimization. In *Proceedings of the 32nd Conference on Winter Simulation*, pages 129–136. Society for Computer Simulation International, 2000.

[91] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[92] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87 of *Applied Optimization.* Springer, 1st edition, 2004.

[93] R. P. Nicolai, R. Dekker, N. Piersma, and G. J. van Oortmarssen. Automated Response Surface Methodology for Stochastic Optimization Models with Unknown Variance. In *Proceedings - Winter Simulation Conference*, volume 1, pages 491–499, 2004.

[94] J. Nocedal and S. Wright. *Numerical Optimization.* Springer Science & Business Media, 2nd edition, 2006.

[95] B. Okken. *Python Testing with pytest: Simple, Rapid, Effective, and Scalable.* Pragmatic Programmers. The Pragmatic Bookself, 2017.

[96] L. Palagi. Global optimization issues in deep network regression: An overview. *Journal of Global Optimization*, 73(2):239–277, 2018.

[97] D. Peri. Self-Learning Metamodels for Optimization. *Ship Technology Research*, 56(3):95–109, 2009.

[98] D. Peri and F. Tinti. A multistart gradient-based algorithm with surrogate model for global optimization. *Communications in Applied and Industrial Mathematics*, 3(1):1–22, 2012.

[99] J. Pinter, editor. *Global Optimization Scientific and Engineering Case Studies.* Nonconvex optimization and its applications. Springer, New York, 2006.

[100] T. Poisot. Best publishing practices to improve user confidence in scientific software. *Ideas in Ecology and Evolution*, 8(1), 2015.

[101] B. Polyak. *Introduction to Optimization.* Optimiaztion Software, 1987.

[102] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5): 1–17, 1964.

[103] pytest-cov. URL `https://pytest-cov.readthedocs.io/`.

[104] G. P. Rangaiah, editor. *Stochastic Global Optimization Techniques and Applications in Chemical Engineering.* Advances in Process Systems Engineering. World Scientific Pub, Singapore, 2010.

[105] M. Raydan and B. F. Svaiter. Relaxed steepest descent and Cauchy-Barzilai-Borwein method. *Computational Optimization and Applications*, 21(2):155–167, 2002.

[106] Read the Docs. URL `https://readthedocs.org/`.

[107] A. G. Rinnooy Kan and G. T. Timmer. Stochastic Global Optimization Methods Part II: Multi level methods. *Mathematical Programming*, 39(1):57–78, 1987.

[108] H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[109] N. L. Roux, M. Schmidt, and F. Bach. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. *arXiv preprint arXiv:1202.6258*, 2012.

[110] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[111] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig. Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*, 9(10): e1003285, 2013.

[112] M. Scammell. Megscammell/METOD-Algorithm, Apr. 2022. URL `https://doi.org/10.5281/zenodo.6474640`.

[113] M. Scammell. Megscammell/Estimate-of-direction-in-RSM, Apr. 2022. URL `https://doi.org/10.5281/zenodo.6474635`.

[114] S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with Python. In *9th Python in Science Conference*, 2010.

[115] A. E. Sepulveda and L. Epstein. The repulsion algorithm, a new multistart method for global optimization. *Structural Optimization*, 11(3-4):145–152, 1996.

[116] D. F. Shanno. Conditioning of Quasi-Newton Methods for Function Minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[117] R. Shaydulin, I. Safro, and J. Larson. Multistart Methods for Quantum Approximate Optimization. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2019.

[118] I. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967. URL `https://www.sciencedirect.com/science/article/pii/0041555367901449`.

[119] H. Song, I. Triguero, and E. Özcan. A review on the self and dual interactions between machine learning and optimisation. *Progress in Artificial Intelligence*, 8(2):143–165, 2019.

[120] J. C. Spall. *Introduction to Stochastic Search and Optimization Estimation, Simulation, and Control.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, 2003.

[121] J. C. Spall. Stochastic Optimization. In *Handbook of Computational Statistics*, pages 173–201. Springer, 2012.

[122] Sphinx. Sphinx Python Documentation Generator. URL `https://www.sphinx-doc.org/`.

[123] M. Staib, S. Reddi, S. Kale, S. Kumar, and S. Sra. Escaping Saddle Points with Adaptive Gradient Methods. In *International Conference on Machine Learning*, pages 5956–5965. PMLR, 2019.

[124] S. Sun, Z. Cao, H. Zhu, and J. Zhao. A Survey of Optimization Methods from a Machine Learning Perspective. *IEEE Transactions on Cybernetics*, 50(8): 3668–3681, 2020.

[125] D. Surmann, U. Ligges, and C. Weihs. Infill Criterion for Multimodal Model-Based Optimisation. *ArXiv*, 2018.

[126] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147. JMLR, 2013.

[127] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Adaptive Computation and Machine Learning. MIT press, 2018.

[128] D. Taubman and A. Zakhor. A multi-start algorithm for signal adaptive sub-band systems. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 213–216. IEEE, 1992.

[129] E. Tekin and I. Sabuncuoglu. Simulation optimization: A comprehensive review on theory and applications. *IIE Transactions*, 36(11):1067–1081, 2004.

[130] The Python Software Foundation. Python. URL `https://wiki.python.org/`.

[131] A. Törn and A. Žilinskas. Global Optimization. *Lecture Notes in Computer Science*, 350:1–252, 1989.

[132] W. Tu and R. Mayne. Studies of multi-start clustering for global optimization. *International Journal for Numerical Methods in Engineering*, 53(9):2239–2252, 2002.

[133] G. Van Rossum, B. Warsaw, and N. Coghlan. Pep 8: Style Guide for Python Code. *Python org*, 1565, 2001. URL `https://www.python.org/dev/peps/pep-0008/`.

[134] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

[135] D. Weichert, P. Link, A. Stoll, S. Rüping, S. Ihlenfeldt, and S. Wrobel. A review of machine learning for the optimization of production processes. *The International Journal of Advanced Manufacturing Technology*, 104(5-8):1889–1902, 2019.

[136] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, et al. Best Practices for Scientific Computing. *PLoS Biology*, 12(1):e1001745, 2014.

[137] M. Yolmeh and S. M. Jafari. Applications of Response Surface Methodology in the Food Industry Processes. *Food and Bioprocess Technology*, 10(3):413–433, 2017.

[138] M. D. Zeiler. AdaDelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[139] A. Zhigljavsky and A. Žilinskas. *Stochastic Global Optimization.* Springer Optimization and its Applications. Springer, 1st edition, 2008.

[140] A. Zhigljavsky and A. Žilinskas. *Bayesian and High-Dimensional Global Optimization.* Springer Nature, 2021.

[141] A. A. Zhigljavsky. *Theory of Global Random Search.* Springer, 1991.

[142] T. Ziade and I. Cordasco. Flake8. URL `https://pypi.org/project/flake8/`.

[143] A. Žilinskas and A. Zhigljavsky. Stochastic Global Optimization: A Review on the Occasion of 25 Years of Informatica. *Informatica*, 27(2):229–256, 2016.

[144] A. Žilinskas, J. Gillard, M. Scammell, and A. Zhigljavsky. Multistart with early termination of descents. *Journal of Global Optimization*, 79(2):447–462, 2019.