Imperial College London
Department of Computing

Global Optimisation Using Gentlest Ascent Dynamics and Saddle Point Stability of Functions on Differentiable Manifolds

Favour Mandanji Nyikosa

September 6, 2013

Supervised by Dr. Panos Parpas
Second Marker: Dr. Daniel Kuhn

Submitted in part fulfilment of the requirements for the degree of
Master of Science in Advanced Computing of Imperial College London

Declaration

I herewith certify that all material in this dissertation which is not my own work has been properly acknowledged.

Favour Mandanji Nyikosa

Abstract

The work in this treatise has dual aims. The first is the numerical distillation of the Gentlest Ascent Dynamics (GAD) in order to develop a numerically stable and globally convergent index-1 saddle-point-finding algorithm. This is for the purpose of solving an important but difficult problem in computational chemistry and physics of finding the transition point along the minimum energy paths on potential energy surfaces. These transition points are index-1 saddle points. The second aim is to construct a global optimisation algorithm that takes advantage of the Morse-theoretic properties of saddle points.

The solution trajectories of the GAD autonomous differential equations escape the basins of attraction of minima and converge at the low-lying index-1 saddle points. However, the GAD equations suffer from an unfavourable property in numerical implementation of easily being unstable. The work in this thesis presents modified GAD algorithms that are computationally cheap, numerically stable and convergent. In addition, a solution algorithm that is able to find all the saddle points on a function is introduced.

Furthermore, a novel deterministic algorithm is proposed for the unconstrained global optimisation of non-convex and nonlinear objective functions. The method makes use of the solution trajectories of the aforementioned GAD equations that converge at saddle points. The saddle points are then used as transition points to aid exploring a multimodal function with the goal of always finding better minimisers at every iteration. Numerical experiments are conducted on prescribed standard test problems from literature are reported together with an application to the global optimisation of the potential energy surfaces of Lennard-Jones clusters.

To the memory of my parents,
who tolerated and tried to answer all of my insistent and nonending questions.
And to the memory of my grandmother Florence,
who believed in me.
I know you'd all be proud.

Acknowledgements

I am greatly indebted to the following for their contribution to this work:

- Panos Parpas, who supervised this work, for his helpful criticism and indispensable advice, guidance on how to approach this research problem, answering my endless awkward questions, always having time for me despite his busy schedule, and always having the appropriate text book or paper lying around — makings of a great supervisor;
- Tim Kimber, my personal tutor, for his help and advice on a diverse array of issues throughout my time at Imperial College;
- Clint Chin Ho, whose invaluable advice, intriguing questions and point of view helped to motivate me to go the extra mile and push me to produce a better piece of research;
- Sahin Mir, Haodan Li and Jinjie Mao, who helped me learn the MATLAB tricks that made the process of writing some of the numerical algorithms easier and less painful;
- Karol Pysniak, who gave me the right material that kickstarted this research and gave me inestimable advice that made me feel less lost in the global optimisation jungle;
- Chisenga Muyoya, who edited my early drafts, for proofreading a bulk of this thesis despite its deep mathematical intricacies, sparing time to do so in spite of her busy schedule and immense workload, and keeping my legendary ambiguously lengthy sentences in check. Mistakes, if any, are all mine;
- And the Rhodes Trust, who under extenuating circumstances granted my request to serve the first year of my tenure as a Rhodes Scholar at Imperial College.

Contents

List of Figures

List of Algorithms

List of Tables

Glossary

$\mathbb{E}^n$ n-dimensional space of real numbers satisfying the properties of a metric space.

$\mathbb{R}^n$ n-dimensional space of real numbers.

GAD Gentlest Ascent Dynamics.

GMRES Generalised Minimal Residual.

$\mapsto$ "maps to".

$\to$ "maps to".

$\in$ "member of".

ODE Ordinary Differential Equation.

$\subset$ "subset of".

1 Introduction

For since the fabric of the universe is most perfect and the work of a most wise Creator, nothing at all takes place in the universe in which some rule of the maximum or minimum does not appear.

– Leonhard Euler, 1744 [1]

We are faced with decision problems on a daily basis. As an example, one may ask: what time should I wake up to make it for my meeting on time? This scenario may seem banal at first but a less cursory examination will make one realise there are certain interesting intricacies that are associated with this question. It is easy to realise that the most basic information the person needs in order to make this decision is what time the meeting is starting and how long it takes to get to there. Using this knowledge the person can easily estimate the best time to wake up.

However, this simple problem can easily become complex after some thought. It is obvious at this point that whatever decision that will be made will depend on how much the person knows and what resources the person has. This decision will attempt to make the best use of this knowledge and resources. Now let's assume that there are many possible routes that this person could use to get to work. This would in turn mean that the best decision will be determined by the person's knowledge of what the best and quickest route to work is. So, if this person is convinced that a certain route is the quickest then this knowledge will factor into the decision of what the best time to wake up is. We can now ask very beguiling questions: what if this person's estimation of the quickest way to get to work is inaccurate? What if there is a quicker way he or she did not know about? In that case, the decision may be the best one with respect to the person's local knowledge but not the best one globally.

Generally, to optimise involves making the best decision in order to make the best use of resources [2]. As illustrated from the example, optimisation problems are embroidered in all the decisions people make because they conceptualise the notion of making the best use of resources

and knowledge one has available. These decisions may be locally or globally optimal depending on the knowledge one has. But everyone, if given a choice, would want to be omniscient in order that they make the globally optimal decision every time — and that is the motivation for global optimisation.

1.1. A Brief Overview of Optimisation Theory and Non-linear Multi-modal Functions

Optimisation can be compactly but clearly defined as the selection of a best element from some set of available alternatives, usually with some criteria [2, 3]. The measure of goodness of an alternative is specified by an *objective function* [2]. Therefore, optimisation is a field of applied mathematics and numerical analysis that deals with finding the extremal value of an objective function in its domain of definition, subject to various constraints [4]. This is mathematically written out as:

$$\begin{aligned} &\underset{x}{\textbf{maximise/minimise}} && F(X) \\ &\text{subject to} && f_i(x) \leq b_i,\ i = 1, \ldots, m. \end{aligned} \tag{1.1}$$

where $F(X)$ is called the objective function and $f_i(x) \leq b_i$ is the set of constraints.

A commonly used categorisation of optimisation problems, also referred to as mathematical programming problems, is in relation to the *convexity* of the objective function [4]. Using this discrimination then problems can either be convex or non-convex.

A problem is convex if it reduces to a minimisation of a convex function where the admissible optimal points are said to be within what is called a convex set [4]. From the illustration in Figures 1.1 and 1.2, a convex function is one with a single trough and generally "$\cup$" shaped, while a concave function is its mirror image and has a single "bump" and "$\cap$" shaped. Because of this symmetry, it suffices to talk only about the minimisation of a convex objective function and we will adhere to this convention for the rest of this thesis. Since a convex function has a single lowest point, finding this optimum point would constitute as finding both the local and global minimum point on the curve. This is a fundamental result of the analysis of convex functions [4]. This result is practically important because optimisation algorithms perform termination tests at each iteration and it is thus imperative that they are computationally efficient. As a result, all termination conditions of these algorithms are local conditions as they test whether the current solution is optimal with respect to a predefined neighbourhood. [4]. Therefore, if the defined problem is convex, then that is enough to guarantee that a solution is globally optimal [4].

On the other end of the spectrum are non-convex problems. These problems have many different local optima. An intuitive illustration is shown in figure 1.3 below where the objective function is convex in some sections and concave in others yielding the multiple optima.

The reason this problem is non-convex is because the admissible points are within a non-convex set. Finding the globally optimal point among the numerous locally optimal points in such problems is an extremely difficult task, and the field dealing with finding the extremal points of non-convex

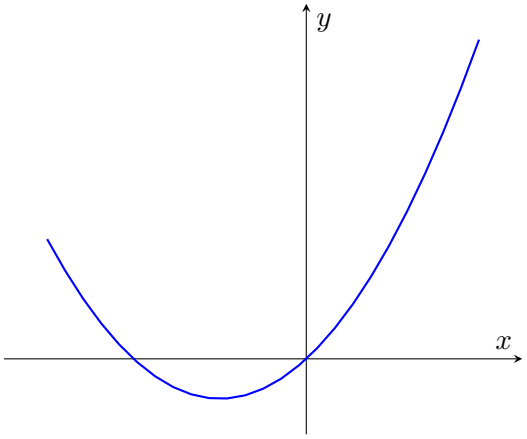


Figure 1.1.: *Convex curve*

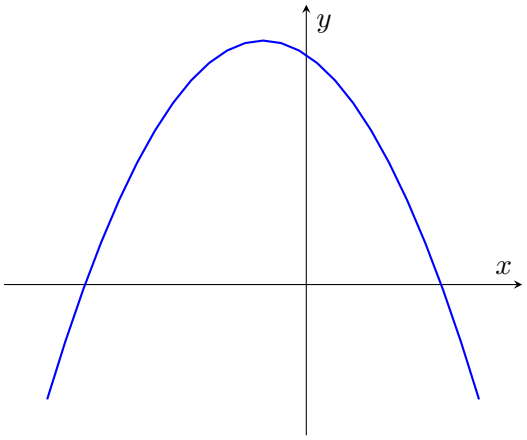


Figure 1.2.: *Concave curve*

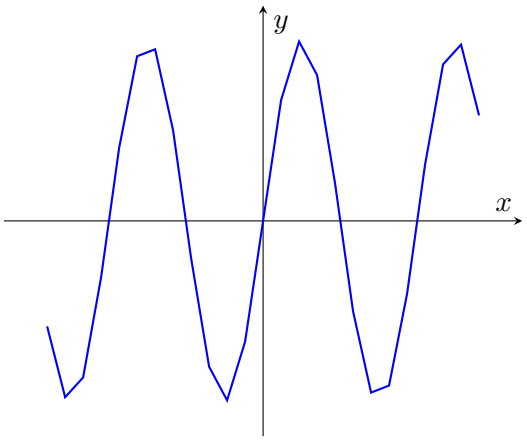


Figure 1.3.: *Non-Convex curve*

functions is called Global Optimisation [4]. Continuing with the intuition introduced in the example from the proceeding section, this is analogous to one making decisions that consider all the possible information rather than taking into consideration only the local information one has. From the wide range of current uses of optimisation available from the advances in the theory made so far, it is evident that attaining the goal of global optimisation will greatly improve the quality of our lives [5].

1.2. Premise of this Research

Non-linear and non-convex functions appear naturally in numerous models representing the real world. A vast majority of the objective functions that represent decision problems are non-convex. Because these functions have numerous maxima and minima, they have numerous saddle points too. A saddle point is a point in the domain of a function that is a stationary point but is neither a local maximum or minimum. The name is derived from an observation in a prototypical surface in two dimensions as illustrated in figure 1.4. It is formed as the result of a surface that curves up in one direction and curves down in a different direction; it appears as a saddle or a mountain pass.

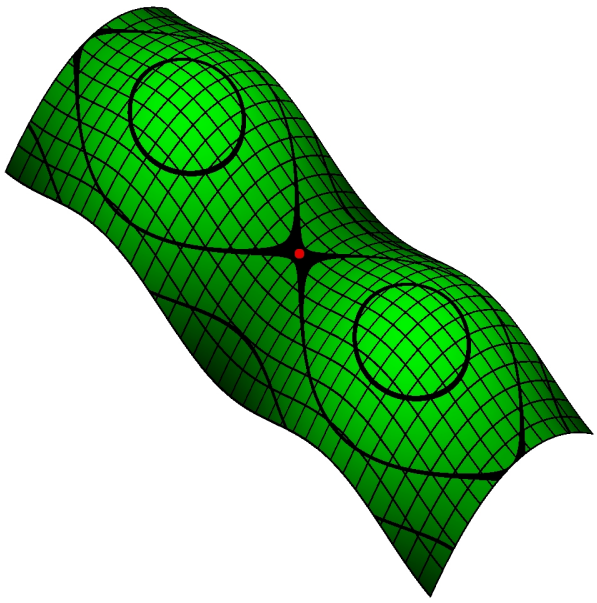


Figure 1.4.: *Illustration of a two dimensional saddle point (Source: Wikimedia Commons)*

A very good example of non-convex functions occurring in the real-world are the potential energy functions that represent the energy changes in atoms undergoing a chemical change. A common and important problem in theoretical chemistry and in condensed matter physics is the calculation of the rate of transitions in phenomena such as chemical reactions. The configuration of atoms changes in some way during this transition. The interaction between the atoms can be obtained from a determined potential energy function. Research has shown that all these transitions happen in such a way that they seek out a minimum energy path that converges at the globally stable minimum of this function as a final state. This minimum energy path is shown to traverse through a transition state that is a saddle point before reaching the final stable state with minimal energy [6].

As a result of the importance of this transition state, there has been a lot of research into saddle-point-finding methods. However, there has not been a single successful method that is able to differentiate and target the different types of saddle points depending on the number of minima they are connected to. Finding a saddle point is a much more difficult task than finding a minimum point. The class of algorithms that have shown much promise are the basin-escaping methods that have inspired a large number of new methods including the Gentlest Ascent Dynamics (GAD) [6, 7, 8, 9].

The previously described natural processes performs a stability-based global optimisation of a non-convex potential energy function. It does this with the least energy by following a path that goes through a saddle point. This illustrates the natural relationship between saddle points and global optimisation.

Generally, problems in global optimisation are formulated in terms of finding a point, say x, in a solution space set X, called the *feasible region*, where an objective function $f : X \to T$ attains a minimum (or maximum) value. T, in this case, is any ordered set and is usually a subset of the set of real numbers $\mathbb{R}$. X, on the other hand, is usually a subset of the vector space $\mathbb{R}^n$ and may be bounded by constraints, a case that will not be considered in this thesis. The extremal point x can then be written out as $(x_1, ..., x_n)$, and the x_i's are called *decision variables*.

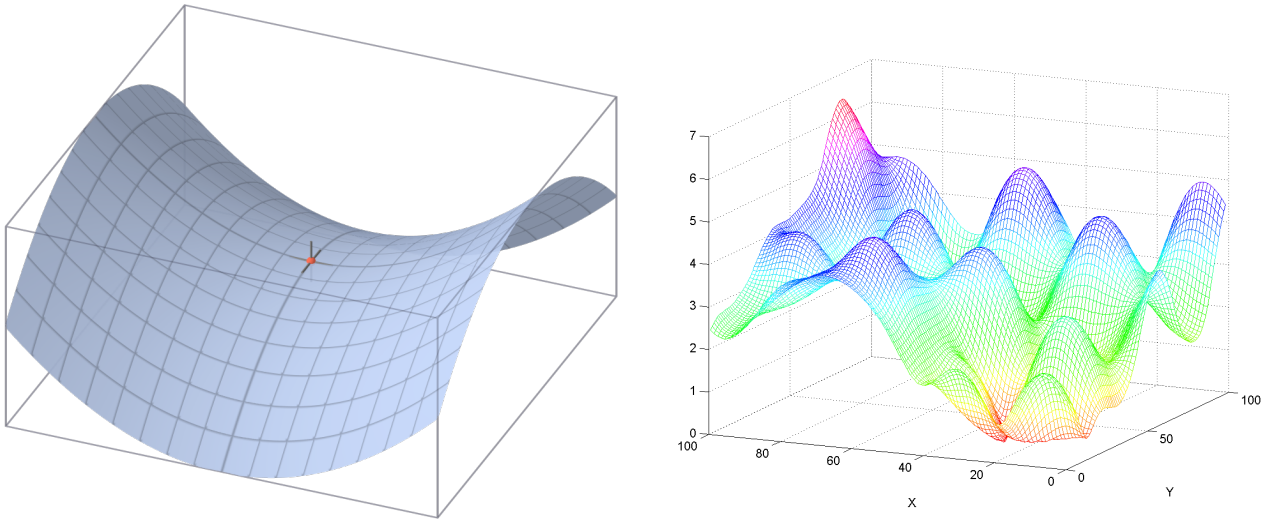
The class of global optimisation problems that are of interest in this thesis are those that have non-linear and non-convex objective functions. These functions have a large number of local minima and maxima and finding an arbitrary local optimum is a straightforward task of applying any local optimisation methods like a gradient method. However, finding the global optimum, a global minimum in our case, of such functions is a difficult task. As stated in the previous section, most optimisation algorithms are iterative in nature and perform termination tests at each iteration. As a result, all termination conditions of these algorithms are local conditions as they test whether the current solution is optimal with respect to a predefined neighbourhood. They start at an arbitrary initial point and generate a sequence of iterates that converge at a locally optimal point [2]. Therefore, to find a global minimum, the initial point should be as close as possible to the global minimiser.

Ideally, a global optimisation algorithm tries to search the whole solution space for the most optimal solution. Historically, the first methods used in global optimisation are divide-and-conquer methods and they came with the advent of computers [4]. From then, there has been an explosion of global optimisation algorithms and they can be can be split into two categories: deterministic and stochastic [4].

Most global optimisation algorithms have two separate phases. The first is the *global phase* which involves an exhaustive exploration of search space. The method used to do this can either be deterministic or stochastic. At each iteration of this global phase, a local optimisation procedure is called to identify a locally optimal point. This is known as the *local phase* and is usually deterministic. Most global optimisation methods use local optimisation methods as a tool. Most calculations take place during the local phase; therefore, a robust and reliable local optimisation algorithms with fast convergence is important [4].

1.3. Contribution of this Research

This work has dual aims. The first goal is to develop a numerically stable and globally convergent index-1 saddle-point-finding algorithm. The second aim is to construct a global optimisation algorithm that takes advantage of the Morse-theoretic properties of saddle points.



(a) Saddle Point

(b) Global Optimisation

Figure 1.5.: *Dual Aims: Saddle Point Finding and Global Optimisation Using Saddle Points (Source: Wikimedia Commons)*

The construction of the saddle-point finding method is inspired by the work of Weinan E and Zhou in [7] on Gentlest Ascent Dynamics (GAD). They present a set of differential equations whose dynamics converge toward the index-1 saddle points of the objective function.

Generally, the invariant sets or the fixed points of the dynamics of differential equations correspond to the stationary or critical points of the analogous underlying function. These critical points can be maxima or minima. Each of these fixed points have an associated basin of behaviour having all initial conditions from which the dynamics lead to or lead away from that fixed point as time goes to infinity. These basins are separated by separatices on which the dynamics converge to saddle points on the original objective function. The GAD seeks out this separatrix with the saddle-point-attracting behaviour and its solution trajectories escape the basins of attraction of minima and converge at the low-lying index-1 saddle points.

However, the GAD equations suffer from an unfavourable property in numerical implementation of easily being unstable due to some assumptions made during their construction.

The work in this thesis seeks to overcome the challenges that arise in trying to discretise and convert GAD into a robust, complete and convergent numerical algorithm.

Furthermore, a novel deterministic algorithm is proposed for the unconstrained global optimisation of non-convex and nonlinear objective functions. The method makes use of the solution trajectories of the aforementioned GAD equations that converge at saddle points. The saddle points are then used as transition points to aid exploring a multimodal function with the goal of always finding better minimisers at every iteration of its *global phase*.

The objectives of this work are:

- Construct a numerically stable and globally convergent saddle-point finding algorithm that uses GAD;

- Construct GAD-based algorithm that is able to explore any non-linear differentiable function and find all its saddle points;
- Design and construct a global optimisation algorithm that uses saddle points as transitional points;
- Compile and analyse of the numerical results from testing on the all the algorithms constructed in this work;
- Apply the constructed global optimisation algorithm on the global optimisation of the potential energy functions of Lennard-Jones clusters.

1.4. Outline of this paper

Chapters 2 to 5 contain background material relevant to this project. These chapters have a semi-didactical purpose and are meant to give a semi-substantial and intuitive overview of the necessary background material needed in this work. This was done in order to make this a self-sufficient treatise given the divergent underpinning concepts. Readers can skip the background chapters containing material they are already familiar with and go straight to chapter 5, and refer to the relevant preceding material when needed. Chapters 6 to 12 contain the main body of work.

The outline of the rest of this thesis is as follows:

- **Chapter 2**: This chapter introduces basic notions in differential topology and Morse theory required for this work;
- **Chapter 3**: This chapter offers background material on Ordinary Differential Equations and Continuous Dynamical Systems;
- **Chapter 4**: This chapter gives an overview of local and global optimisation theory, and global descent functions;
- **Chapter 5**: This chapter introduces notions on the Gentlest Ascent Dynamics pivotal to this work;
- **Chapter 6**: This chapter involves the conversion of the Gentlest Ascent Dynamics into a stable and convergent numerical algorithm;
- **Chapter 7**: This chapter deals with solving the problem of finding all the saddle points on a function;
- **Chapter 8**: This chapter concerns the design, construction and development of a global optimisation algorithm using saddle points found with the Gentlest Ascent Dynamics;
- **Chapter 9**: Contains a compilation and analysis of the numerical results from tests on the all the algorithms developed in this thesis;
- **Chapter 10**: This chapter concerns the results of applying the global optimisation algorithm on the free energy minimisation of the potential energy functions of Lennard-Jones clusters;

- **Chapter 11**: This chapter critically examines the numerical results found in the preceding two chapters in the light of the previous state of the subject matter, and includes judgments on what has been learnt in this work;
- **Chapter 12**: This is the final chapter of this treatise with conclusions, recommended future work and final thoughts.

2

Background: Elementary Differential Topology and Morse Theory

This chapter introduces the basic notions in Differential Topology and Morse Theory used in this work.

2.1. Sets, Maps and Spaces

A map or a function from a set X to a set Y is a set of ordered pairs from X and Y with a restriction that every element of X is mapped to only one element in Y, referred to as the **uniqueness of image property** [10, 11, 2].

Definition 2.1.1. (Function/Map)
A **map** $f : X \to Y$ *is a relationship such that for all* $x \in X$ *there exists a unique* $y \in Y$ *related to the given* x*.*

This function or map can be represented as $y = f(x)$ or $f : X \to Y : x \mapsto f(x)$. This can be imagined to be a set of coordinates (x, y) in the graph of a function $f(X)$ [10].

We will now generalise these sets X and Y and delve into the notion of spaces. Our treatment will be restricted to an intuitive one as intricate details are not required to understand the concepts that follow. Furthermore, delving into such details would cloud the purpose of this chapter.

The best example of a space is the three-dimensional Euclidean space model of the physical universe in which we exist. It considers any three directions that do not lie in the same plane. This is basically a set of triples representing the coordinates in this three-dimensional *point space*. This is referred to as the $\mathbb{E}^3$, the three-dimensional Euclidean space. With this space, we can also define a *vector space*

where, intuitively, a vector is a directed arrow from one point in $\mathbb{E}^3$ to another. Therefore, we have a vector space of triples $\mathbb{R}^3$ and a point space of triples $\mathbb{E}^3$. The important fact to grasp is that $\mathbb{E}^3$ consists of points represented by coordinates $p = (p_1, p_2, p_3)$ while the directed difference between a pair of such points p, q is a vector in $\mathbb{R}^3$ $\mathbf{pq}$ or $\mathbf{q} - \mathbf{p}$ with components $(q_1 - p_1, q_2 - p_2, q_3 - p_3)$ [10].

This can be generalised to dimensions higher than 3. So, we can have have point spaces and vector spaces $\mathbb{E}^n$ and $\mathbb{R}^n$, respectively, where $n = 1, 2, 3, 4, ...$; of course, in dimensions higher than 3, the extra directions will arise from other features than ordinary space – such as time, temperature, pressure, among others [10].

The spaces described above are examples of a larger class of spaces known as *topological spaces* whose precise definition we will not delve into. In general, the spaces $\mathbb{E}^n$ and $\mathbb{R}^n$ are the same space. However, it is important to note that there are spaces, such as *manifolds* and *metric spaces*, that are specialisations of topological spaces with extra constraints. The examples given above are both examples manifolds and metric spaces.

Now we can define maps of the form $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n, m = 1, 2, 3, 4, ...$ where the relationship is between spaces or manifolds.

2.2. Parameterisation of Functions

This section introduces the concept of parametric functions, an important tool in many areas of mathematics. To appreciate why one may need to parameterise a function, let's begin with an arbitrary function $y = f(x)$.

As described earlier, and as can be seen from the figure 2.1, a curve is presented as a graph of a function $f(x)$. As x is varied, $y = f(x)$ is computed by the function f, and the pair of coordinates (x, y) sweeps out the curve. This is called the explicit form of the curve [12].

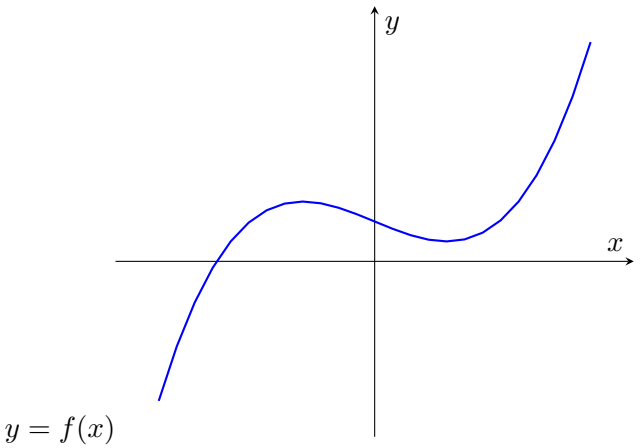


Figure 2.1.: *Illustartion of a curve for the function* $y = f(x)$

The problems with using this explicit form can be best illustrated by using a circle's representation. The limitations of the explicit form are:

- **Single-Valued due to Uniqueness Property**: Based on the definition of a function, the curve is **single-valued** along any line parallel to the y axis. For example, only parts of the circle may be defined explicitly as shown in figure 2.2 [12].

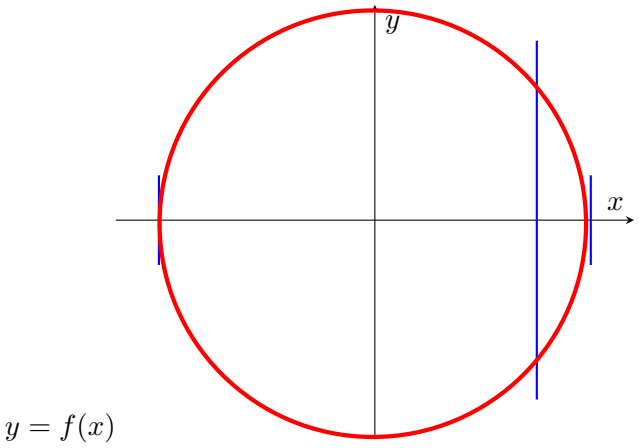


Figure 2.2.: *Illustration of the Single-Valued due to Uniqueness Property on a Circle* $y = f(x)$

- **Infinite Slope**: An explicit curve cannot have infinite slope; the derivative $f'(x)$ is not defined parallel to the y-axis. Hence there are two points on the circle that cannot be defined as shown in figure 2.3 [12].

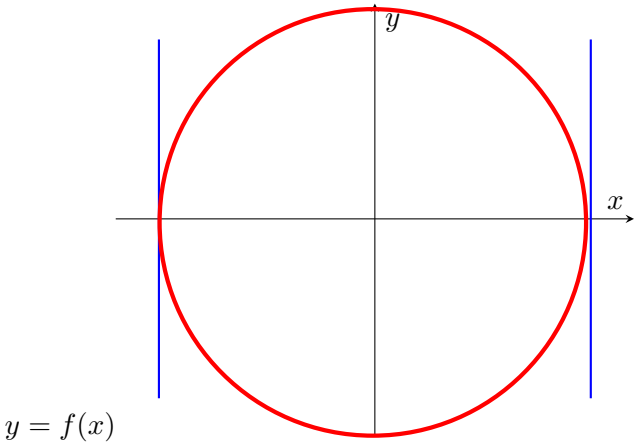


Figure 2.3.: *Illustration of the Infinite Slope on a Circle* $y = f(x)$

- **Transformation Problems**: Any transformation, such as rotation or shear, may cause an explicit curve to violate the two points above [12].

The problems outlined above can be overcome by using an alternative representation of a function. This done using *parametric functions*. This form of representation provides a method, known as *parameterisation*, that defines motion on the curve defined by the function. The motion on the curve refers to the way that the point (x, y) traces out the curve. This parameterisation uses an independent parameter or variable to compute points on the curve. It gives the *motion* of a point on the curve [10, 11, 13].

2.3. Curves and Surfaces

Let's consider a curve that lies in the $X - Y$ plane. It can be defined by two functions, $x(t)$ and $y(t)$, which use the independent parameter t. $x(t)$ and $y(t)$ are coordinate functions, since their values represent the coordinates of points on the curve. As t varies, the coordinates $(x(t), y(t))$ sweep out the curve [12, 13].

The independent parameter t can be any suitable variable such as a number, an angle, or even a length but it must satisfy the following conditions:

1. Each point on the curve must be related to a unique value of the parameter [13];
2. Each value of the parameter must give coordinates of only one point on the curve [13].

Any of the conditions above can be relaxed to facilitate a variety of properties. The explicit form of the equation can be found by eliminating the parameter t [13].

A good physical model for parametric curves is that of a moving particle and we will adhere to this convention for the rest of the discussion. In this model the parameter t represents time. Thus, at any time t, the position of the particle is denoted as $(x(t), y(t))$ [12].

Now let's consider manifolds $\mathbb{E}^n$ and $\mathbb{R}^n$. If we have a point $x \in \mathbb{E}^n$ and vector $y \in \mathbb{R}^n$, then there exists another point $z \in \mathbb{E}^n$ such that:

$$z = x + y$$

Therefore the line segment from x (in the direction of vector y) to q, parameterised by $t \in [0, 1]$, is given by:

$$LS : [0, 1] \to \mathbb{R}^n : t \mapsto x + ty$$

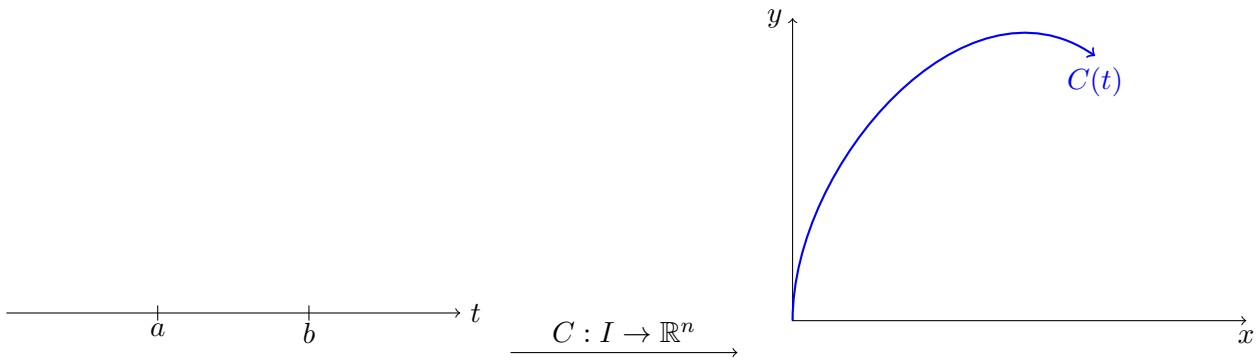
The manifolds $\mathbb{E}^n$ and $\mathbb{R}^n$ can be used interchangeably since in this context they refer to the same space; we only distinguished them in earlier discussions to illustrate notions of points and vectors. We can now define a curve.

Definition 2.3.1. (Curve)
A parameterised continuous **curve** *in* $\mathbb{R}^n$ $(n = 2, 3, ...)$ *is a continuous map* $C : I \to \mathbb{R}^n$ *, where* $I \subset \mathbb{R}$ *is an open interval (of end points* $-\infty \leq a < b \leq \infty$*) [14].*

This can be imagined to be as shown in figure 2.4.

It is imperative to that we distinguish the notion of a curve and its trace. Using our physical world model, a curve describes the motion of a particle in n-space, and the trace is the trajectory of the particle. If the particle follows the same trajectory, but with different speed or direction, the curve is considered to be different. Therefore, the curve is the map C and it's trace is the map's image set [10, 12, 14].

For the purposes of our discussion, we required that the curve is continuous because we want to assume that a curve is differentiable. This enables us to have a way of finding the velocity of



(a) Set of parameter values t (b) Map of the curve C (c) Trace of the curve $C(t)$

Figure 2.4.: *Illustration of a parameter set t, map of a Curve C and its trace $C(t)$*

a particle $C'(t)$ and its acceleration $C''(t)$ [10, 12]. A function that is differentiable to any desired degree is referred to as a **smooth function**. We will consider these type of functions in this chapter.

From this we can now explore the idea of a **surface**. The notion of a surface is a generalisation of that of a curve: a surface is a continuous image of a product of (any kinds of) intervals; intuitively, we can imagine that we make surfaces like a patchwork quilt, from a collection of overlapping pieces, each piece being a curve [10, 14]. For the purpose of this discussion, we will restrict ourselves to dimensions 2 and 3.

Definition 2.3.2. (Surface)
A parametrized continuous **surface** *in $\mathbb{R}^3$ is a continuous map $S : U \to \mathbb{R}^3$, where $U \subset \mathbb{R}^2$ is an open, non-empty set [14].*

It is often be convenient to consider the pair of parameters $(u, v) \in U$ to be a set of coordinates of points in a $u - v$ plane and the surface S to the image $S = \sigma(U)$ [10].

2.4. Morse Theory

Morse theory investigates how functions defined on a manifold are related to the geometric aspects of that manifold [15]. In other words, it is a tool that analyses the topology of a manifold by using differentiable functions defined on it. In the section we will look at Morse theory on surfaces.

2.4.1. Basics: One-Dimensional Case

Let us consider a function $y = f(x)$ in one variable, where $x, y \in \mathbb{R}$. We know that we can use calculus to study the increase and decrease of f by first differentiating it to get $f'(x)$, and then finding x_0 that satisfies $f(x_0) = 0$, and then study the changes of $f'(x)$ around x_0 [15].

Definition 2.4.1. (Critical Point of a Function of One Variable)
A point x_0 that satisfies

$$f'(x_0) = 0$$

is called a **critical point**.

To study the changes of $f'(x)$ around x_0, we make use of the second derivative $f''(x)$. With this we can define types of critical points.

Definition 2.4.2. (Critical Point Degeneracy for One-Dimensional Functions) *Let point x_0 be a critical point. If*

$$f''(x_0) = 0$$

then x_0 is called a **degenerate** *critical point. If*

$$f''(x_0) \neq 0$$

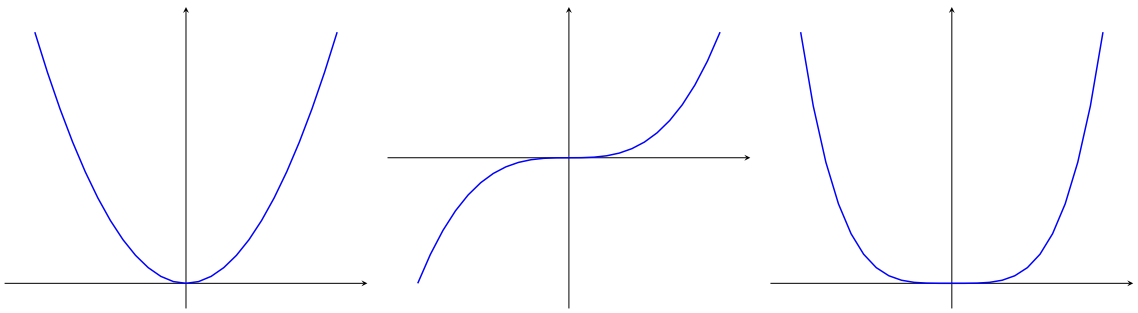
then x_0 is called a **non-degenerate** *critical point.*

To get an intuitive handle on what this means, let us consider real-valued functions in one variable of the form $y = x^n$ with $n = 2, 3, ...$ where x and y are real numbers.

The quadratic function $y = x^2$ has the critical point $x = 0$ ($y' = 2x$, and $y' = 0$ when $x = 0$) and the second derivative is $y'' = 2$. Therefore, the critical point $x = 0$ of the quadratic function $y = x^2$ is non-degenerate [15].

For functions with $n \geq 3$, the critical point $x = 0$ is degenerate since the second derivative of those functions $y = x^n$ has the form $y'' = n(n-1)x^{n-2}$ which is zero at $x = 0$ [15].

Now, looking at the graphs of functions $y = x^2$, $y = x^3$ and $y = x^4$ in figure 2.5, we notice that there is a higher degree of tangency to the horizontal x-axis for degenerate critical points than for non-degenerate critical points [15]. From the figure, the tangency of the horizontal axis at $x = 0$ for $y = x^3$ and $y = x^3$ is greater than that of for $y = x^2$ at $x = 0$.



(a) $y = x^2$ (Non-degenerate) (b) $y = x^3$ (Degenerate) (c) $y = x^4$ (Degenerate)

Figure 2.5.: *Graphs of functions $y = x^2$, $y = x^3$ and $y = x^4$ to illustrate critical point degeneracy*

Another difference between degenerate and non-degenerate critical points arises when we perturb the functions functions slightly [15]. Let's consider only two of those functions; $y = x^2$ which has a non-degenerate critical point at $x = 0$ and $y = x^3$ which has a degenerate critical point at $x = 0$. Let's then change them by adding a linear function $y = ax + b$ [15].

For the quadratic function $y = x^2$ the perturbation yields:

$$y = x^2 + ax + b.$$

Differentiating it gives

$$y' = 2x + a;$$

thus the critical point is

$$x = -\frac{a}{2}$$

which is non-degenerate as $y'' = 2$. Therefore, after perturbation, the critical point $x = -\frac{a}{2}$ that appears near the non-degenerate critical point $x = 0$ is also non-degenerate. We notice that the "non-degeneracy" is "preserved".

For the cubic function $y = x^3$, the changes give:

$$y = x^3 + ax + b.$$

Differentiating yeilds

$$y' = 3x^2 + a,$$

and setting y' to zero gives

$$x = \pm\sqrt{\frac{-a}{3}}.$$

For $a > 0$ no real solutions exists; thus, after the change, we lose critical points for $a > 0$.

For $a < 0$ real solutions exists and we obtain two critical points. Since $y'' = 6x$, the critical points take non-zero values and are thus non-degenerate.

After perturbation, the degenerate critical point $x = 0$, for $a > 0$, disappears and for $a < 0$ it splits into two non-degenerate critical points $x = \pm\sqrt{\frac{-a}{3}}$. Of course, all this depends on the way we perturb the function [15].

We observe that the "degeneracy" is not preserved after perturbation.

The main point to take away from this illustration is that non-degenerate critical points are "*stable*" in that they are predictable; on the other hand, degenerate critical points are "*unstable*" due to their unpredictable nature [15].

2.4.2. Two-Dimensional and Multi-Dimensional Case

As stated earlier, our major concern is the Morse theory on surfaces. Therefore, we need to extend the ideas from the previous section to a higher dimensional case. An analysis in two-dimensions will suffice for our current exposition because the goal is not to be rigorous but illustrative and intuitive. Of course, all the concepts that will be illustrated using examples and definitions in two-dimensions can be naturally extended to higher dimensions.

Let us consider a two-dimensional real-valued smooth function

$$z = f(x, y) \tag{2.1}$$

where x and y are real numbers. It would be helpful if we considered the pair (x, y) to be a point on an XY-plane so that the function (2.1) is defined on this plane. Thus, the function will be such that it assigns each point on this plane to a real number — or, intuitively, to its elevation as shown in the figure 2.6 [15].

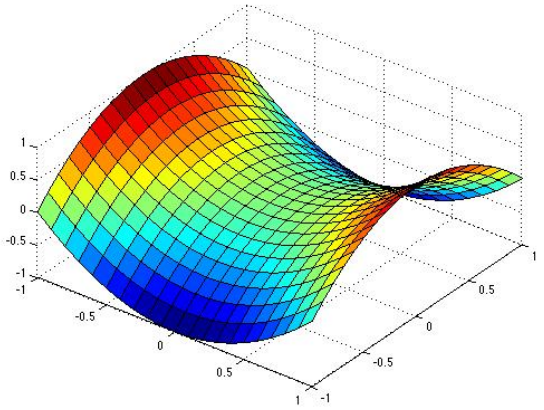


Figure 2.6.: *Surface of function* $z = x^2 - y^2$

We can now to extend the previously defined idea for this case.

Definition 2.4.3. (Critical Points of Functions of Two Variables)
A point $p_0 = (x_0, y_0)$ *in the* XY*-plane is a* **critical point** *of a function* $z = f(x, y)$ *if:*

$$\frac{\partial f}{\partial x}(p_0) = 0, \ \frac{\partial f}{\partial y}(p_0) = 0.$$

or

$$\nabla f(x_0, y_0) = 0 \tag{2.2}$$

[15]

Having done this, a subsequent natural question is: how do we now define *degenerate* and *non-degenerate* critical points for functions of two variables?

Firstly, we know that the *degeneracy* is specified by the second derivative. Thus, for the multivariate case, it will be specified by the characteristics of the *Hessian* matrix.

Definition 2.4.4. (Critical Points Degeneracy)

1. *Suppose a point* $p_0 = (x_0, y_0)$ *is a* **critical point** *of a function* $z = f(x, y)$*. The matrix of second order partial derivatives evaluated at* p_0

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x^2}(p_0) & \frac{\partial^2 f}{\partial x \partial y}(p_0) \\ \frac{\partial^2 f}{\partial x \partial y}(p_0) & \frac{\partial^2 f}{\partial y^2}(p_0) \end{bmatrix}$$

is called the **Hessian**, *which we will denote as* $\nabla^2 f$.

2. *A critical point* p_0 *of* $z = f(x, y)$ *is* **degenerate** *if the Hessian is* **singular***; that is, the determinant of the Hessian is equal to zero, that is:*

$$\textbf{det } \nabla^2 f(x_0, y_0) = 0 \tag{2.3}$$

3. *A critical point* p_0 *of* $z = f(x, y)$ *is* **non-degenerate** *if the Hessian is* **non-singular***; that is, the determinant of the Hessian is* **NOT** *equal to zero, that is:*

$$\textbf{det } \nabla^2 f(x_0, y_0) \neq 0 \tag{2.4}$$

[15]

To extend these concepts, we will now define a Morse function and state an important lemma in Morse Theory.

Definition 2.4.5. (Morse Function)
A smooth scalar function is called a Morse function if all of its critical points are non-degenerate [16].

Lemma 2.4.1. (The Morse Lemma)
Let p_0 *be a non-degenerate critical point of a function* f *of two variables. Then we can choose a local coordinate system* (X, Y) *such that we can express the function* f *in terms of* (X, Y) *in one of these three forms:*

1. $f = X^2 + Y^2 + c$,
2. $f = X^2 - Y^2 + c$,
3. $f = -X^2 - Y^2 + c$,

where c *is a constant (* $c = f(p_0)$ *) and the* p_0 *is the origin origin is* $p_0 = (0, 0)$ *[15].*

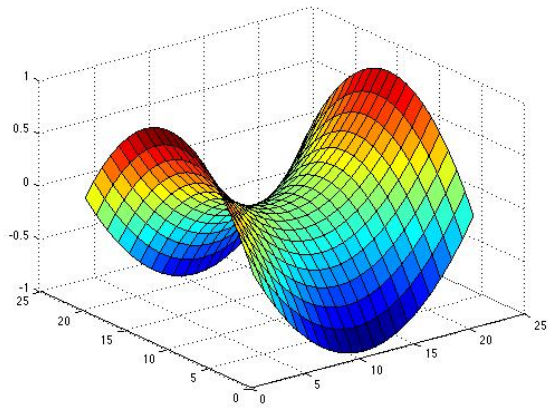
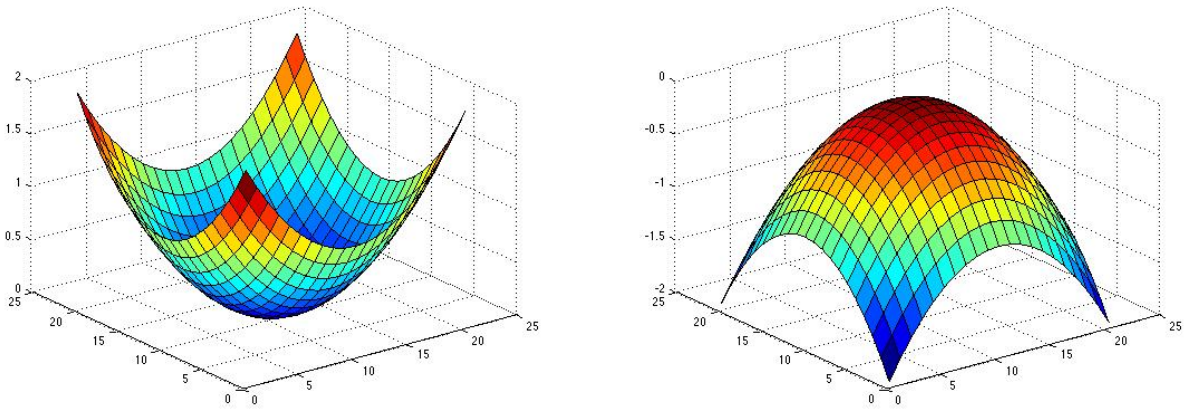


Figure 2.7.: *Graph of* $z = x^2 - y^2$ *(saddle point)*

The main gist of this lemma is that a function looks simple near a non-degenerate critical point; that is, using a suitable coordinate system, a function of two variables can be transformed into a simple form that aids in having an intuitive handle on the nature of non-degenerate critical points.



(a) $z = x^2 + y^2$ (minima)

(b) $z = -x^2 - y^2$ (maxima)

Figure 2.8.: *Graphs of* $z = x^2 + y^2$ *and* $z = -x^2 - y^2$

Figures 2.8 and 2.7 show how these forms look like with $c = 0$. One important observation is that after this transformation one observes that there is no other critical point near the origin, which is the non-degenerate critical point. In other words, one deduces that non-degenerate critical points are isolated and are thus easier to analyse [15].

We now define an important concept related to non-degenerate critical points.

Definition 2.4.6. (Index of a Non-Degenerate Critical Point)
Let p_0 *be a non degenerate critical point of a function* f *in two variables.*
Let the function f *be transformed into a form specified by Morse lemma.*
Then the index of a non-degenerate critical point p_0 *is*
0 *for* $f = X^2 + Y^2 + c$;
1 *for* $f = X^2 - Y^2 + c$; *and*
2 *for for* $f = -X^2 - Y^2 + c$.

It's fairly obvious that the index of the critical point is analogous to the number of *minuses* in the transformed equation. To get an intuitive feel of the concepts presented thus far, we'll use a classic example in Morse theory.

Let's consider a mountainous landscape. The peaks correspond to the maxima, the basins to minima and passes to saddle points — all critical points. Now let's imagine that there is an explorer who has a large ball and she is looking for points on this landscape where where she could place this ball and it would remain stationary. The answer is fairly obvious that the points she is looking for are the critical points. If the ball is placed at any other points then it would roll downhill under the influence of gravity. So, the question now becomes how stable are these critical points in relation to keeping the ball stationary? The mountain peak is obviously the most unstable since there are many directions that the ball could roll off. The basin is the most stable since if a ball is placed there and is pushed, it returns to the basin. So, there are no unstable directions there. The pass, on the other hand, has a degree of instability midway between that of the basin and the mountain peak since it curves upwards in one direction and downwards in another. This notion of "unstable directions" or "degree of instability" encapsulates the idea of the index of a

non-degenerate critical point. Thus, building from the definition given earlier, the index of the basin (minima) is 0, the index of a pass (saddle point) is 1, and the index of the mountain peak is 2 [15, 17, 18, 16]. Figures 2.8 and 2.7 illustrate this.

Formally, the degree of instability is defined as the number of negative eigen-values of the Hessian matrix [16]. This corresponds to the intuitive notion that the index is the number of directions in which a function f decreases at a critical point.

One question that Morse theory tries to address is the classification of critical points. So, if we restrict our attention to Morse functions then we will be able to classify all critical points based on their degree of instability. This restriction is justified because it can be shown that any smooth scalar function can be slightly perturbed to remove all degeneracies and hence obtain a Morse function [15, 16].

2.5. Summary

This chapter covered the basics of differential topology and Morse theory that are used in this work. It starts by introducing notions of maps and spaces and building on that to introduce parameterised curves and surfaces. The rest of the chapter delves in to the ideas of critical point stability in relation to the manifold that a map is defined on and this is used to introduce the notion of degenerate and non-degenerate critical points and their indices.

3

Background: Ordinary Differential Equations and Continuous Dynamical Systems

This chapter explores the background on differential equations and continuous dynamical systems used in this work.

3.1. Introduction

Differential equations are important in mathematics and engineering because many physical laws and relationships in nature appear mathematically in the form of differential equations [19, 20, 21]. They usually arise after the process of formulating a problem as a mathematical expression in terms of variables, functions, among other things. The resulting expression is referred to as a model. If this model is an equation which contains one or several derivatives of various orders of an unknown function then this model is a differential equation [19, 20].

3.2. Ordinary Differential Equations

The main concern of this section is that of ordinary differential equations because they are very well suited for numerical computation. We start by defining them.

Definition 3.2.1. (Ordinary Differential Equation - ODE)
An Ordinary Differential Equation (ODE) is an equation containing an unknown function of a single independent variable and its derivatives [19, 20].

The term *ordinary* is used to show a distinction from other differential equations that may be defined in terms of an unknown function dependent on several variables. This other type of differential equation is said to contain *partial* derivatives and are therefore called partial differential

equations [20].

In order to make our discussion more illustrative, let us denote the unknown function to be $y(x)$ where x is independent variable and y is the dependent variable. This will allow us to denote the unknown function alternatively as $y = y(x)$.

An ODE is said to be of order n if the nth derivative of the unknown function $y(x)$ is the highest derivative of $y(x)$ in the equation. As one can imagine, this can make the study of ODEs an extremely sophisticated one because of the many forms that they can take. We will therefore try to classify ODEs based on the various forms that they can take as this will help us have a more structured grasp on how to deal with these varied forms.

Definition 3.2.2. (Explicit Form of an ODE)
Let F be a given function of x, y, and derivatives of y. Then an equation of the form

$$y^{(n)} = F(x, y, y', y'', ..., y^{(n-1)}) \tag{3.1}$$

is called an **explicit** *ordinary differential equation of order n [19, 20].*

Definition 3.2.3. (Implicit form of an ODE)
Let F be a given function of x, y, and derivatives of y. Then an equation of the form

$$F(x, y, y', y'', ..., y^{(n)}) = 0 \tag{3.2}$$

is called an **implicit** *ordinary differential equation of order n [19, 20].*

Unfortunately, there are many more classifications than the ones given above.

Definition 3.2.4. (Linear ODE)
An ODE is said to be linear if it can be written as a linear combination of the derivatives of y:

$$y^{(n)} = \sum_{i}^{(n-1)} a_i y^{(i)} + r(x) \tag{3.3}$$

where $a_i(x)$ and $r(x)$ are functions of x. In engineering, the function $r(x)$ is usually called the **input** *or the* **source term** *and the unknown function $y(x)$ is usually called the* **output** *or the* **response term** *[19, 20].*

Naturally, it follows that ODEs that cannot be written in the form specified above are referred to as **Non-linear** ODEs.

This introduction of the idea of source term $r(x)$ that influences the response term $y(x)$ allows us to classify ODEs further.

Definition 3.2.5. (Homogeneous ODE)
An ODE is said to be homogeneous if its source term $r(x) = 0$ [19, 20].

Definition 3.2.6. (Non-Homogeneous ODE)
An ODE is said to be non-homogeneous if its source term $r(x) \neq 0$ [19, 20].

The source term $r(x)$ is simply the part of the ODE that is a function of x and is also independent of any derivative of y. This classification is important as it helpful in determining how a particular ODE will be solved.

In our discussion thus far, we have considered the variable x to be the independent variable and y to be dependent variable where the independent variable x appears explicitly in the ODE. However, there is a class of ODEs which do not explicitly contain the independent variable (usually denoted by time t).

Definition 3.2.7. (Autonomous ODE)
An autonomous ODE is an ODE of the form

$$\frac{d}{dt}x(t) = f(x(t)) \tag{3.4}$$

where $x \in \mathbb{R}^n$ and t is usually time [19, 20, 22, 23]. The usual shorthand for $\frac{d}{dt}x(t)$ is $\dot{x}$.

$$\frac{d}{dt}x(t) = \dot{x}$$

ODEs are essentially *equations* and thus it is imperative that we take time to discuss what a solution to an ODE is.

Definition 3.2.8. (Solution of an ODE)
A function

$$y = h(x)$$

is a solution of the ODE

$$F(x, y, y', y'', ..., y^{(n)}) = 0$$

on some open interval $x \in (a, b)$ if $h(x)$ is continuous and n-times differentiable over this interval and

$$F(x, h, h', h'', ..., h^{(n)}) = 0$$

[19, 20].

A graph of a solution is called a **solution curve**. As one can imagine, obtaining solutions to an ODE involves integration — a process that introduces an arbitrary constant. Solutions that include this arbitrary constant are called **general solutions** of the ODE. Geometrically, general solutions of ODEs are an infinite number of solution curves, one for each possible value of the arbitrary constant, and they are usually called a *family of solutions* as illustrated by figure 3.1.

A solution corresponding to specific instantiation of c is called a **particular solution** of the ODE and it does not contain any arbitrary constants.

In many cases it is possible to obtain a unique solution of a given ODE from its general solution by using an initial condition that is usually specified as $y_0 = y(x_0)$. In this case, the values x_0 and y_0 are specified and are used to find the value of the arbitrary constant.

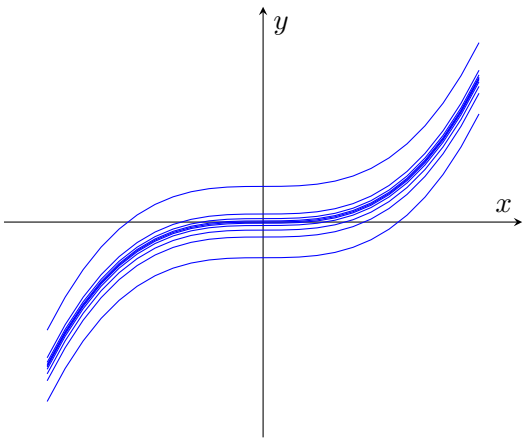


Figure 3.1.: *Example of Solution Curves*

Definition 3.2.9. (Initial Value Problem - IVP)
An initial value problem is an ODE together with an initial condition which is the specified value of the unknown function at a given point in the domain of the solution. It is expressed in the form

$$y^{(n)} = F(x, y, y', y'', ..., y^{(n-1)}), \qquad y(x_0) = y_0 \tag{3.5}$$

[19] [20].

Geometrically, an initial value problem (IVP) just specifies that the solution should pass through a particular point (x_0, y_0) in the space where the solution is defined [20].

3.3. Qualitative Analysis of Ordinary Differential Equations

In this section we will explore the qualitative aspects of solutions to ODEs by exploring their geometric features when they are graphed. A first-order ODE

$$y' = f(x, y)$$

has a very simple geometric interpretation. We know from calculus that the derivative $y'(x)$ of $y(x)$ is the slope or the gradient of function $y(x)$ or the slope of the line that's a tangent at a point x on the function. Therefore, if a solution passes through a point (x_0, y_0) then it follows that

$$y'(x_0) = f(x_0, y_0).$$

If we represented the gradient $y'(x)$ by short directed lines inclined at the specified gradient and located tangent to a particular point in the xy-plane, we can generate a graphical representation of the solutions of the ODE above. This representation is called a **Slope Field** (or **Direction Field**). We can then approximate solution curves passing through the slope fields.

Definition 3.3.1. (Slope/Direction Field)
A slope field (or direction field) is a graphical representation of the solutions of a first-order ODE [19] [20].

Using slope fields can be viewed as a creative way to plot a real-valued function of two real vari-

ables $f(x, y)$ in the xy-plane. For a given pair (x, y), a vector with the components $\begin{bmatrix} 1 \\ f(x,y) \end{bmatrix}$ is drawn at the point on the x, y-plane. Usually, the vector is made to be of unit length in order to make the plot better looking for a human eye. The advantage of using slope fields is that one can approximate the solution to an ODE without actually solving it [19, 20, 22, 23].

Let us consider a simple example of a first order ODE

$$\frac{dy}{dx} = 2x.$$

In order to generate the direction fields of this ODE in a structured fashion, let us start by finding the values of x where $\frac{dy}{dx} = 0$. It's fairly obvious that we are looking for $x = 0$. So, at points where $x = 0$ in the xy-plane we plot horizontal directed vectors $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. We then pick values where $x > 0$ and $x < 0$ and plot vectors $\begin{bmatrix} 1 \\ \frac{dy}{dx} \end{bmatrix}$ in the same fashion. The resulting plot should look like figure 3.2.

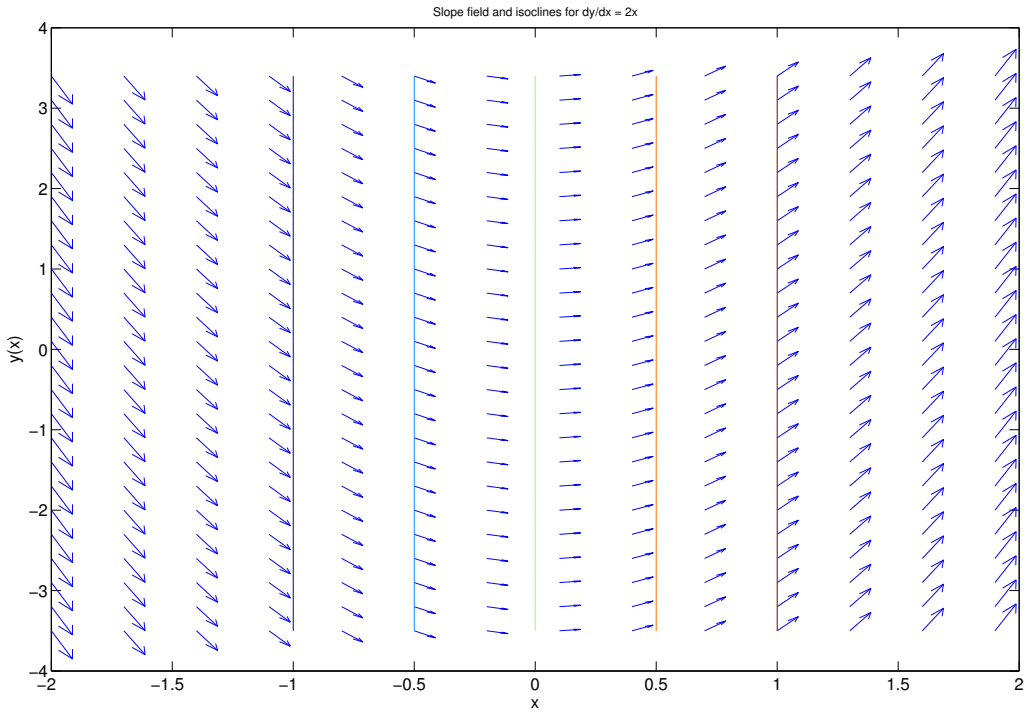


Figure 3.2.: *Direction Field of non-autonomous ODE* $\frac{dy}{dx} = 2x$

So why should we be interested in direction fields? There are two main reasons for this.

1. **Graphical Sketch of solutions**. Since the vectors in the direction fields are in fact tangents to the actual solutions to the ODE we can use these as guides to sketch the graphs of solutions [19].

2. **Examine Long Term Behavior**. There may be cases where we are not interested in the

actual solutions to the ODEs but in how the solutions behave as time passes. Direction fields can be used to find information about this long term behaviour of ODE solutions [19].

The first point is quite easy to understand from the example we have just looked at. If we were interested in a particular solution that passed through a point (x_0, y_0), we could simply pick it out from the generated directed field.

The second point is a more interesting one since it speaks of the collective behaviour of all the solutions of an ODE. Notice that there is an idea of time involved. For this, we need to remember **autonomous** ODEs — ODEs that have an independent variable that does not appear explicitly in the ODE. In our case, this independent variable is time t.

Let us consider as example an autonomous ODE

$$\dot{y} = \frac{dy(t)}{dt} = y' = (y-1)(y-2).$$

As before, in order to generate the direction fields of this ODE in a structured fashion, let us start by finding the values of y when $y' = 0$, that is, solve $f(x, y) = (y-1)(y-2) = 0$. As shown in figure 3.3, the values of y where $y' = 0$ are $y = 1$ and $y = 2$. At these values, we draw the direction fields as horizontal vectors $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

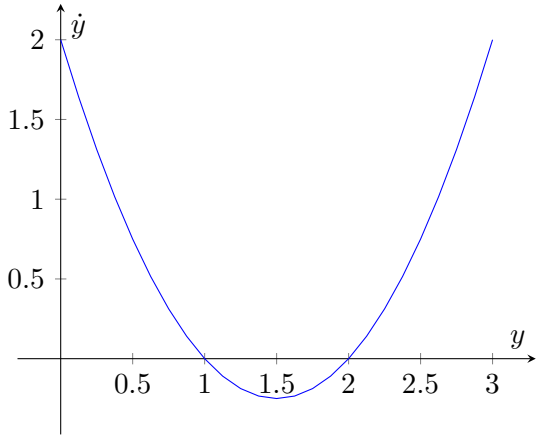


Figure 3.3.: *Plot of parabola* $f(x, y) = (y-1)(y-2)$

Let us denote $y_1 = 1$ and $y_2 = 2$. We can now examine the values of y' around y_1 and y_2. We can start by choosing values of $y > y_2$, say $y = 2.5$, and plot the associated vectors. We continue with the process on values of $y_1 < y < y_2$ and also $y < y_1$ and plot the vectors $\begin{bmatrix} 1 \\ f(x,y) \end{bmatrix}$ and the resulting direction field plot will look something like what is shown in figure 3.4.

Before we can discuss the implication of what we have found, let us make a few observations. Firstly, the autonomous ODE y' has constant solutions at points where $\dot{y} = y' = 0$. This is because if we have $y' = 0$ then, by integration, the solution is $y = \textit{constant}$. These solutions are called **equilibrium solutions** or **equilibrium points**. They are also called **critical points** [20].

Definition 3.3.2. (Equilibrium Point/Solution))

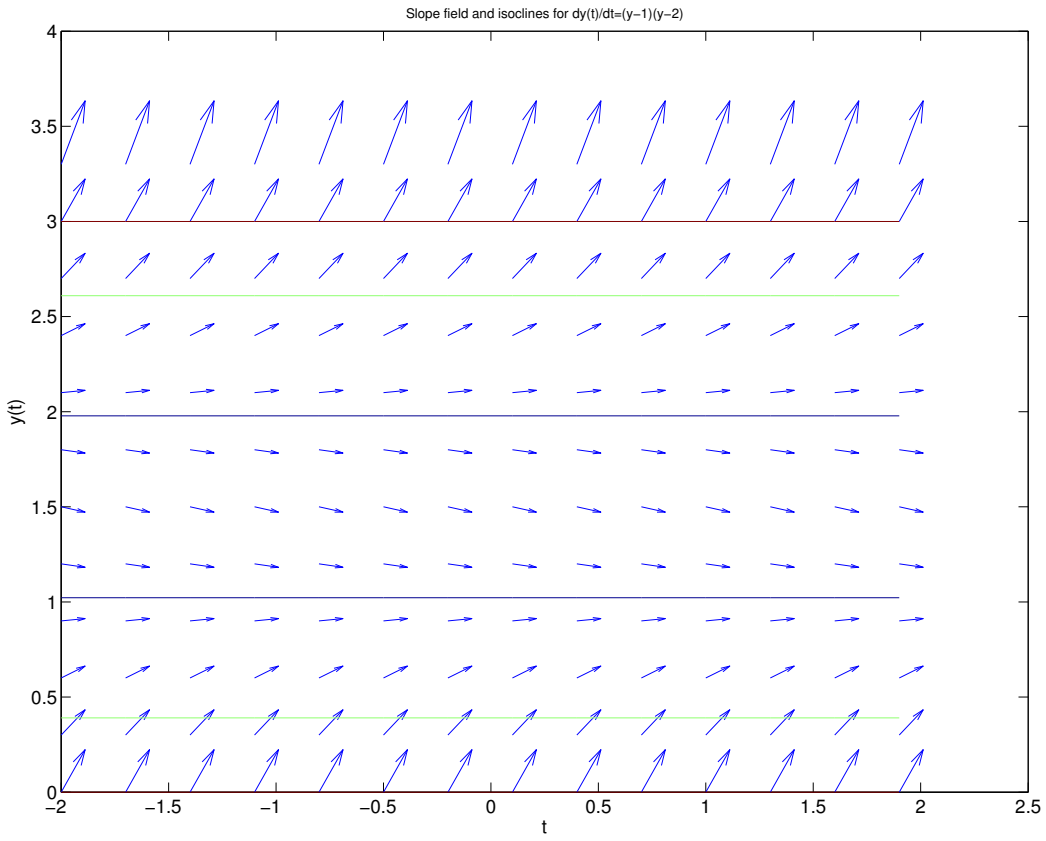


Figure 3.4.: *Direction Field of autonomous ODE* $\dot{y} = (y-1)(y-2)$

If

$$\dot{x} = \frac{dx(t)}{dt} = f(x),$$

then the equilibrium points of the ODE are solution(s) to the equation

$$\dot{x} = 0.$$

From figure 3.4, we notice the direction field shows the behaviour of the solution as t increases. The "long term behaviour" of the solutions is dependent on the initial value of y at time $t = 0$, $y(0)$. This is called the **initial condition** [19, 20].

For solutions where $y(0) < 1$, we notice that $y \to 1$ as $t \to \infty$. And for those with initial conditions in the range $1 < y(0) < 2$, $y \to 1$ as $t \to \infty$. Furthermore, for $y(0) > 2$, we can see that $y \to \infty$ as $t \to \infty$.

Value of $y(0)$	**Behaviour as** $t \to \infty$
$y(0) < 1$	$y \to 1$
$1 < y(0) < 2$	$y \to 1$
$y(0) > 2$	$y \to \infty$

The equilibrium solutions $y = 1$ and $y = 2$ have different characteristics in terms of how solutions with initial conditions near them behave. Solutions close to $y = 1$ for some t remain close to it for all further t. Therefore, $y = 1$ is a **stable** equilibrium solution. The solution $y = 2$, on the other

hand, is **unstable** because all solutions close to it at time t do not remain close to it for further t.

Definition 3.3.3. (Stable Equilibrium Solution)
An Equilibrium solution is **stable** *if solutions close to it at time t remain close to it for all further times t [20].*

Definition 3.3.4. (Unstable Equilibrium Solution)
An Equilibrium solution is **unstable** *if solutions close to it at time t do not remain close to it for all further times t [20].*

3.4. Continous Dynamical Systems

The classification of the equilibrium solutions adds a new dimension to our study of ODEs — an aspect of how solutions of an ODE evolve in time. This idea of studying the long term behaviour of solutions to ODEs has its roots in the study of motion of particles and systems of this kind are referred to as **Dynamical Systems**. Generally, dynamical systems are mathematical rules that describe the time dependence of a point's position in its surrounding space [22, 23].

Dynamical systems theory is important because it aids in the study of evolution of processes in time. Despite being motivated by autonomous ODEs, there is a much more general and broader treatment that it gives to this evolution behaviour. There is a treatment that deals with discrete time steps, and other variants that deal with other ideas of what form the time should take. In this chapter, we are interested in dynamical systems that we can construct from autonomous ODEs. These dynamical systems are called *continuous dynamical systems*. Before we unify what we have looked at in ODEs with the notions we are about to introduce, let's define a continuous dynamical system.

Definition 3.4.1. (Continous Dynamical System)
Let $M \subset \mathbb{R}^n$. A continuous dynamical system is a map

$$D : \mathbb{R} \times M \to M.$$

It is also called a **flow** *[23].*

A concept central to dynamical system is that of evolution **trajectories** or **orbits**. These are essentially parameterised curves. Using the previous example of the analysis of the ODE $\dot{y} = (y-1)(y-2)$, trajectories are analogous to the solutions or the direction fields of the ODE because they encapsulate the general behaviour of the solutions of the ODE with time. The space having these trajectories is called the **phase space**. A phase portrait corresponding to the example is shown in figure 3.5. Therefore, given an initial point it is possible to determine all its future positions. And this collection of points is what we call a trajectory or orbit [19, 20, 22, 23, 24].

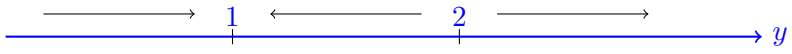


Figure 3.5.: *Phase portrait for $\dot{y} = (y-1)(y-2)$*

We observed that the equilibrium solutions tended to be unique in that solutions with initial

conditions near them presented interesting properties. With dynamical systems, the equilibrium orbits are also called **fixed points**.

Definition 3.4.2. (Stable Fixed Point)
A fixed point P_0 is **stable** *if trajectories close to it at time t remain close to it for all further t; precisely, if for every disk D_ϵ of radius $\epsilon > 0$ and centre P_0 there is a disk D_δ of radius $\delta > 0$ and also with centre P_0 such that every trajectory of the dynamical system that has a point P_1 (corresponding to, say, time $t = t_1$) in D_δ has all its points corresponding times $t \geq t_1$ in D_ϵ.*

Fixed point P_0 is called **stable and attractive** *or* **asymptotically stable** *if P_0 is stable and every trajectory that has a point in D_δ approaches P_0 as $t \to \infty$ — it is called an* **attractor** *[20].*

Definition 3.4.3. (Unstable Fixed Point)
A fixed point P_0 is **unstable** *if trajectories close to it at time t do not remain close to it for all further t.*

Fixed point P_0 is called a **repeller** *or* **asymptotically unstable** *if P_0 is unstable and every trajectory that has a point in D_δ moves away from P_0 as $t \to \infty$ [20, 23, 24].*

The definitions above introduce several interesting new ideas. Firstly, the most obvious is the idea of being asymptotically stable or unstable. The gist of this classification is that attractors "attract" trajectories near them and repellers "repel" them as $t \to \infty$.

Secondly, a more subtle idea that is introduced is that of **basins of attraction** for attractors. A basin of attraction is basically a set where all initial conditions of the trajectories approach the attractor as $t \to \infty$. In the definition above, this was simply the disk D_δ. Basins of attractions can be thought to be "trapping regions" of trajectories that eventually approach an attractor. In the analysis of trajectories their long term behaviour is called the **dynamics** of the dynamical system. For example, the dynamics of the autonomous ODE we looked at $\dot{y} = (y-1)(y-2)$ are that the fixed point $y = 1$ is an *attractor* and the fixed point $y = 2$ is a repeller.

There are instances where the notion of fixed points can be extended to represent them as sets. In such cases, they are called **invariant sets**. So in the discussion of dynamical systems, it is common to speak of "basins of attractions of invariant sets". So, in a phase plane of trajectories, there may be intervals that have different behaviour depending on what invariant sets are close by. Such intervals are delimited by separatrices where a **separatrix** is a boundary separating two modes of behaviour of a dynamical system.

3.5. Summary

In this chapter we explored concepts dealing with ordinary differential equations (ODEs)and continuous dynamical systems. This background is necessary in this work because most of the analysis will involve autonomous ODEs which are used to construct dynamical systems that are essential in the modelling of the ascent and descent steps of the algorithms this work presents.

4

Background: Non-Linear Local and Global Optimisation Theory

We now explore the parts of optimisation theory used in this work.

4.1. Formulation

We consider the optimisation problem

$$\begin{aligned} \underset{x}{\text{minimise}} \quad & f(x) \\ \text{subject to} \quad & \mathbf{x} \in \Omega. \end{aligned} \tag{4.1}$$

The real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ we wish to minimise is the *objective function*. The vector of independent components $x = [x_1, x_2, ..., x_n]$ that we wish to find is called the *minimiser* of f. The independent components of x are also referred to as *decision variables* and they characterise the best decision when the optimisation problem is viewed as a decision problem. There may be more than one minimiser for a particular problem. Finding any of these usually suffices [2].

This formulation of the problem is termed as *constrained* because we have imposed a restriction on the minimiser $\mathbf{x}$ that it should be in the set Ω. This set Ω is called the *feasible region*. However, if the set $\Omega = \mathbb{R}^n$ then the problem is referred to as an *unconstrained* optimisation problem. We will restrict our discussion of the generalised optimisation problem to the unconstrained case [2]. Using this characterisation of the problem we will define two types to minimisers.

Definition 4.1.1. (Local Minimiser)
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real-valued function defined on some set $\Omega \subset \mathbb{R}^n$. A point $x^* \in \Omega$ is a

local minimiser of f over Ω if there exists $\epsilon > 0$ such that $f(x^*) \leq f(x)$ for all $x \in \Omega \setminus \{x^*\}$ and $\|x - x^*\| < \epsilon$ [2].

Definition 4.1.2. (Global Minimiser)
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real-valued function defined on some set $\Omega \subset \mathbb{R}^n$. A point $x^* \in \Omega$ is a local minimiser of f over Ω if $f(x^*) \leq f(x)$ for all $x \in \Omega \setminus \{x^*\}$ [2] .

If these definitions switch the "$\leq$" with "$<$", then we get a *strict local minimiser* and a *strict global minimiser*, respectively. Pedagogically, the optimisation problem 6.6 we defined is only solved when a global minimiser is found. However, realistically, these are extremely hard to find so in practice finding local minimisers suffices [2].

4.2. First and Second Order Conditions

We will now look at the conditions that a point x^* should satisfy in order to be local minimiser. Since our formulation 6.6 has a feasible set Ω, there is a possibility that x^* might lie on its boundary. For this reason we introduce the idea of feasible directions.

Definition 4.2.1. (Feasible Direction)
A vector $\mathbf{d} \in \mathbb{R}^n$, $\mathbf{d} \neq 0$, is a feasible direction at $x \in \Omega$ if there exists $\alpha_0 > 0$ such that $x + \alpha\mathbf{d} \in \Omega$ for all $\alpha \in [0, \alpha_0]$ [2].

The intuition behind feasible directions is fairly simple. From a point $x \in \Omega$, a feasible direction $\mathbf{d}$ is a direction that one can move and still be in the feasible region Ω. We now state and prove two important theorems.

Theorem 4.2.1. (First-Order Necessary Condition - General Case)
Let the feasible set $\Omega \subset \mathbb{R}^n$ and objective function f be once continuously differentiable function defined on Ω. If x^* is a local minimiser of f over Ω then for all feasible directions $\mathbf{d}$ at x^* we have

$$\mathbf{d}^\top \nabla f(x^*) \geq 0$$

Proof. Let us first parameterise x with α

$$x(\alpha) = x^* + \alpha\mathbf{d} \in \Omega$$

then let's define a composite function with f dependent on α

$$\phi(\alpha) = f(x(\alpha))$$

Using a Taylor series expansion of $\phi(\alpha)$ about $\alpha = 0$ gives

$$\phi(\alpha) = \phi(0) + \phi'(\alpha - 0) + \frac{1}{2}(\alpha - 0)^2\phi''(0) + ...$$

$$\Rightarrow \phi(\alpha) = \phi(0) + \phi'(0) + \frac{1}{2}\alpha^2\phi''(0) + ...$$

Considering a first order approximation gives

$$\Rightarrow \phi(\alpha) - \phi(0) = \phi'(\alpha)$$

But $\phi(\alpha) = f(x(\alpha))$ and $\phi'(\alpha) = \mathbf{d}^\top f(x(\alpha))$

$$\Rightarrow f(x(\alpha)) - f(x(0)) = \mathbf{d}^\top \nabla f(x(0))$$

And, from the definition of $x(\alpha)$, we have $x(0) = x^*$, where x^* is the minimiser, so

$$\Rightarrow f(x(\alpha)) - f(x^*) = \mathbf{d}^\top \nabla f(x(0))$$

To get

$$f(x(\alpha)) - f(x^*) = \mathbf{d}^\top \nabla f(x(\alpha)) \tag{4.2}$$

Since x^* is a minimiser, then

$$f(x^*) \leq f(x(\alpha)) \Rightarrow f(x^*) \leq f(x^* + \alpha\mathbf{d})$$

and thus

$$f(x^* + \alpha\mathbf{d}) - f(x^*) \geq 0$$

using this on equation 4.2 gives

$$\mathbf{d}^\top \nabla f(x(\alpha)) \geq 0$$

as required. Hence proved. □

Theorem 4.2.2. (First-Order Necessary Condition - Interior Case)
Let the feasible set $\Omega \subset \mathbb{R}^n$ and objective function f be once continuously differentiable function defined on Ω. If x^* is a local minimiser of f over $\Omega$and if x^* is an interior point of Ω then

$$\nabla f(x^*) = 0$$

[2]

Proof. Let x^* be an interior point of Ω and be a minimiser of $f(x)$. Since x^* is an interior point, then the whole $\mathbb{R}^n$ is a feasible direction at x^*. Thus, for any $\mathbf{d} \in \mathbb{R}^n$, we have

$$\mathbf{d}^\top \nabla f(x(\alpha)) \geq 0$$

and

$$-\mathbf{d}^\top \nabla f(x(\alpha)) \geq 0$$

which implies

$$\nabla f(x^*) = 0$$

as required. Hence proved. □

We now state and prove two second order necessary conditions for minimisers.

Theorem 4.2.3. (Second-Order Necessary Condition - General Case)
Let $\Omega \subset \mathbb{R}^n$, f be a twice differentiable function defined over Ω, x^* be a minimiser of function f over Ω, and $\mathbf{d}$ be a feasible direction at x^*. If $\nabla f(x^*) = 0$ then

$$\mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d} \geq 0$$

where $\nabla^2 f(x^*)$ is the Hessian [2].

Proof. We prove this by contradiction. Let us suppose that we have a feasible direction $\mathbf{d}$ such that $\mathbf{d}^\top \nabla f(x^*) = 0$ and $\mathbf{d}^\top \nabla f(x^*)\mathbf{d} < 0$. Let us define a composite function where f is defined in terms of α as before

$$\phi(\alpha) = f(x(\alpha)) = f(x^* + \alpha\mathbf{d})$$

Using a second order Taylor series approximation of $\phi(\alpha)$ about $\alpha = 0$ we get

$$\phi(\alpha) = \phi(0) + (\alpha - 0)\phi'(0) + \frac{1}{2}(\alpha - 0)^2\phi''(0)$$

$$\Rightarrow \phi(\alpha) = \phi(0) + \alpha\phi'(0) + \frac{1}{2}\alpha^2\phi''(0)$$

But we supposed that $\mathbf{d}^\top \nabla f(x^*) = \phi'(0) = 0$ so

$$\phi(\alpha) = \phi(0) + \frac{1}{2}\alpha^2\phi''(0)$$

$$\Rightarrow \phi(\alpha) - \phi(0) = \frac{1}{2}\alpha^2\phi''(0)$$

re-writing in terms of f

$$f(x(\alpha)) - f(x(0)) = \frac{1}{2}\alpha^2\mathbf{d}^\top \nabla^2 f(x(0))\mathbf{d}$$

$$\Rightarrow f(x^* + \alpha\mathbf{d}) - f(x^*) = \frac{1}{2}\alpha^2\mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d}$$

but we also supposed that $\mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d} < 0$ so

$$f(x^* + \alpha\mathbf{d}) - f(x^*) < 0$$

which contradicts the assumption that x^* is a minimiser.

Therefore,

$$\mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d} \geq 0$$

as required. Hence proved. □

Theorem 4.2.4. (Second-Order Necessary Condition - Interior Case)
Let $\Omega \subset \mathbb{R}^n$, f be a twice differentiable function defined over Ω, x^* be an interior point of Ω. If x^* is a minimiser of function f over Ω then

$$\nabla f(x^*) = 0$$

and the Hessian $\nabla^2 f(x^*)$ is positive semi-definite, that is, for any feasible direction $\mathbf{d} \in \mathbb{R}^n$ at x^*

$$\mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d} \geq 0$$

[2].

Proof. Let x^* be an interior point of Ω and be a minimiser of $f(x)$.

Since x^* is an interior point, we use theorem 4.2.2 to get

$$\nabla f(x^*) = 0$$

We also know that for the interior case the whole $\mathbb{R}^n$ is a feasible direction at x^*. Using theorem 4.2.3 for any $\mathbf{d} \in \mathbb{R}^n$ we get

$$\mathbf{d}^\top \nabla^2 f(x(\alpha))\mathbf{d} \geq 0$$

as required. Hence proved. □

The theorems we have stated so far are only necessary and not sufficient conditions for minimisers [2]. Therefore, we need a stronger characterisation of a minimiser that would serve as a sufficient condition. We now define the idea of the Rayleigh's Inequalities and then state and prove a theorem that does this.

Definition 4.2.2. (Rayleigh's Inequalities)
If $n \times n$ matrix $\mathbf{P}$ is a real symmetric positive definite, then

$$\lambda_{min}(\mathbf{P})\|\mathbf{x}\|^2 \leq \mathbf{x}^\top \mathbf{P}\mathbf{x} \leq \lambda_{max}(\mathbf{P})\|\mathbf{x}\|^2$$

where $\lambda_{min}(\mathbf{P})$ and $\lambda_{max}(\mathbf{P})$ are the smallest and largest eigenvalues of $\mathbf{P}$, respectively [2].

Theorem 4.2.5. (Second-Order Sufficient Condition - Interior Case)
Let f be a real-valued function defined on a region where x^* is an interior point. Suppose that

1. $\nabla f(x^*) = 0$
2. $\nabla^2 f(x^*) > 0$

then x^* is a strict local minimiser of f [2].

Proof. By the assumption that $\nabla^2 f(x^*) > 0$ we know that the Hessian is positive definite.
Using the Rayleigh's Inequalities, we can deduce that for any feasible direction $\mathbf{d} \in \mathbb{R}^n$, with restriction $\mathbf{d} \neq 0$, we can write

$$0 < \lambda_{min}(\nabla^2 f(x^*))\|\mathbf{d}\|^2 \leq \mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d}$$

Now, using a Taylor series second order expansion of composite function $f(x^* + \alpha\mathbf{d})$ about $\alpha = 0$ gives

$$f(x^* + \alpha\mathbf{d}) = f(x^*) + \mathbf{d}^\top \nabla f(x^*) + \frac{1}{2}\mathbf{d}^\top f(x^*)\mathbf{d}$$

but the first assumption says $\nabla f(x^*) = 0$ so

$$f(x^* + \alpha\mathbf{d}) = f(x^*) + \frac{1}{2}\mathbf{d}^\top f(x^*)\mathbf{d}$$

And

$$\Rightarrow f(x^* + \alpha\mathbf{d}) - f(x^*) = \mathbf{d}^\top f(x^*)\mathbf{d}$$

Using the our characterisation of the Rayleigh's inequality $0 < \lambda_{min}(\nabla^2 f(x^*))\|\mathbf{d}\|^2 \leq \mathbf{d}^\top \nabla^2 f(x^*)\mathbf{d}$ we get the relationship

$$f(x^* + \alpha\mathbf{d}) - f(x^*) = \frac{1}{2}\mathbf{d}^\top f(x^*)\mathbf{d} \geq \frac{1}{2}\lambda_{min}(\nabla^2 f(x^*))\|\mathbf{d}\|^2$$

$$\Rightarrow f(x^* + \alpha\mathbf{d}) - f(x^*) \geq \frac{1}{2}\lambda_{min}(\nabla^2 f(x^*))\|\mathbf{d}\|^2$$

So, for all $\mathbf{d}$ where $\|\mathbf{d}\|$ is sufficiently small, then

$$f(x^* + \alpha\mathbf{d}) > f(x^*)$$

as required, which completes the proof [2]. □

4.3. Gradient Descent Methods

In practice, optimisation problems are solved by iterative algorithms. Given a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$, an iterative algorithm that seeks to minimise this function is of the form

$$x_{k+1} = x_k + \alpha_k \mathbf{d}_k \tag{4.3}$$

The vector $\mathbf{d}_k$ is called the *search direction* and α_k is called *step-size* which satisfy $\alpha_k \geq 0$. The intuition behind the algorithm depicted by iterative equation 4.3 is that it starts an an initial point, say x_0, and calculates the direction $\mathbf{d}_0$ in which to move in order to minimise f and the associated step-size α_0 and uses these to move to the next point x_1, and so on [2]. The general algorithm is:

Algorithm 4.1 General Gradient Descent Algorithm

1. Given point x_k
2. Derive search direction $\mathbf{d} \in \mathbb{R}^n$
3. Decide on step-size α_k
4. Move to next point
$$x_{k+1} = x_k + \alpha_k \mathbf{d}_k$$

A common choice for $\mathbf{d}_k$ is usually characterised by the gradient of the function f at x_k. The most obvious choice for $\mathbf{d}_k$ is one such that the function decreases ($\mathbf{d}_k^\top \nabla f(x_k) < 0$). After some thought, it's fairly obvious that $-\nabla f(x_k)$ is the direction of maximum decrease of the function. Other variants of characterising $\mathbf{d}_k$ exist such as those that also consider second order derivatives [2].

The choice for α_k is one that aims to achieve the condition:

$$f(x_{k+1}) < f(x_k)$$

This makes sense since it is desirable that the algorithm seek out points that minimise the function. The choice of α_k can be thus be imagined to one that seeks to ensure that, given we have a direction of descent $\mathbf{d}_k$, α_k is the step-size ensuring function f is minimum in that direction [2]. Formally, we seek to minimise

$$\phi_k(\alpha) = f(x + \alpha\mathbf{d}) \tag{4.4}$$

This function is in terms of α and this therefore one-dimensional. Thus α_k is

$$\alpha_k \in \text{argmin} f(x_k + \alpha\mathbf{d}_k) \tag{4.5}$$

This one-dimensional optimisation is called a *line search*. A type of gradient descent algorithm that uses the step-size characterisation above and $\mathbf{d}_k = -\nabla f(x_k)$ is called the *steepest descent algorithm*, which can be described as:

Algorithm 4.2 Steepest Descent Algorithm

1. Given point x_k
2. Compute gradient at x_k and derive search direction
$$\mathbf{d}_k = -\nabla f(x_k)$$
3. Decide on step-size α_k, $(\alpha_k \geq 0)$
$$\alpha_k \in \text{argmin} f(x_k + \alpha \mathbf{d}_k)$$
4. Move to next point
$$x_{k+1} = x_k + \alpha_k \mathbf{d}_k$$

The consideration above presumes we are able to find the *exact* solution to the line search problem. Therefore, the line search phase is usually called an *exact line search*. In practice, however, many things could go wrong. Firstly, solving the problem may be computationally demanding and may take longer than we desire. Secondly, a minimiser to the problem may not exist. It is for these reasons that in practice *inexact line searches* are use that approximate the solution to the optimisation problem by using pre-defined termination conditions. The goal is to ensure that the there is a sufficient decrease in f from one iteration to the next despite having an approximate value of a_k. This is done by ensuring that the value of a_k is not too large or not too small [2].

The set of inequalities that used to perform this *inexact line search* are called *Wolfe conditions*.

Definition 4.3.1. (Wolfe Conditions)
Let $\phi_k(\alpha) = f(x_k + \alpha \mathbf{d}_k)$ be an objective function. A step-size a_k is said to satisfy the **Wolfe Conditions** *if the following hold:*

1. $f(x_k + \alpha_k \mathbf{d}_k) \leq f(x_k) + \epsilon_1 \alpha_k \mathbf{d}_k^\top \nabla f(x_k)$
2. $\mathbf{d}_k^\top \nabla f(x_k + \alpha_k \mathbf{d}_k) \geq \epsilon_2 \mathbf{d}_k^\top \nabla f(x_k)$

where $0 < \epsilon_1 < \epsilon_2 < 1$ [2].

The first condition ensures that α_k is not too large and the second one ensure that it is not too small. The first condition is commonly referred to as *Armijo's Rule*. A simple and practical *inexact line search* algorithm that makes use of the *Armijo's Rule* is called *Armijo's Backtracking* [2]. The Algorithm is shown in algorithm

Algorithm 4.3 Armijo's Backtracking Algorithm

Initialise $k = 0, \tau \in (0, 1)$;
Choose initial $\alpha_0 > 0$;
while (α_k NOT satisfy Amijo's Rule) {

1. $\alpha_{k+1} = \tau\alpha_k$;
2. $k = k + 1$;

}

The intuition behind this algorithm is that it starts at an initial α_k and checks if it satisfies *Armijo's Rule*. If it does, this initial α_k is used as the step-size. Otherwise, we iteratively decrease it by a factor $\tau \in (0, 1)$ and re-check if the new value satisfies *Armijo's Rule*. This algorithm is said to "backtrack" because after j iterations, we have $\alpha_k = \tau^j\alpha_0$; so, indadvertedly, the algorithm *backtracks* from the initial value α_0 until the *Armijo's Rule* is satisfied [2].

4.4. Penalty Methods

Let us consider a constrained optimisation problem

$$\min \{f(x) : x \in \Omega\}.$$

We will now discuss methods that solve the problem above using techniques from unconstrained optimisation. Essentially, we want to approximate the problem above as the unconstrained optimisation problem

$$\min \{f(x) + \gamma P(x)\},$$

where $\gamma \in \mathbb{R}$ is a positive constant and $P : \mathbb{R}^n \to \mathbb{R}$ is a given function. We solve this problem and use its solution to approximate the solution to the original problem. The constant γ is called the *penalty parameter* and the function P the *penalty function* [2].

Definition 4.4.1. (Penalty Function)
A function $P : \mathbb{R}^n \to \mathbb{R}$ is called a Penalty Function of a constrained optimisation problem if it satisfies the following conditions:

1. *P is continuous;*
2. *$P(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$; and*
3. *$P(\mathbf{x}) = 0$ if and only if $\mathbf{x}$ is feasible, that is $\mathbf{x} \in \Omega$.*

For the unconstrained problem to be a good approximation of the original constrained problem then P should be specified appropriately. The role of the penalty function is to *penalise* points outside the feasible set [2].

4.5. Global Optimisation and the Global Descent Functions

As already stated in the introductory chapter, researchers have developed sound theories and efficient numerical techniques for solving convex optimisation problems. However, most optimisation

problems are non-convex in nature and the existence of multiple locally optimal solutions makes global optimisation a great challenge. From the uses of optimisation available from the advances in the theory made so far, it is evident that attaining the goal of global optimisation will greatly improve the quality of our lives [5].

In this section we will look at general continuous global optimisation problem of the form

$$\min \{f(x) : x \in \mathbb{X}\}. \tag{4.6}$$

If $\mathbb{X} = \mathbb{R}^n$, then the problem above is an unconstrained global optimisation problem. The goal here is to find the global minimiser of $f(x)$ over $x \in \mathbb{X}$. Generic global optimisation algorithms have two separate phases. The first is the *global phase* which involves an exhaustive exploration of search space. The method used to do this can either be deterministic or stochastic. At each iteration of this global phase, a local optimisation procedure is called to identify a locally optimal point. This is known as the *local phase* and is usually deterministic. Most global optimisation methods use local optimisation methods as a tool. Most calculations take place during the local phase; therefore, a robust and reliable local optimisation algorithm with fast convergence is important [4].

However, in this section, we are interested in a particular approach to global optimisation that is usually referred to as the function modification approach (see, e.g., [5, 25, 26, 27]). This way of approaching global optimisation is one of the most promising paradigms for solving general global optimisation problems. This method is able to facilitate the transcending of a current local minimiser at each iteration to find a better one by adopting some appropriate modifications of the objective function f [5]. The general solution procedure of algorithms following this a paradigm is composed of the two-phased cycle shown in algorithm 4.4.

Algorithm 4.4 Function Modification Global Optimisation Algorithm

Phase 0 (initialisation): Select a feasible initial point x_{ini} and set the iteration count at $N_{iter} := -1$. Use x_{ini} as the initial point of the local search in Phase 1.

Phase 1 (local search): Start from a given feasible point and use any suitable local minimisation method to search for a local minimiser x of f over $\mathbb{X}$. Update the iteration count $N_{iter} := N_{iter} + 1$. Perform the global search in Phase 2.

Phase 2 (global search): Construct an auxiliary function (a proper modification of f) such that its minima are not worse than (or preferably better than) x and use it to search for a feasible point x such that $f(x) \leq f(x^*)$ (or preferably $f(x) < f(x)$). If such a transitional point is obtained, then use x as the initial point of the local search in Phase 1. If no such transitional point is obtained, stop the iteration process and return x as a (putative) global minimiser of problem

$$\min \{f(x) : x \in \mathbb{X}\}$$

In order to use the function modification approach we are going to make the following assumptions about the problem 4.6:

Assumption 1. $\mathbb{X} \subset \mathbb{R}^n$ is a *compact* and *connected set* with its interior, int($\mathbb{X}$), *nonempty* [5].

Assumption 2. $f : \mathbb{X} \mapsto \mathbb{R}$ is a continuously differentiable function that satisfies the following Lipschitz condition for any pair of x_1 and x_2 in $\mathbb{X}$:

$$|f(x_1) - f(x_2)| \leq L||x_1 - x_2||,$$

where $0 < L < \infty$ is the Lipschitz constant and $||\cdot||$ is the usual Euclidean norm [5].

Assumption 3. The set of the values of all local minima

$$\{f^* = f(x^*) : x^* \text{ is a local minimiser of } f \text{ over } \mathbb{X}\}$$

is finite [5].

Assumption 4. All global minimisers of f are contained in int($\mathbb{X}$) [5].

Let us discuss, albeit briefly, the implications of the above stated assumptions. To begin with, assumption 1 implies that there exists a positive constant K such that

$$0 < \max ||x_1 - x_2|| \leq K < \infty,$$

with $x_1, x_2 \in \mathbb{X}$ [5].

Assumption 2, on the other hand, implies that there is a positive constant M such that the norm of the gradient of the function f is bounded by M, that is,

$$0 < \max ||\nabla f(x)|| \leq M < \infty,$$

with $x \in \mathbb{X}$ [5].

Assumption 3 does not exclude cases in which the number of local minimisers of f over $\mathbb{X}$ is infinite. Assumption 4 allows for non-global minimisers to be located on the boundary of $\mathbb{X}$. It is also worth noting that when the function f is coercive, that is, $f(x) \to \infty$ as $||x|| \to \infty$, there always exists a box containing all global minimisers of f. Therefore, the unconstrained global optimisation problem we want to solve, $\min f(x) : x \in \mathbb{R}^n$, can be reduced into an equivalent problem formulation in equation 4.6 [5].

One efficient approach to global optimisation that uses the function modification methodology is that of *filled functions* introduced by Ge in [25], which we will now define.

Definition 4.5.1. (Filled Function)
Let x^* be an isolated local minimiser of f over $\mathbb{X}$. A function F_{x^*} is said to be a **filled function** *of f at x^* if it satisfies the following conditions:*

(P1) x^* is a maximiser of F_{x^*} and the whole basin B^* of f at x^* becomes part of a hill of F_{x^*};

(P2) F_{x^*} *has no minimisers or saddle points in any basin of* f *higher than* B^**;*

(P3) if f *has a basin* B^{**} *at* x^{**} *that is lower than* B^* *, then there is a point* $x' \in B^{**}$ *that minimises* F_{x^*} *on the line through* x^* *and* x'' *for every* x'' *in some neighbourhoods of* x^{**}*.*

[5, 25]

Let's try to elucidate what this means from a topological perspective. Firstly, the intuition behind the name "filled function" is described by property (P1) and is as follows: an objective function f with an isolated local minimiser x^* will have this minimiser located in a basin or "trough" or "hole" on that function which becomes a hill on the corresponding filled function F_{x^*} at x^*; therefore, these hills on the filled function can "fill" the basins on the analogous objective function.

Property (P2) just says what was described that the local minima on the original objective function f inscribed on the corresponding filled function will include the minimiser x^* and better minimisers that yield values of f lower that $f(x^*)$. In other words, the filled function ignores or the minimisers that are worse that x^*. Some thinking will make one realise that this is actually a very good property to have especially when dealing with global optimisation because we are always looking for better minimisers. This prevents us from wasting time on worse minimisers that are of no consequence in the global optimisation scheme.

The last property (P3) is the most interesting. It says that if the objective function f has another local minimiser x^{**} with an associated basin B^{**} that is better than our minimiser x^* and its basin B^*, then there exists an intermediate point x' with $x' \in B^{**}$ that minimises F_{x^*} along a straight line passing through x^* and all the points x'' in the neighbourhood of the better minimiser x^{**}. In other words, we can find a point close to the better minimiser on the original function f by minimising the filled function F_{x^*}; we can do this by initialising a local optimisation algorithm from a point near x^*.

As one can imagine, a filled function has the beauty of being applicable in many different areas and contexts. Many researchers have incorporated filled functions into their algorithms (see, e.g., [28, 29, 30, 31]) and others have applied it to practical problems (see, e.g., [32, 33, 34, 35, 36]).

However, filled functions have their problems. Firstly, as pointed out by Ge and Qin [37], filled functions suffer from a problem in numerical implementation in that changes in both the filled functions and the gradient are indistinguishable when $||x - x^*||$ is large. In addition to this numerical problem, one major weakness associated with the underpinnings of the filled function methods is that the aforementioned properties (P1) – (P3) do not guarantee a minimiser of the domain of the minimiser $\mathbb{R}^n$ for the filled function [5]. There are even cases where a filled function has no local minimiser in its domain $\mathbb{R}^n$ even though the current minimiser of f is not the global minimiser [5].

To deal with the shortcomings of the filled functions are **global descent functions** that enhance the concept of filled functions to retain all their attractive properties and also guarantee that if there is a better minimiser than the current minimiser x^* then we should be able to find the minimiser of x' with $f(x') < f(x^*)$ in the neighbourhood of the better minimiser x^{**}. Let's define the global descent functions.

Definition 4.5.2. (Global Descent Function)
Let x^* be a known local minimiser of an objective function f where x^* is an interior point of the feasible set, $x^* \in int(\mathbb{X})$. We also let

$$\hat{X}(x^*) = \{x^* \in \ int(\mathbb{X}) : x \neq x^*, f(x) \geq f(x^*)\}$$

A function $G_{x^*} : \mathbb{X} \mapsto \mathbb{R}$ is the **Global Descent Function** *of f at x^* if it satisfies the following conditions:*

(C1) x^* is strict local maximiser of G_{x^*} over $\mathbb{X}$;

(C2) G_{x^*} has no stationary points in the set $\hat{X}(x^*)$;

(C3) If the function f has a minimiser $x^{**} \in \ int(\mathbb{X})$ with $f(x^{**}) < f(x*)$ then G_{x^*} has a minimiser x' such that $x' \in N_\epsilon(x^{**}) \subset \mathbb{X}$ and $f(x) < f(x^*)$ for all $x \in N_\epsilon(x^{**})$, where $N_\epsilon(x^{**})$ is the ϵ-neighbourhood of x^{**}.

For the purpose of the work in this dissertation, we will make use of the family of two-parameter global descent functions defined by Ng, Li and Zhang in [5] which are as follows:

Definition 4.5.3. (Family of Two-Parameter Global Descent Functions)

$$G_{\mu,\rho,x^*} = A_\mu(f(x) - f(x^*)) - \rho||x - x^*||, \tag{4.7a}$$

$$A_\mu(y) = y \cdot V_\mu(y), \tag{4.7b}$$

$$V_\mu(y) : \mathbb{R} \mapsto \mathbb{R}, \tag{4.7c}$$

where $\rho > 0$, $0 < \mu < 1$ are problem dependent parameters and $V_\mu(y) : \mathbb{R} \mapsto \mathbb{R}$ is continuously differentiable function that satisfies the following conditions:

(V1) $V_\mu(-\tau) = 1$, $V_\mu(0) = \mu$, and $V_\mu(y) \geq c\mu$ for all y;

(V2) $V'_\mu(y) < 0$ for all $y < 0$ and $-c'\mu \leq V'_\mu(y) \leq 0$ for all $y > 0$,

where $V'_\mu(y)$ is the derivative of V_μ, $\tau > 0$ is a sufficiently small number, $0 < c \leq 1$ is a constant, and $c' \geq 0$ is a constant or a function of μ with $\lim_{\mu \to 0} c\mu = 0$.

For complete proofs of the properties given in the definition above, the reader can refer to [5].

Ng, Li and Zhang went on to describe a solution algorithm shown in algorithm 4.5. It is obvious from the structure of the algorithm that it uses the function modification framework specified in algorithm 4.4.

Phases 0 (initialisation) and phase 1 (local search) are obvious and quite intuitive as they mirror the description given in 4.4; however, the global search phase in the algorithm is the most interesting one so let's take some time to understand what is going on.

In phase 2(a) a set of m initial points is generated to minimise G_{μ,ρ,x^*}. Any method can be used to generate these points as long as they are distributed efficiently enough to yield a better minimiser. One way to do so is to distribute the m start points symmetrically around the current

local minimiser x^*. Another approach to take may be to generate m random points uniformly on $\mathbb{X}$ [5].

Phase 2(c) covers the case where the algorithm's current point x_{cur} is in a lower basin than the current minimiser x^* by satisfying $f(x_{cur}) < f(x^*)$. Because of this, we can break away from the global phase and start any local minimisation algorithm from x_{cur} and find a better minimiser than x_{cur}.

Phase 2(d) is slightly more involving. What the algorithm is doing in this step is trying to modify the constant μ until the conditions

$$G_{\mu,\rho,x^*}(x_{cur}) \not\to \text{ and } (x_{cur} - x^*)^\top \nabla G_{\mu,\rho,x^*}(x_{cur}) < 0$$

hold simultaneously. Why is this important? We want to modify G_{μ,ρ,x^*} in order to guarantee that we are able to find a minimiser while searching from x_{cur}. One of the results proved during the construction of of the global descent function in [5] is that if $f(x_{cur}) \geq f(x^*)$ and μ is sufficiently small, then x_{cur} cannot be a stationary point of G_{μ,ρ,x^*}. Notice that $(x_{cur} - x^*)$ is a descent direction of G_{μ,ρ,x^*} at x_{cur} because we already know that $f(x_{cur}) \geq f(x^*)$. Consequently, we modify μ so that the above conditions hold.

The advantage that the global descent function has over the prototypical filled function is that it ensures that G_{μ,ρ,x^*} has a minimiser in the domain int($\mathbb{X}$). Therefore, we can use any paradigm to get the descent direction D of G_{μ,ρ,x^*} at x_{cur} in phase 2(e) to approach the minimiser of G_{μ,ρ,x^*}. Examples of the direction paradigms that can be employed is the steepest descent direction or the Newton's direction, among others [5].

However, this search on G_{μ,ρ,x^*} should be done with caution. A local search that is too aggressive with a large step-size may lead to a solution trajectory outside the the basin at the minimiser. Therefore, limiting the step-size in the line search scheme is a key to success in the global search phase [5].

The constant ρ used for the global descent function G_{μ,ρ,x^*} should be selected to be small enough because there could be no minimiser for G_{μ,ρ,x^*}, even when x_{cur} is in a basin of f lower than the basin at the current minimiser x^*. Thus, the value of ρ is reduced to a preselected fraction of itself, $\hat{\rho}$, in phase 2(g) of the solution process if no better solution is found while minimising G_{μ,ρ,x^*}. If the value of ρ reached its predefined lower bound ρ_L and no better minimiser is found , then the current minimiser is taken to be (putative) global minimiser of f [5].

During any implementation of this algorithm the constants used may not be easy to estimate and therefore need to guessed and fine-tuned.

Algorithm 4.5 Global Descent Method for Global Optimisation

Phase 0: Initialisation

(a) Choose a function v_μ satisfying conditions (V1) and (V2).

(b) Choose $\rho_{ini} > 0$ (the initial value of ρ), $0 < \mu_{ini} < 1$ (the initial value of μ), $\mu_L > 0$ (the lower bound of ρ), $0 < \hat{\rho} < 1$(a fraction for the reduction of ρ), $0 < \hat{\rho} < 1$ (a fraction for the reduction of μ), $\epsilon > 0$ (the radius of a small neighbourhood of x^*), $\kappa > 0$ (a small tolerance), and $\lambda_U > 0$ (the maximum step-size for a line search).

(c) Set $\rho := \rho_{ini}$ and $\mu := \mu_{ini}$.

(d) Choose/generate an initial point $x_{ini} \in \mathbb{X}$ for problem

$$\min \{f(x) : x \in \mathbb{X}\}$$

and set the current iterative point $x_{cur} := x_{ini}$ and the iteration count $N_{iter} := -1$.

(e) Perform the local search in Phase 1.

Phase 1: Local Search

(a) Starting from x_{cur}, use any local minimisation method to search for a local minimiser x^{**} of f over $\mathbb{X}$. Update the current local minimiser $x^* := x^{**}$ and the iteration count $N_{iter} := N_{iter}+1$.

(b) Perform the global search in Phase 2.

Phase 2: Global Search

(a) Generate a set of m initial points: $\{x_{(i)}^{ini} \in \mathbb{X}\backslash N_\epsilon(x^*) : i = 1, 2, ..., m\}$. Set $i := 1$.

(b) Set the current iterative point $x_{cur} := x_{(i)}^{ini}$

(c) If $f(x_{cur}) < f(x^*)$, then perform the local search in Phase 1.

(d) If

$$||\nabla G_{\mu,\rho,x^*}(x_{cur})|| < \kappa \text{ and } (x_{cur} - x^*)^\top \nabla G_{\mu,\rho,x^*}(x_{cur}) \geq 0,$$

then choose a positive integer l such that $\mu_l := \mu^l \mu$ and

$$||\nabla G_{\mu,\rho,x^*}(x_{cur})|| \geq \kappa \text{ or } (x_{cur} - x^*)^\top \nabla G_{\mu,\rho,x^*}(x_{cur}) < 0.$$

Update $\mu := \mu_l$.

(e) Choose a descent direction D of G_{μ,ρ,x^*} at x_{cur} Find a new x along D by a line search method such that G_{μ,ρ,x^*} can reduce to a certain extent and the step-size of the line search $\lambda \leq \lambda_U$; then set $x_{cur} := x$ and go to Phase 2(c). However, if x attains the boundary of $\mathbb{X}$ during minimisation, then go to Phase 2(f);

(f) Set $i := i + 1$. If $i \leq m$, then go to Phase 2(b);

(g) Set $\rho := \hat{\rho}\rho$ and reset $\mu := \mu_{ini}$. If $\rho \geq \rho_l$, then go to Phase 2(a). Otherwise, the algorithm is incapable of finding a minimiser of f better than the current local minimiser x^* starting from the initial points $x_{(i)} : i = 1, 2, ..., m$. The algorithm stops, and x^* is taken as a (putative) global minimiser.

4.6. Summary

This chapter is an overview of the optimisation theory used in this work. It covers the concepts in non-linear local and global optimisation theory, including global descent functions.

5

The Gentlest Ascent Dynamics (GAD)

We are interested in the opposite dynamics: the dynamics of escaping a basin of attraction.
– Weinan E and Xiang Zhou, 2011 [7]

This chapter gives an overview of the work by Weinan E and Zhou in [7] on the Gentlest Ascent Dynamics (GAD).

5.1. Formulation

The work in this chapter presents dynamical systems that escape from basins of attraction of attractors. The stable fixed points of these dynamics are index-1 saddle points.

To get a picture of the implication of this result, let us consider a function V defined on $\mathbb{R}^n$. Let x denote the independent variable that we will parameterise with time t to get $x(t)$. If we are interested in the minima of this function $V(x(t))$, we can write its steepest descent dynamics as

$$\frac{dx(t)}{dt} = -\nabla V(x(t)) \tag{5.1}$$

or more commonly as

$$\dot{x} = -\nabla V(x). \tag{5.2}$$

After some thought, it is easy to realise that if some $x(t)$ is the solution of the autonomous differential equation (5.2), then the original function $V(x(t))$ is a decreasing function of t. This makes sense since we formulated equation (5.2) as "Steepest Descent Dynamics" by seeking out the direction of *steepest decrease* with respect to t. Therefore, if this solution exists, then $V(x(t))$ is decreasing in t.

We can further observe that the stable fixed points of the dynamics presented by (5.2) are the local minima of V. This is quite intuitive since we know that the fixed points of (5.2) correspond

to the stationary points of V; since the dynamics of (5.2) seek out the minima, its fixed points will correspond to the minima of V. Continuing with this idea, we know that each of these fixed points or minima have basins of attractions associated with them where all initial conditions from which the steepest descent dynamics described by (5.2) converges to that local minimum as time goes to infinity. These can be viewed as *trapping regions* for the particular minima. Thus, each basin of attraction can be thought to define its own behaviour of the differential equation (5.2) in its region. From dynamical systems theory as outlined in section 3.4, we remember that a *separatrix* is a boundary separating two modes of behaviour in a differential equation. Thus, these basins of attractions are separated from each other by separatrices, on which the dynamics converge to saddle points [7].

However, the goal in this exposition is to find the opposite dynamics — dynamics that *escape* these basins of attraction. One may suggest we reverse the sign for (5.2) but that would convert the dynamics (5.2) into the *steepest ascent dynamics* and will find the local maxima instead. This is not what we want. What we want is the *gentlest* way in which the dynamics can escape or "climb out" of a basin of attraction. Since we know that basins of attractions have separatrices on which the dynamics converge to saddle points, all we need is to find dynamics that converge to the index-1 saddle points of V.

Weinan E and Zhou found that the following dynamics serve this purpose:

$$\dot{x} = -\nabla V(x) + 2\frac{\langle \nabla V, v\rangle}{\langle v, v\rangle} v \tag{5.3a}$$

$$\dot{v} = -\nabla^2 V(x) v + \frac{\langle v, \nabla^2 V v\rangle}{\langle v, v\rangle} v. \tag{5.3b}$$

They showed that the stable fixed points of the dynamics of equation (5.3a) are precisely the index-1 saddle points of the function V and that the equation (5.3b) showed the unstable directions at the saddle points.

The intuition behind the idea is quite straightforward. The equation (5.3b) attempts to find the unstable directions of function V's index-1 saddle points by finding the direction that corresponds to the smallest eigenvalue of the Hessian $\nabla^2 V$. The first equation (5.3a) is the normal steepest descent dynamics plus a "noise term" that makes the *unstable* directions derived in the second equation (5.3b) the *ascent* direction.

In the formulation of the differential equations (5.3a) and (5.3b), we started out with a dynamical system (5.2) that involved the "gradient" of function V. These are called *gradient systems*. Weinan E and Zhou showed that this idea can be extended to more general *non-gradient systems* or dynamical systems of the form

$$\dot{x} = F(x). \tag{5.4}$$

Therefore, we can talk about about the stable fixed points of the dynamics of (5.4) and how to escape their basins of attraction as we did before. More specifically, we can talk about the finding the index-1 saddle points of $F(x)$. However, there is no guarantee that, under the influence of small "noise", the escape path will be via saddle points [7, 8].

Thus, for non-gradient systems of the form of (5.4), the differential equations (5.3a) and (5.3b) have to be modified to

$$\dot{x} = F(x) - 2\frac{\langle F(x), w\rangle}{\langle w, v\rangle}v, \tag{5.5a}$$

$$\dot{v} = (\nabla F(x))v - \alpha(v)v, \tag{5.5b}$$

$$\dot{w} = (\nabla F(x))^{\top}w - \beta(v, w)w. \tag{5.5c}$$

The two direction vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ are required in order to follow both the right and left eigenvectors of the Jacobian $\nabla F(\boldsymbol{x})$. Given the matrix $\nabla F(\boldsymbol{x})$, the scalar valued functions $\alpha(\boldsymbol{v})$ and $\beta(\boldsymbol{v}, \boldsymbol{w})$ are defined by

$$\alpha(v) = \langle v, (\nabla F(x))v\rangle, \tag{5.6a}$$

$$\beta(v, w) = 2\langle w, (\nabla F(x))v\rangle - \alpha(v). \tag{5.6b}$$

In the original paper, Weinan E and Zhou denoted the normalisation $\langle v, v\rangle = 1$ and $\langle w, v\rangle = 1$. They showed at as long as this held initially it would be preserved. Therefore, if one lets $v = w$, then the equations (5.5a)–(5.5c) can be reduced to equations (5.3a)–(5.3b).

In order to illustrate the insight behind the idea of GAD, let us consider dynamics represented by force $\mathbf{F}$ at an arbitrary point x. The force is simply what causes the dynamics, and in this case can be imagined to be $-\nabla V(x)$. Let v_1 and v_2 be the unstable and stable right eigenvectors, respectively; let w_1 and w_2 represent the corresponding left eigenvectors. It is worth noting that with this description, unstable left eigenvector w_1 is orthogonal to the stable right eigenvector v_2 and so are the other to, that is,

$$w_1 \perp v_2$$

$$w_2 \perp v_1$$

Therefore $\mathbf{F}$ is decomposable into

$$\mathbf{F} = \mathbf{F}_1 + \mathbf{F}_1 = c_1v_1 + c_2v_2,$$

where the coefficient c_1 is

$$c_1 = \frac{\langle \mathbf{F}, w_1\rangle}{\langle v_1, w_1\rangle}.$$

We know that the GAD represents the opposite of the unstable directions of $\mathbf{F}$ escaping the basin of attraction. So, if we denote GAD by $\bar{\mathbf{F}}$, then

$$\bar{\mathbf{F}} := -\mathbf{F}_1 + \mathbf{F}_2$$

$$\Rightarrow \bar{\mathbf{F}} = -\mathbf{F}_1 + (\mathbf{F} - \mathbf{F}_1)$$

$$\Rightarrow \bar{\mathbf{F}} = \mathbf{F} - 2\mathbf{F}_1$$

$$\Rightarrow \bar{\mathbf{F}} = \mathbf{F}_1 - 2c_1v_1$$

$$\bar{\mathbf{F}} = \mathbf{F}_1 - 2c_1v_1$$

which matches 5.5a. An illustration is on figure 5.1.

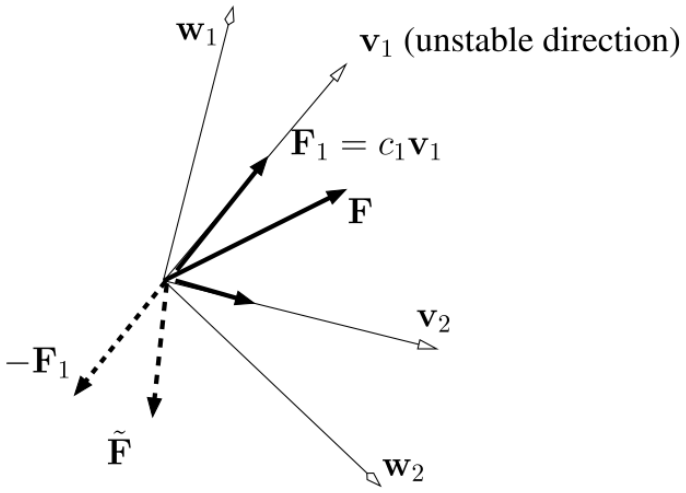


Figure 5.1.: *Illustration of GAD from Weinan E and Zhou paper [7]*

5.2. Analysis

In this section we will state one important proposition and one theorem from the Weinan E and Zhou paper.

Definition 5.2.1. (Vector Field)
Let $I \subset \mathbb{R}^n$, a **vector field** *V is a vector-valued function $V : I \to \mathbb{R}^n$.*

In this section we will consider a vector field $\boldsymbol{F}$ defined in $\mathbb{R}^n$ that we will assume is three times continuously differentiable.

Proposition 5.2.1. (Fixed Point of GAD)
If $(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{w})$ is a fixed point of the GAD (5.5a) – (5.5c) and $\boldsymbol{v}$, $\boldsymbol{w}$ are normalised such that $\boldsymbol{v}^\top \boldsymbol{v} = \boldsymbol{v}^\top \boldsymbol{w} = 1$, then $\boldsymbol{v}$ and $\boldsymbol{w}$ are the right and left eigenvectors , respectively, of $\nabla F(\boldsymbol{x})$ corresponding to one eigenvalue λ^*, that is,

$$(\nabla F(\boldsymbol{x}))\boldsymbol{v} = \lambda^* \boldsymbol{v},$$

$$(\nabla F(\boldsymbol{x}))^\top \boldsymbol{w} = \lambda^* \boldsymbol{w},$$

and $\boldsymbol{x}$ is a fixed point of the original dynamics system, that is, $F(\boldsymbol{x}) = 0$.

Proof. Let $(x^*, v^*, w*)$ be a fixed point.
We have the GAD

$$\dot{x} = F(x) - 2\frac{\langle F(x), w\rangle}{\langle w, v\rangle} v,$$

$$\dot{v} = (\nabla F(x))v - \alpha(v)v,$$

$$\dot{w} = (\nabla F(x))^\top w - \beta(v, w)w,$$

with scaler valued functions

$$\alpha(v) = \langle v, (\nabla F(x))v\rangle,$$

$$\beta(v, w) = 2\langle w, (\nabla F(x))v\rangle - \alpha(v).$$

At fixed point $\dot{v} = \dot{w} = 0$, so we get

$$(\nabla F(x^*))v^* = \alpha(v^*)$$

$$(\nabla F(x^*))^\top w^* = \beta(v^*, w^*)w^*$$

Rewriting the definition of β and using result above and the normalisation condition gives

$$\alpha(v^*) = \nabla F(x^*),$$

$$\beta(v^*, w^*) = 2\langle w^*, (\nabla F(x^*))v^*\rangle - \alpha(v^*) = 2w^{*\top}(\nabla F(x^*))v^* - \alpha(v^*) = 2w^{*\top}(\alpha(v^*))v^* - \alpha(v^*)$$

$$\Rightarrow \beta(v^*, w^*) = 2w^{*\top}(\alpha(v^*))v^* - \alpha(v^*) = 2\alpha(v^*) - \alpha(v^*) = \alpha(v^*)$$

$$\Rightarrow \beta(v^*, w^*) = \alpha(v^*)$$

This means that v^* and w^* share the same eigenvalue which we will denote as λ^*. So

$$\lambda^* = \alpha(v^*) = \beta(v^*, w^*)$$

From the fixed point condition, $\dot{x} = 0$ and normalisation condition we obtain

$$F(x^*) - 2w^{*\top}F(x^*)v^* = 0$$

$$\Rightarrow F(x^*) - 2w^{*\top}F(x^*)v^* = 0$$

getting inner product of the equation above with by w^*

$$w^{*\top}F(x^*) - 2w^{*\top}F(x^*)w^{*\top}v^* = 0$$

$$\Rightarrow w^{*\top}F(x^*) - 2w^{*\top}F(x^*) = 0$$

$$\Rightarrow w^{*\top}F(x^*) = 0$$

and, consequently,

$$\Rightarrow F(x^*) = 0$$

Therefore, from the condition $\dot{x} = F(x^*) - 2w^{*\top}F(x^*)v^* = 0$, we obtained $F(x^*) = 0$, as required. □

Theorem 5.2.1. (Stable Fixed Point of GAD)
Let x_s be a fixed point of the original dynamical system $\dot{x} = F(x)$. If the Jacobian matrix $\mathbb{J}(x_s) = \nabla F(x_s)$ has n distinct real eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ and n linearly independent right and left eigenvectors, denoted by v_i and w_i correspondingly, that is,

$$\mathbb{J}(x_s)v_i = \lambda_i v_i,$$

$$\mathbb{J}(x_s)^\top w_i = \lambda_i w_i,$$

$$i = 1, ..., n$$

and, in addition, we impose the normalisation condition $v_i^\top v_i = w_i^\top v_i = 1$ $(\forall i)$, then for all

$i = 1, ..., n$ (x_s, v_i, wI) is a fixed point of the GAD

$$\dot{x} = F(x) - 2\frac{\langle F(x), w\rangle}{\langle w, v\rangle}v,$$

$$\dot{v} = (\nabla F(x))v - \alpha(v)v,$$

$$\dot{w} = (\nabla F(x))^\top w - \beta(v, w)w.$$

Furthermore, among these n fixed points, there exists one fixed point (x_s, v_j, w_j) which is linearly stable if and only if x_s is an index-1 saddle point of the original dynamical system $\dot{x} = F(x)$ and the eigenvalue λ_j corresponding to this v_j, w_j is the only positive eigenvalue of $\mathbb{J}(x_s)$.

Proof. We only provide an overview of this proof. For the details, refer to [7].

An overview of the proof is as follows: at a fixed point, the Jacobian matrix of the GAD will be zero elsewhere apart from the diagonal elements. Therefore, the eigenvalues and eigenvectors of the Jacobian at the fixed point can be calculated from the diagonal block elements. From the the characteristics of the resulting eigenvalues such as the multiplicities one can deduce the properties stated in the theorem. □

5.3. Example: Analysis of a Gradient System

Since the work in this thesis deals with only gradient systems, we will examine an example of that type and apply GAD to it. We will use the form shown in equations 5.3a and 5.3b.

In order to have a better grasp on the dynamics described by GAD, we will reduce it to a very simple form. Firstly, by using the normalisation condition $\boldsymbol{v}^\top \boldsymbol{v} = 1$ and using a relation parameter τ for the direction of v, we get

$$\dot{x} = -\nabla V(x) + 2\langle \nabla V, v\rangle v,$$

$$\tau\dot{v} = -\nabla^2 V(x)v + \langle v, \nabla^2 V v\rangle v.$$

In order to get a closed system for x, we will let $\tau \to 0$, to obtain only

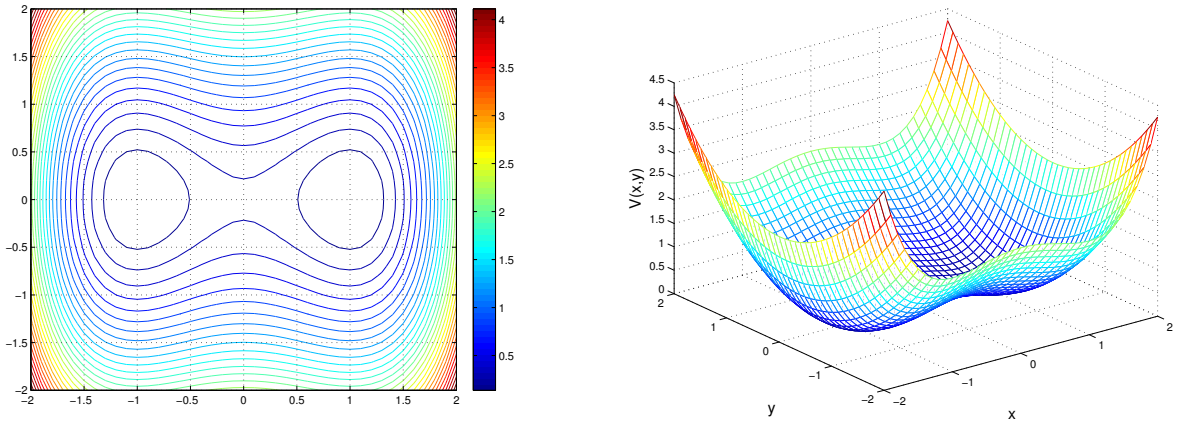
$$\dot{x} = -\nabla V(x) + 2\langle \nabla V, v(x)\rangle v(x), \tag{5.7}$$

where $v(x)$ is simply just the eigenvector of $\nabla^2 V(x)$ corresponding to the smallest eigenvalue.

Now lets consider the following two-dimensional system

$$V(x, y) = \frac{1}{4}(x^2 - 1)^2 + \frac{1}{2}\mu y^2$$

where μ is a positive parameter. The contour and surface plots of this system are shown on figure 5.2. The two stable critical points (and hence the two stable fixed points of the associated dynamics) are at $(1, 0)$ and $(-1, 0)$. It is also visible from the illustration that $(0, 0)$ is the index-1 saddle point. The gradient and Hessian of V are



(a) Contour Plot (b) Surface Plot

Figure 5.2.: *Contour and Surface plot of* $V(x,y) = \frac{1}{4}(x^2-1)^2 + \frac{1}{2}\mu y^2$

$$\nabla V(x,y) = \begin{bmatrix} x(x^2-1) \\ \mu y \end{bmatrix},$$

$$\nabla^2 V(x,y) = \begin{bmatrix} 3x^2-1 & 0 \\ 0 & \mu \end{bmatrix}.$$

The eigenvalues and eigenvectors of the Hessian at a point (x,y) are

$$\lambda_1 = 3x^2 - 1 \text{ and } v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix};$$

$$\lambda_2 = \mu \text{ and } v_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We know that $\mu > 0$, so the smallest eigenvalue of the Hessian will depend on which is smaller between λ_1 and λ_2. The GAD direction will depend on this comparison. Therefore, the direction corresponding to the smallest eigenvalue that will be picked by GAD, which we will denote as v_{GAD}, is

$$v_{GAD}(x) = \begin{cases} v_1, & \text{if } |x| < \sqrt{\frac{1+\mu}{3}} \\ \\ v_2, & \text{if } |x| > \sqrt{\frac{1+\mu}{3}} \end{cases} \tag{5.8}$$

So, if we define

$$V_1(x,y) = -\frac{1}{4}(x^2-1)^2 + \frac{1}{2}\mu y^2$$

and

$$V_2(x,y) = \frac{1}{4}(x^2-1)^2 - \frac{1}{2}\mu y^2,$$

where are choice for the signs in V_1 correspond to the GAD direction $V_{GAD} = v_1$ and V_2 to that of $V_{GAD} = v_2$.

We can deduce from all this that the value of μ determines what direction GAD chooses. Therefore, in order to use the GAD (5.8), we define a modified version of the function V, which we will call

V_{GAD}, that will serve as the underlying function that will drive the GAD (5.8) as

$$V_{GAD} = V_1(x,y) \cdot 1_{|x|<\sqrt{\frac{1+\mu}{3}}} + V_2(x,y) \cdot 1_{|x|>\sqrt{\frac{1+\mu}{3}}} \tag{5.9}$$

where $1(x)$ an an indicator function — a function that is 0 when an element is not a member of a specified set and is 1 if it is.

The introduction of this definition of V_{GAD} may be quite confusing at first but it is easier to think of it as a manipulated version of the function V. After this manipulation, the point $(0,0)$, instead of being an index-1 saddle point of V, will now be a stable fixed point of V_{GAD} and the two points $(1,0)$ and $(-1,0)$ are the saddle points of V_{GAD} — they swap places. A closer look at V_{GAD} will show that it is similar to the index-1 function forms prescribed by Morse's lemma. This manipulation is dependent on the directions v_1 and v_2 and is done for illustrative purposes to show the the effect of using these directions as ascent directions. In other words, when we use these directions v_1 and v_2, it is as though the function V is actually V_{GAD} — or, V_{GAD} is the behaviour of V after perturbation by direction directions v_1 and v_2. Therefore, applying GAD to V is analogous to applying *normal steepest descent dynamics* to V_{GAD} — the *attractors* of both dynamics are the index-1 saddle points of V.

The function V_{GAD} is not continuous at $|x| = \pm\sqrt{\frac{1+\mu}{3}}$ (figure 5.3). The point $(0,0)$ becomes the unique local minimum of V_1, with a basin of attraction $\{(x,y) : -1 < x < 1\}$. Outside this basin of attraction, the dynamics tend to $(x = \pm\infty, y = 0)$ and the value of V_1 falls to $-\infty$. This shown by figure 5.4.

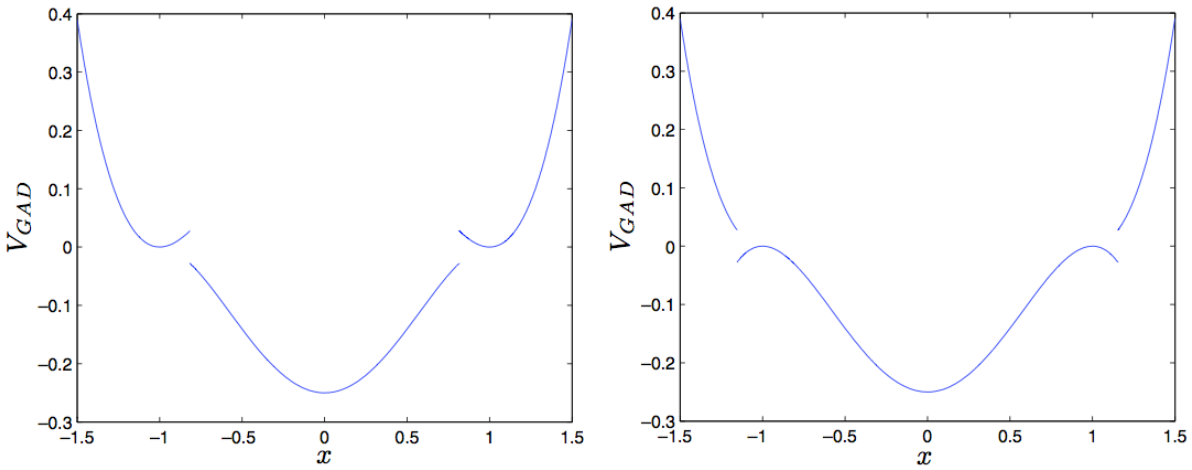
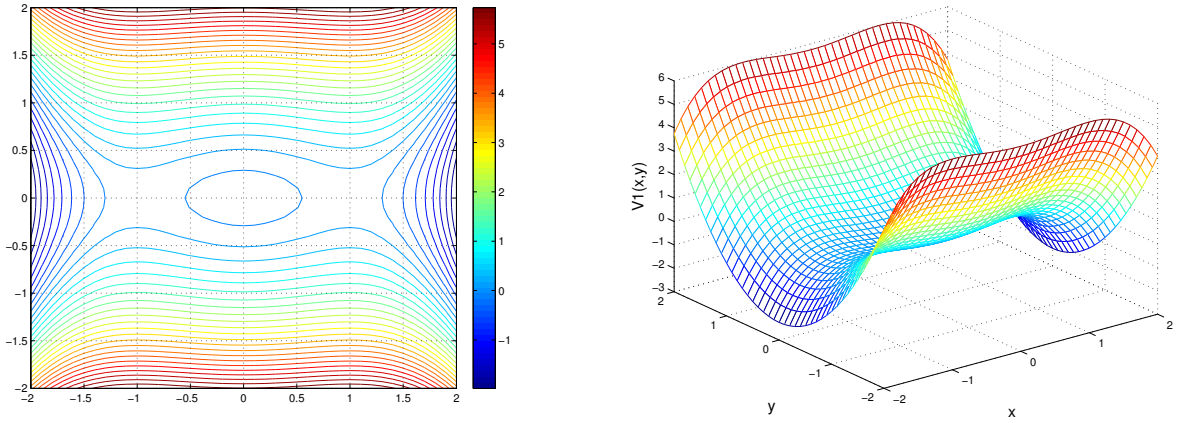


Figure 5.3.: *Illustration of the discontinuity of* $V_{GAD}(x, y = 0)$ *at* $|x| = \pm\sqrt{\frac{1+\mu}{3}}$ *from the Weinan E and Zhou paper. Left:* $\mu < 2$*, right:* $\mu > 2$

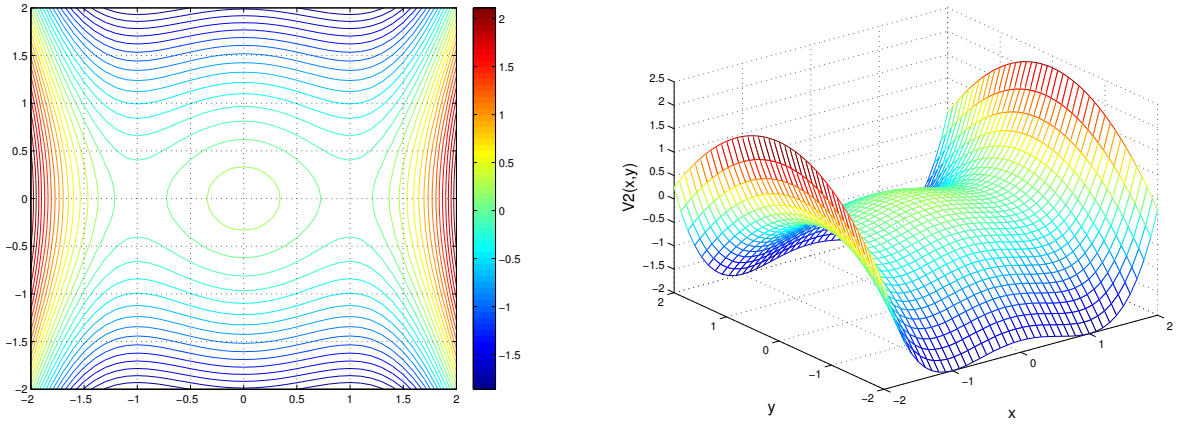
For V_2, as shown by figure 5.5, the point $(0,0)$ is the unique local maximum and all solutions of the dynamics go to $(x = \pm 1, y = \pm\infty)$.

If we start the GAD on V with an initial value $x_\pm = (\pm 1, 0)$, its two attractors, there are two possible ways that the dynamics escape them that depend on whether $\mu < 0$ or $\mu > 0$. Although



(a) Contour Plot (b) Surface Plot

Figure 5.4.: *Contour and Surface plot of* $V_1(x, y) = -\frac{1}{4}(x^2 - 1)^2 + \frac{1}{2}\mu y^2$



(a) Contour Plot (b) Surface Plot

Figure 5.5.: *Contour and Surface plot of* $V_1(x, y) = -\frac{1}{4}(x^2 - 1)^2 + \frac{1}{2}\mu y^2$

the behaviour that GAD induces makes $x_\pm$ behave like a saddle point for any $\mu \neq 2$, the unstable direction for $\mu < 2$ is $\pm v_2$ while the unstable directions for $\mu > 2$ is $\pm v_1$, as illustrated by figure 5.6 from [7]. Furthermore, from figure 5.6 and the discussion so far, it is clear that the basin of attraction of the function V_{GAD} is $-\sqrt{\frac{1+\mu}{3}} < x < \sqrt{\frac{1+\mu}{3}}$ for $\mu < 2$ and $-1 < x < 1$ for $\mu > 2$ — larger than the basins of attractions for the Newton-Raphson method [7]. As a result, the GAD with an initial value (x_0, y_0) near the local minimum $x_\pm$ of V escapes its basin of attraction and then converges to point $(0, 0)$ of our interest when $\mu > 2$ and $|x_0| < 1$.

The above discussion shows and suggests that GAD may not necessarily converge globally and instabilities can occur when GAD is used as a numerical algorithm. When instabilities occur, one may simply re-initialise the initial position or the direction [7].

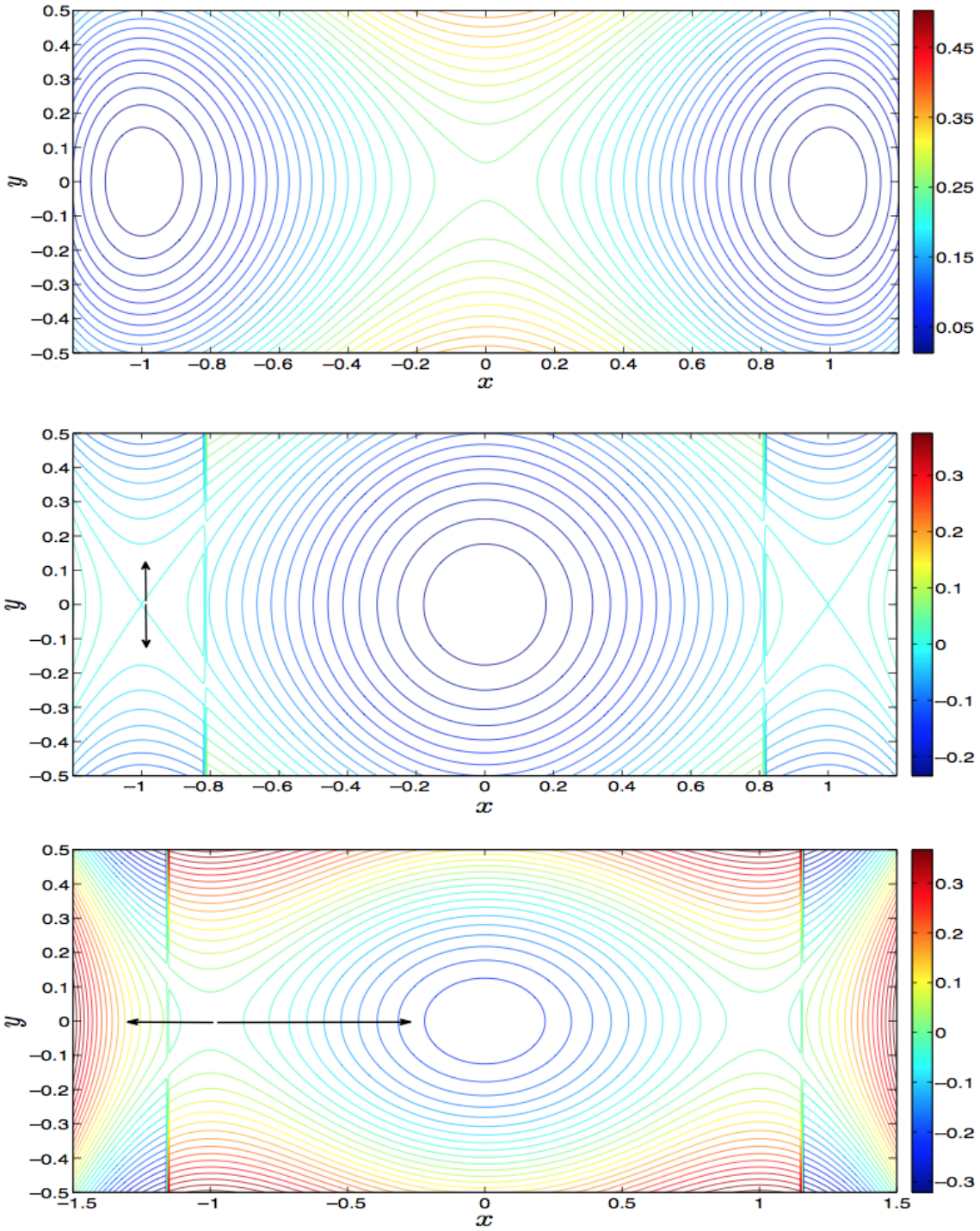


Figure 5.6.: *The contour plots of V, V_{GAD} for $\mu = 1$ and V_{GAD} for $\mu = 3$, from the top to the bottom, respectively. For the plot of V_{GAD}, the plot of V_1 lies in the region $-\sqrt{\frac{1+\mu}{3}} < x < \sqrt{\frac{1+\mu}{3}}$ and for V_2 it's on the two outer regions. The arrows show the direction of the flow of GAD equation (5.7).*

5.4. Conclusion

In this chapter we have introduced GAD which describes dynamical systems that escape the basins of attraction of minima and converge to index-1 saddle points. With further exposition and analysis on the case on gradient systems, it was discovered that GAD may not always converge toward an index-1 saddle point as time goes to infinity despite being able to escape the basins of attraction of the maxima and minima. This is due to some a phenomenon that can be only be flimsily described as a form of sensitivity to initial conditions. This may be a problem for numerical algorithms that may want to use the GAD paradigm.

6

Distillation of the Gentlest Ascent Dynamics into a Numerical Algorithm

6.1. Introduction

Our goal now is to construct a numerical algorithm from the GAD equations. Before proceeding, we need to define the context that we will use in solving the problem at hand.

Sticking to the conventions in topology, we will think of this problem in terms of finding the saddle points on of a surface or, more generally, a hyper-surface. We will use the word *surface* for the sake of brevity. Let us consider a real-valued function $V(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, whose saddle points we want to find using the GAD equations

$$\dot{\mathbf{x}} = -\nabla V(\mathbf{x}) + 2\frac{\langle \nabla V(\mathbf{x}), \mathbf{v} \rangle}{\langle \mathbf{v}, \mathbf{v} \rangle}\mathbf{v}, \tag{6.1a}$$

$$\dot{\mathbf{v}} = -\nabla^2 V(\mathbf{x})\mathbf{v} + \frac{\langle \mathbf{v}, \nabla^2 V(\mathbf{x})\mathbf{v} \rangle}{\langle \mathbf{v}, \mathbf{v} \rangle}\mathbf{v}. \tag{6.1b}$$

In order to simplify the equations above to a less clattered form we will assume the normalisation condition $\mathbf{v}^\top \mathbf{v} = 1$ holds and, borrowing from the convention used in the illustration of GAD for non-gradient systems by Weinan E and Zhou in [7], we will denote the **force** $-\nabla V(\mathbf{x})$ as $\mathbf{F}(\mathbf{x})$. The force here can be imagined as what "causes" the direction of the trajectories' movement of the steepest descent dynamics. We can also let the **Hessian** $\nabla^2 V(\mathbf{x})$ be denoted as $\mathbf{H}(\mathbf{x})$. Therefore, the resulting simplified version of the GAD equations becomes

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v} \rangle \mathbf{v} \tag{6.2a}$$

$$\dot{\mathbf{v}} = -\mathbf{H}(\mathbf{x})\mathbf{v} + \langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v} \rangle \mathbf{v}. \tag{6.2b}$$

As before, the second equation defines the dynamics of the "ascent" direction $\mathbf{v}$. The first term $-\mathbf{H}(\mathbf{x})\mathbf{v}$ on the right hand side makes sure that $\mathbf{v}$ converges to the eigenvector corresponding to the smallest eigenvalue of the Hessian $\mathbf{H}(\mathbf{x})$ and the second term $\langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle\mathbf{v}$ ensures that $\mathbf{v}$ is of unit length. The first equation perturbs the steepest descent dynamics by making them follow the "ascent" direction [7, 9].

6.2. Discretisation of the GAD Equations

An important step in constructing a numerical algorithm using GAD is discretising the dynamics because we want to make use of numerical evaluations implemented on a computer. For the first equation 6.2a,

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v}\rangle\mathbf{v},$$

$$\Rightarrow \frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v}\rangle\mathbf{v}, \text{ where } t \text{ is time.}$$

Considering discrete time steps

$$\frac{\Delta\mathbf{x}}{\Delta t} = \mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v}\rangle\mathbf{v}$$

$$\Rightarrow \frac{\mathbf{x}_{t+1} - \mathbf{x}_t}{\Delta t} = \mathbf{F}(\mathbf{x_t}) - 2\langle \mathbf{F}(\mathbf{x_t}), \mathbf{v_t}\rangle\mathbf{v_t}$$

$$\Rightarrow \mathbf{x}_{t+1} - \mathbf{x}_t = \Big[\mathbf{F}(\mathbf{x_t}) - 2\langle \mathbf{F}(\mathbf{x_t}), \mathbf{v_t}\rangle\mathbf{v_t}\Big] \times \Delta t$$

which becomes

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{F}(\mathbf{x_k}) \cdot \Delta t - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v}\rangle\mathbf{v} \cdot \Delta t.$$

And for the second equation 6.2b,

$$\dot{\mathbf{v}} = -\mathbf{H}(\mathbf{x})\mathbf{v} + \langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle\mathbf{v},$$

$$\Rightarrow \frac{d\mathbf{v}}{dt} = -\mathbf{H}(\mathbf{x})\mathbf{v} + \langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle\mathbf{v}$$

Considering discrete time steps gives

$$\frac{\Delta\mathbf{v}}{\Delta t} = -\mathbf{H}(\mathbf{x})\mathbf{v} + \langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle\mathbf{v}$$

$$\Rightarrow \frac{\mathbf{v}_{t+1} - \mathbf{v}_t}{\Delta t} = -\mathbf{H}(\mathbf{x_t})\mathbf{v_t} + \langle \mathbf{v_t}, \mathbf{H}(\mathbf{x_t})\mathbf{v_t}\rangle\mathbf{v_t}$$

$$\Rightarrow \mathbf{v}_{t+1} - \mathbf{v}_t = \Big[-\mathbf{H}(\mathbf{x})\mathbf{v_t} + \langle \mathbf{v_t}, \mathbf{H}(\mathbf{x_t})\mathbf{v_t}\rangle\mathbf{v_t}\Big] \times \Delta t$$

which yields

$$\mathbf{v_{t+1}} = \mathbf{v_t} - \mathbf{H}(\mathbf{x_0})\mathbf{v_t} \cdot \Delta t + \langle \mathbf{v_t}, \mathbf{H}(\mathbf{x_t})\mathbf{v_t}\rangle\mathbf{v_t} \cdot \Delta t.$$

Therefore, the discretised GAD equations are

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{F}(\mathbf{x_k}) \cdot \Delta t - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v}\rangle\mathbf{v} \cdot \Delta t \tag{6.3a}$$

$$\mathbf{v_{t+1}} = \mathbf{v_t} - \mathbf{H}(\mathbf{x_0})\mathbf{v_t} \cdot \Delta t + \langle \mathbf{v_t}, \mathbf{H}(\mathbf{x_t})\mathbf{v_t}\rangle\mathbf{v_t} \cdot \Delta t. \tag{6.3b}$$

We will now proceed to formulate a numerical algorithm.

6.3. Basic and Natural GAD Numerical Algorithm

Now that we have the discretised equations, we now need to think about what conditions must be true initially for our algorithm.

Naturally, we can have an initial point $\mathbf{x_0}$ and a random initial direction $\mathbf{v}_0$ and then apply GAD on them iteratively. This gives rise to the basic algorithm shown in algorithm 6.1 .

Algorithm 6.1 Basic and Natural GAD Numerical Algorithm

Require: Initial point $\mathbf{x_0}$
Require: Initial normalised direction $\mathbf{v_0}$
Require: Function $V(\mathbf{x})$
Require: Force $\mathbf{F}(\mathbf{x})$ and Hessian $\mathbf{H}(\mathbf{x})$
Require: Tolerance parameters τ_1, τ_2 and τ_3
Require: Time-step Δt
Require: Set $k = 1$
1: **while** ($\epsilon_1 > \tau_1$ & $\epsilon_2 > \tau_2$ & $\epsilon_3 > \tau_3$) **do**
Evaluate force $F(\mathbf{x_k})$;
Evaluate Hessian $H(\mathbf{x_k})$;
Move to next point

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{F}(\mathbf{x_k}) \cdot \Delta t - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v} \rangle \mathbf{v} \cdot \Delta t$$

Get new direction

$$\mathbf{v}_{k+1} = \mathbf{v}_k - \mathbf{H}(\mathbf{x_k})\mathbf{v_k} \cdot \Delta t$$

Normalise direction

$$\mathbf{v_{k+1}} = \frac{\mathbf{v_{k+1}}}{|\mathbf{v_{k+1}}|}$$

Calculate parameters for termination conditions

$$\epsilon_1 = V(\mathbf{x_{k+1}}) - V(\mathbf{x_k})$$

$$\epsilon_2 = ||\mathbf{F}(\mathbf{x_k}) - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v} \rangle \mathbf{v}||$$

$$\epsilon_3 = ||\mathbf{F}(\mathbf{x_k})||$$

Set flag for next iteration

$$k = k + 1$$

This algorithm starts with an initial point $\mathbf{x_0}$ and normalised random direction $\mathbf{v_0}$ and calculates the values of the force $\mathbf{F}$ and Hessian $\mathbf{H}$ at the point $\mathbf{x_0}$. The algorithm then uses these values to calculate the next point and the next normalised direction, then it iterates. This sequence of using the GAD sub-equations is in the true spirit of GAD because it simulates the gradual cooling process which eventually converges on a separatrix. It is from this separatrix that the GAD dynamics converge toward a saddle point. This is why we have called this algorithm both *basic* and *natural*.

Notice that in our treatment we use extra information to aid the iteration such as the step-size Δt and error tolerances τ. The error tolerances are used to implement termination conditions for the algorithm. In the case of the algorithm 6.1, the conditions used are based on three key aspects of the function. These are:

1. **Change in Function Value**: $V(\mathbf{x_{k+1}}) - V(\mathbf{x_k}) < \tau$;
2. **Norm of the Residual**: $||\mathbf{F}(\mathbf{x_k}) - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v}\rangle \mathbf{v}|| < \tau$; and
3. **Norm of the Gradient**: $||\mathbf{F}(\mathbf{x_k})|| < \tau$.

The first condition relies on the fact that at a solution the change in the function value between iterations will be very small. The second condition uses the knowledge that at a fixed point the residual $\dot{\mathbf{x}} \approx 0$. The third condition uses the property of the gradient at any critical point. In this case, the norm of the gradient being 0. The value that can be set for τ is dependent on the degree of accuracy required.

On the other hand, the step-size Δt is used to determine how far in the GAD direction the next point will be in each iteration. The value must be small enough to allow the algorithm to stabilise into a separatrix. Of course, this will in turn result in a large number of iterations for the trajectory to stabilise. On the other hand, a large step-size may be too aggressive and the algorithm may never converge. For the basic algorithm, this value is set to be a small fixed value like 0.1 or 0.05.

So does this algorithm actually work? Let us try it on the same example function $V(x, y) = \frac{1}{4}(x^2 - 1)^2 + \frac{1}{2}\mu y^2$ with $\mu = 3$ used in the GAD exposition chapter. This function $V(x, y)$ has minima at $(\pm 1, 0)$ and a saddle point at the origin $(0, 0)$ as shown in the figure 6.1. Various initial-

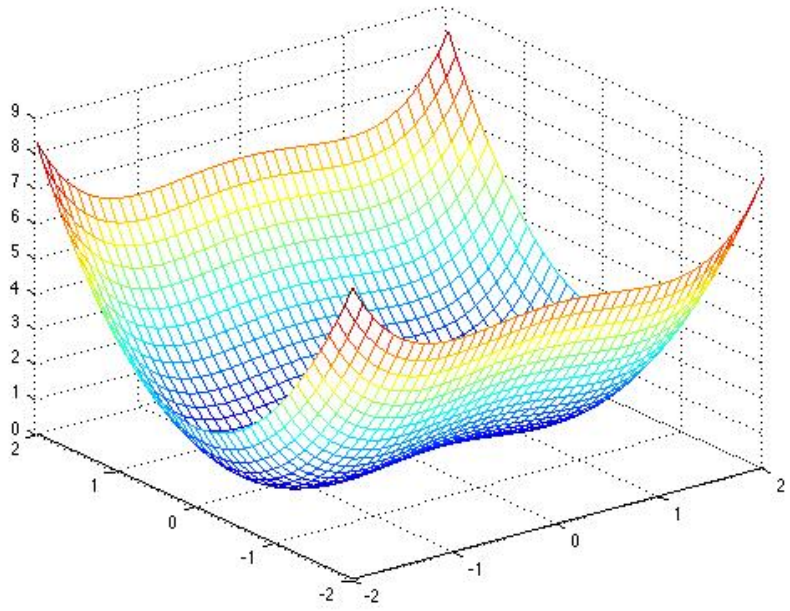
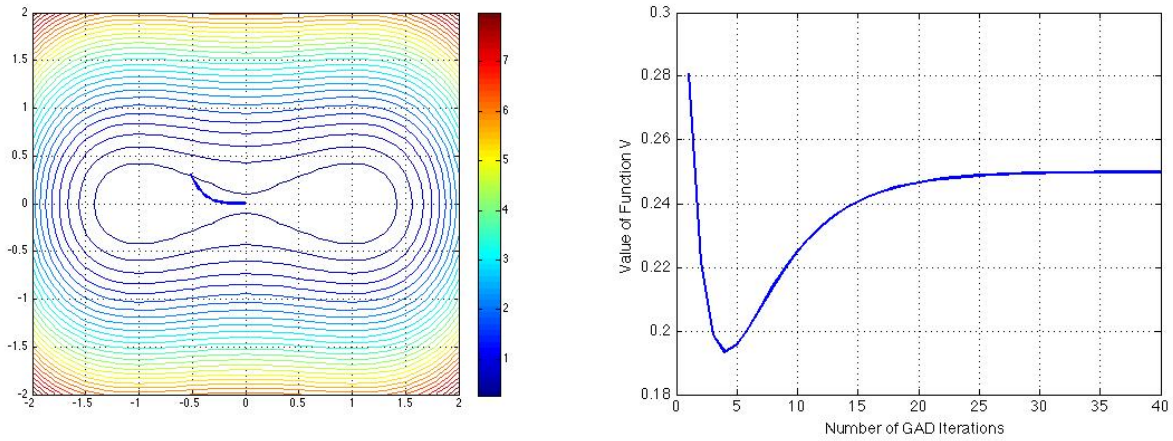


Figure 6.1.: *Function* $V(x, y) = \frac{1}{4}(x^2 - 1)^2 + \frac{3}{2}y^2$ *with minima at* $(\pm 1, 0)$ *and a saddle point at the origin* $(0, 0)$.

isations of the algorithm with $\Delta t = 0.1$ and $\tau = 0.005$ give very interesting results.

Figure 6.2 shows the GAD solution trajectory with the algorithm initialised at point $(-0.51, 0.31)$ with a random initial direction $[-1, 0]$. As illustrated, the trajectory is stable and converges toward the saddle point at $(0, 0)$. As can also be seen from the elevation plot in figure 6.2, the trajectory escapes the basin of attraction of the minima at $(-1, 0)$ and "gently ascends" toward the saddle

point. The trajectory does not head toward the saddle point directly but eventually "cools" from the initial configuration until it stabilises into a separatrix before it converges toward the saddle point. This is expected as it was stated by Weinan E and Zhou in the original paper.

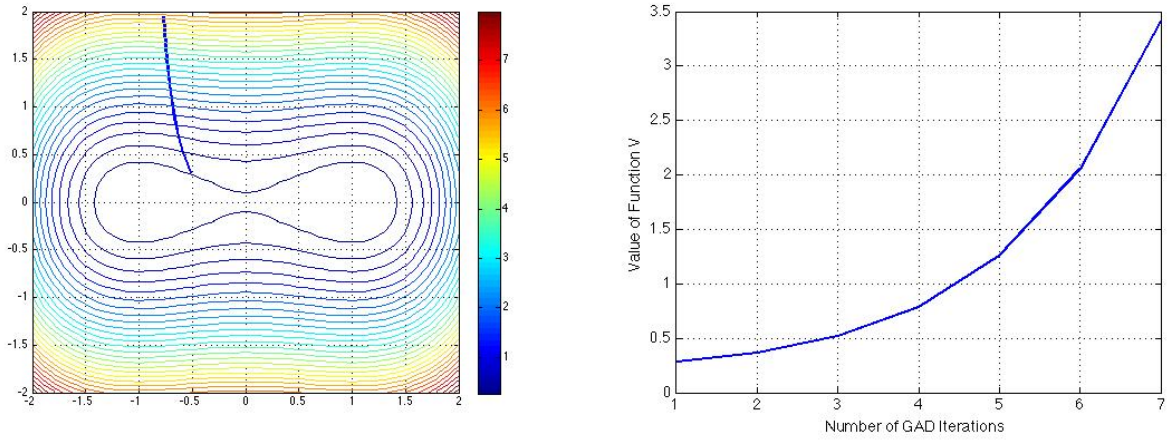


(a) Stable GAD Solution Trajectory

(b) GAD Elevation Plot

Figure 6.2.: *GAD trajectory and elevation plots on* $V(x, y)$ *with algorithm initialised from point* $(-0.51, 0.31)$ *with initial direction* $[-1, 0]$. *The minima at* $(\pm 1, 0)$ *and a saddle point at the origin* $(0, 0)$

Now, let's try to change the initial direction to $[0, -1]$ and see what happens. As can be seen from figure 6.3 the resulting trajectory is unstable as it is heading higher and away from the saddle point. Other initialisations done nearer to the minima $(-1, 0)$ yield similar results as seen from figure 6.4b.

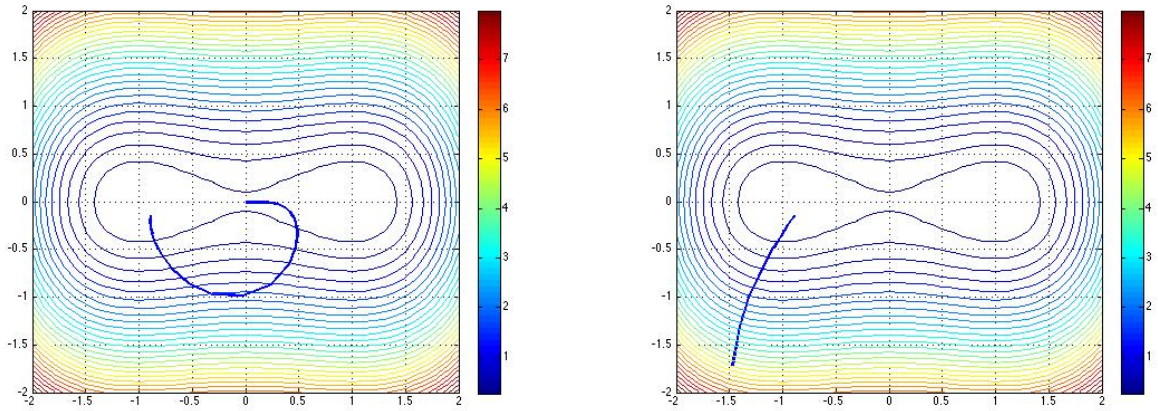


(a) Unstable GAD Solution Trajectory

(b) GAD Elevation Plot

Figure 6.3.: *GAD trajectory and elevation plots on* $V(x, y)$ *with algorithm initialised from point* $(-0.51, 0.31)$ *with initial direction* $[0, -1]$. *The minima at* $(\pm 1, 0)$ *and a saddle point at the origin* $(0, 0)$

We were warned about this instability in [7] and E and Zhou advised reinitialising either the initial direction or the initial point. However, this is undesirable. What we want is an algorithm that



(a) A Gradually Stable GAD Solution Trajectory

(b) An Unstable GAD Solution Trajectory

Figure 6.4.: *Other GAD solution trajectories on function* $V(x, y) = \frac{1}{4}(x^2 - 1)^2 + \frac{3}{2}y^2$

is numerically stable and always converges toward an index-1 saddle point. Our task now is to achieve this.

6.4. Improvements on the Natural GAD Numerical Algorithm: Initial Conditions

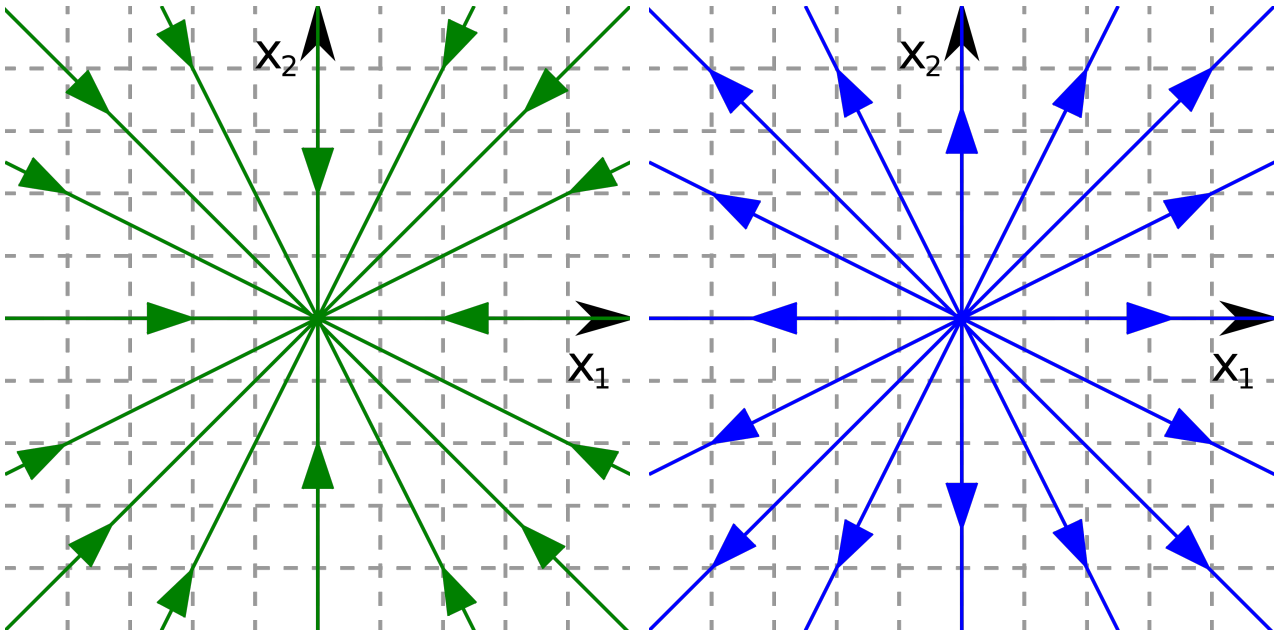
In order to solve the instability problem we need to figure out what initial conditions lead to stable trajectories. To do this, we need to gain insight into the inner workings of GAD. The problem of finding saddle points close to local minima means that trajectories need to escape the basins of attraction of the minima along preferential directions in a multidimensional configurational space. The difficulty we are facing now is being able to have a theoretical handle on these preferential directions because they represent only a very small fraction of all directions emanating from locally stable fixed point on the high dimensional surface.

The task at hand now is to reduce the size of the configurational space by selecting a small fraction of it that represents our preferential directions. Firstly, from dynamical systems theory, we know that the reason some initial conditions led to unstable GAD solution trajectories is that these initial conditions caused the trajectories to cross the boundary of behaviour dividing the part of the phase space where GAD is well behaved to where it is not. So, if we had an idea of where this boundary is or how to ensure our dynamics do not cross it then we can solve the problem. In other words, we want to find the initial conditions that always ensure our algorithm is initialised in the basin of attraction of the saddle points with respect to GAD.

To proceed with this line of thought, we remember that each fixed point (stationary points or critical points) have associated with them sets where the dynamics are influenced by this point. Let us consider the steepest descent dynamics for a real valued function $V(\mathbf{x})$ such as

$$\dot{\mathbf{x}} = -\nabla V(\mathbf{x}).$$

In this case, a local minimum (locally stable fixed point) has an associated basin of attraction where all the dynamics converges to that local minimum as time goes to infinity. This basin of attraction is called the **stable manifold/set** of the locally stable fixed point. Trajectories with initial conditions within this set eventually converge toward this stable fixed point as time passes. This is shown in figure 6.5a. Similarly, a local maximum (locally unstable fixed point) has an associated basin of "repulsion" where all the steepest dynamics diverge away the local maximum as time goes to infinity. This basin of "repulsion" is called the **unstable manifold/set** of the locally stable fixed point. This is also shown in figure 6.5b. Trajectories with initial conditions within this set eventually diverge away this unstable fixed point as time passes.



(a) Stable attracting fixed point and its stable manifold (b) Unstable repelling fixed point and its unstable manifold

Figure 6.5.: *Attractor and Repellor and their associated Manifolds (Source: Wikimedia Commons)*

There is another type of fixed point that has the combined behaviour of the two aforementioned fixed point types because it has both stable and unstable manifolds. These are saddle points. This means that for the steepest descent dynamics above these distinct stable and unstable manifolds for the associated distinct stable and unstable fixed points are separated by separatrices on which the dynamics converge toward saddle points. This is shown in figure 6.6. Therefore, we can surmise that a separatrix will occur in between any two fixed points. Why? Because their basins of behaviour interact to form this separatrix. This knowledge tells us that our initial point for GAD should be in the vicinity of at least two fixed points, preferably in between two fixed points, in order to ensure it is in the basin of attraction of the saddle point. Intuitively, this is equivalent to saying that on a mountainous surface with many hills and troughs you will find a saddle point close to any two hills or valleys.

What about the initial direction? What insight can we use to ensure they led to stable GAD trajectories? Samanta and E in [9] observed that in materials physics many deformation processes proceed along minimum energy paths and they can be easily traced using the softest eigenmodes of the Hessian. Why is this important to us? As stated in the introductory chapter, the dynamics of many complex systems proceed via sequences of metastable states and when the system is a simple Potential Energy Surface (PES) like the aforementioned deformation process the path between

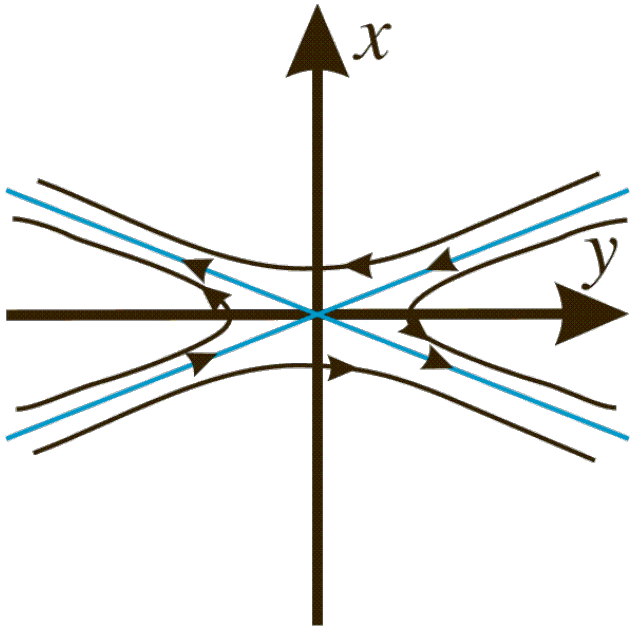


Figure 6.6.: *Saddle point. Blue lines are on Separatrix (Source: Wikimedia Commons)*

these states follows a minimum energy path that passes through a saddle point [9].

This insight and analysis now enables us to define the desirable initial conditions for the numerical GAD algorithm.

Definition 6.4.1. Desirable Initial Conditions for GAD

1. *The initial point* $\mathbf{x_0}$ *must be close to at least two local fixed point or close to a saddle point;*
2. *The initial direction* $\mathbf{v_0}$ *must correspond to the eigenvector with the smallest eigenvalue of the Hessian at the point* $\mathbf{x_0}$ *close to the local fixed point.*

The first point in the definition is self-explanatory. All it says is that the point must be within the vicinity of at least two minima or maxima, or between a minimum and a maximum. This is in order to ensure it's close to a saddle point. This condition is quite easy to satisfy if the function has a large number of local maxima and minima, as is the case with the functions we are interested in, because practically every point is in the vicinity of at least two stationary points.

The second condition, however, is quite interesting. It states that the initial direction must be the most unstable eigenvector of the Hessian of the initial point. Traditionally, negative eigenvalues correspond to unstable eigenvectors of the Hessian. Therefore, the smaller the eigenvalue of an eigenvector then the more unstable it is. In other words, during initialisation, we need to choose an initial direction corresponding to the "softest" or the "smallest" eigenvalue because this direction is part of the configuration space that would enable the GAD to gradually relax into the the correct trajectory toward a saddle point.

Algorithm 6.2 shows an improved Natural GAD numerical algorithm based on our new insights. In order to ensure that the initial point x_0 is in the vicinity of at least two locally fixed points one may simply run a few steps of the steepest descent algorithm and start the GAD algorithm at the point much closer to a minimum. Or find the the closest minimiser to a random start point and

then slightly perturb from this point to get the starting point x_0.That is, from a random point x_r, we do a local minimisation

$$x_{lm} = \min \{f(x_r)\},$$

and then perturb this local minimiser x_{lm} by small number ϵ in a direction $\mathbf{e}$ to get our start point x_0; that is,

$$x_0 = x_{lm} \pm \epsilon e.$$

For the initial direction $\mathbf{v_0}$, we find the softest eigenvector of the Hessian $\mathbf{H}$ at a point x_0 by solving the problem

$$\mathbf{v_0} = \min \{\mathbf{v}^\top \mathbf{H}\mathbf{v} : \mathbf{v}^\top \mathbf{v} = 1\},$$

and use the resulting x_0 and $\mathbf{v_0}$ in the algorithm.

In cases where determining the eigenvectors with the smallest eigenvalues is computationally demanding or infeasible Samanta and E in [9] discovered that the unit force vector at a point close to a locally stable fixed point has similar characteristics to the low-lying eigenvectors that correspond to the preferential directions. Therefore, this can be used as an initial direction in those cases.

Algorithm 6.2 Natural GAD Numerical Algorithm

Require: Initial point $\mathbf{x_0}$ close to at least two local fixed points.
Require: Initial normalised direction $\mathbf{v_0}$ corresponding to minimum eigenvalue of Hessian at $\mathbf{x_0}$.
Require: Function $V(\mathbf{x})$
Require: Force $\mathbf{F}(\mathbf{x})$ and Hessian $\mathbf{H}(\mathbf{x})$
Require: Tolerance parameters τ_1, τ_2 and τ_3
Require: Time-step Δt
Require: Set $k = 1$
1: **while** ($\epsilon_1 > \tau_1$ & $\epsilon_2 > \tau_2$ & $\epsilon_3 > \tau_3$) **do**
Evaluate force $F(\mathbf{x_k})$;
Evaluate Hessian $H(\mathbf{x_k})$;
Move to next point

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{F}(\mathbf{x_k}) \cdot \Delta t - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v}\rangle \mathbf{v} \cdot \Delta t$$

Get new direction

$$\mathbf{v}_{k+1} = \mathbf{v}_k - \mathbf{H}(\mathbf{x_k})\mathbf{v_k} \cdot \Delta t$$

Normalise direction

$$\mathbf{v_{k+1}} = \frac{\mathbf{v_{k+1}}}{|\mathbf{v_{k+1}}|}$$

Calculate parameters for termination conditions

$$\epsilon_1 = V(\mathbf{x_{k+1}}) - V(\mathbf{x_k})$$

$$\epsilon_2 = ||\mathbf{F}(\mathbf{x_k}) - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v}\rangle \mathbf{v}||$$

$$\epsilon_3 = ||\mathbf{F}(\mathbf{x_k})||$$

Set flag for next iteration

$$k = k + 1$$

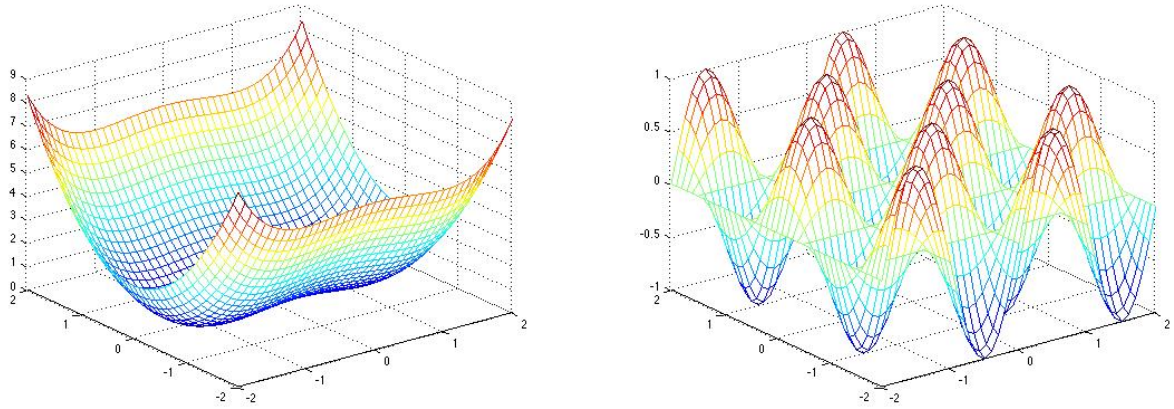
Implementation of the algorithm 6.2 and testing it on functions

$$V(x,y) = \frac{1}{4}(x^2-1)^2 + \frac{1}{2}3y^2$$

with $\mu = 3$ and

$$V(x,y) = sin(x)cos(y),$$

which is shown in figure 6.7, resulted in contour plots and trajectories as shown in figures 6.8 and 6.9.



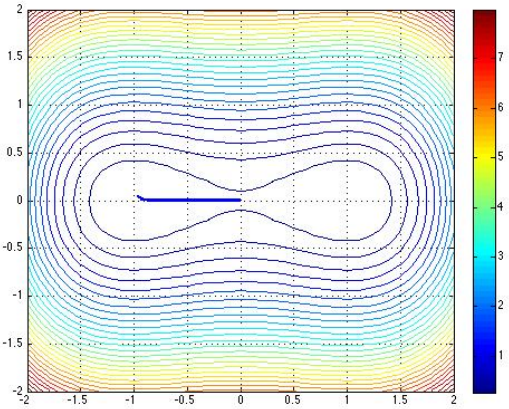
(a) $V(x,y) = \frac{1}{4}(x^2-1)^2 + \frac{1}{2}3y^2$ (b) $V(x,y) = sin(x)cos(y)$

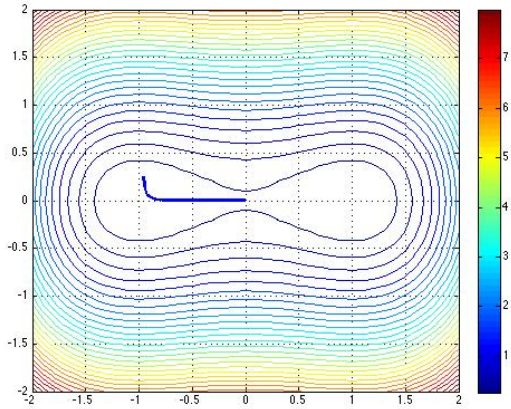
Figure 6.7.: *Test Functions for the Natural GAD Algorithm*

This algorithm is stable unlike the basic algorithm postulated in the previous section. However, it suffers in the aspect that it only achieves local convergence — that is, it can be unstable if the initial conditions do not satisfy the Desirable Initial Conditions defined previously. In fact, this result is hardly surprising because it is in the spirit that the GAD ODEs were derived in that the trajectories must gradually cool into the correct configuration. This gives us an intuitive idea about the bounds of the basin of attraction of the GAD which, as one can imagine, is much larger than local optimisation algorithms because it spans over the vicinities different types of critical points [7].

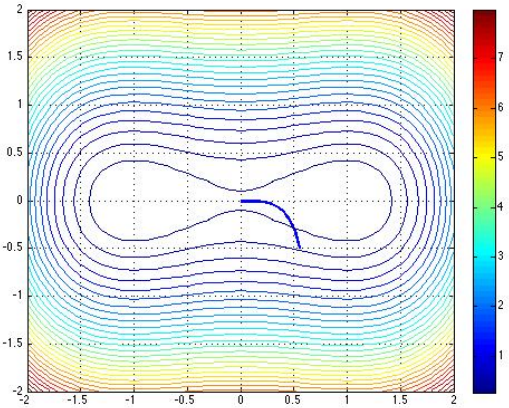
Tests on this algorithm showed some interesting observations which are listed below.

1. **GAD algorithm works with initial conditions near maxima**: As can be seen in figure 6.9, the GAD algorithm initiated near a locally unstable fixed point (maximum) still converged toward a saddle point. This makes sense because the unstable manifold of this unstable fixed point are much easier to find and hence the preferential directions for the GAD are easily found.
2. **Trajectory seeks out the closest saddle point**: The GAD equations and algorithm simulate a gradual process. The dynamics move from initial conditions and gradually "cool" toward an index-1 saddle point. During this relaxation process, anything can happen as the trajectory can even escape the basin of attraction depending on how large the step-size is. As can be seen from solution trajectories on figure 6.8, these trajectories can have long and

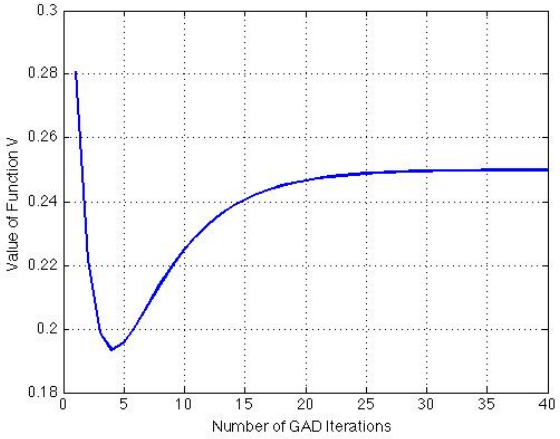




(a) Initial point close to minimum

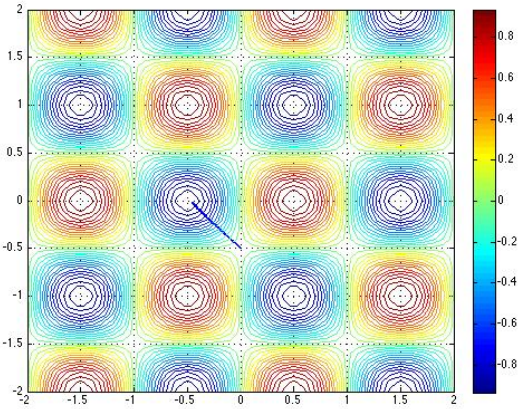


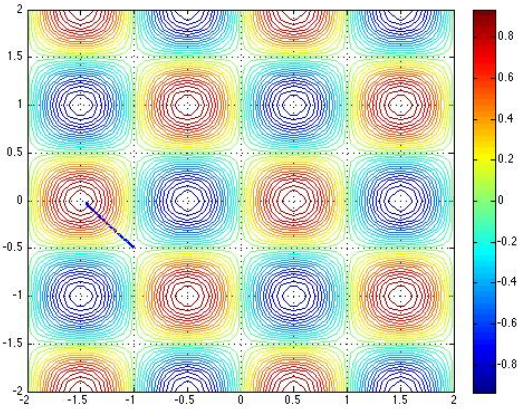
(b) Inital point from higher elevation



(c) Example Elevation plot

Figure 6.8.: *Solution Trajectories for the Natural GAD Algorithm on* $V(x,y) = \frac{1}{4}(x^2-1)^2 + \frac{1}{2}3y^2$





(a) Initial point close to a minimum

(b) Initial point close to a maximum

Figure 6.9.: *Solution Trajectories for the Natural GAD Algorithm on* $V(x,y) = sin(x)cos(y)$

winding paths before relaxing to the softest eigenvector in the vicinity of a saddle point and thus converge toward it. With the initial conditions we have defined, the trajectories always head toward the closest index-1 saddle point if the step-size is small enough. For higher

dimensional cases, the Natural GAD will converge at the closest higher-index saddle point if the step-size used is not small enough for the algorithm to strictly follow GAD and find an index-1 saddle point. In general, if the step-size is small enough, the Natural GAD algorithm will converge at the closest index-1 saddle point. However, if the step-size is not large enough, the algorithm will converge at the closest saddle point regardless of its degree of instability.

3. **Trajectory elevation fluctatuatation**: As GAD has a gradual direction relaxation convergence scheme, some steps may be ascent steps and others may be descent steps as can be seen from the elevation plot in figure 6.8. This is as a result of the algorithm's tendency to gradually relax toward the separatrix; consequently, the resulting trajectory may ascend and descend on the function during this process. This is a problem because it's harder to efficiently implement a traditional step-size strategy. This will be discussed in section 6.6 of this chapter.

6.5. Rayleigh GAD Numerical Algorithm: Rayleigh Optimisation

At this point we have an effective algorithm using GAD to find saddle points. However, the Natural GAD has the following short-comings:

1. **No Global Convergence**: If the initial conditions do not satisfy the *Desirable Initial Conditions for GAD*, then the algorithm does not converge. Specifically, if a point is not in the vicinity of at least two fixed points, which less is likely in a multimodal function, then convergence fails.
2. **Rate of Relaxation to a Separatrix**: In the previous section, we observed that for the Natural GAD to work very well it needed to have a small enough step-size. This was for the purpose of allowing the solution trajectory to gradually relax into a separatrix before it can converge toward a saddle point. This relaxation process is slow and takes up a large fraction of the iterations of the algorithm.
3. **Use in Global Optimisation**: Our goal is to use a GAD-based algorithm to find saddle points that we want to use as transitional points for a global optimisation algorithm. The Natural GAD algorithm suffers from the fact that it requires a large number of iterations in order to converge. This large number of iterations translates to a larger number of function, gradient and Hessian evaluations which may be detrimental to algorithm complexity.

So, we ask the following questions: Is it possible to have a GAD-based algorithm that is never unstable regardless of the initial conditions? Can we construct such an algorithm with a faster rate of convergence toward a separatrix? Can we build an algorithm that converges at a saddle point with fewer iterations? Constructing such an algorithm is the goal of this section.

To take on this task we need to take a closer look at the GAD equations:

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v}\rangle \mathbf{v} \tag{6.4a}$$

$$\dot{\mathbf{v}} = -\mathbf{H}(\mathbf{x})\mathbf{v} + \langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle \mathbf{v}. \tag{6.4b}$$

As before, the second equation defines the dynamics of the "ascent" direction $\mathbf{v}$. The first term $-\mathbf{H}(\mathbf{x})\mathbf{v}$ on the right hand side makes sure that $\mathbf{v}$ converges to the eigenvector corresponding to the smallest eigenvalue of the Hessian $\mathbf{H}(\mathbf{x})$ and the second term $\langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle \mathbf{v}$ ensures that $\mathbf{v}$ is of

unit length. The first equation perturbs the steepest descent dynamics by making them follow the "ascent" direction [7, 9].

The second ODE can be constructed from the classical optimisation problem

$$\begin{aligned} \underset{\mathbf{v}}{\text{minimise}} \quad & \mathbf{v}^\top \mathbf{H}\mathbf{v} \\ \text{subject to} \quad & \mathbf{v}^\top \mathbf{v} = 1. \end{aligned} \tag{6.5}$$

A term of the form

$$\frac{\mathbf{v}^\top \mathbf{H}\mathbf{v}}{\mathbf{v}^\top \mathbf{v}}$$

is called the *Rayleigh Quotient* and it reduces to $\mathbf{v}^\top \mathbf{H}\mathbf{v}$ with the constraint $\mathbf{v}^\top \mathbf{v} = 1$. The optimisation above is called the **Rayleigh optimisation** and is a classical formulation finding the eigenvector corresponding to the smallest eigenvalue of a symmetric matrix [38]. The objective function in this case is $f(\mathbf{v}) = \mathbf{v}^\top \mathbf{H}\mathbf{v}$. Differentiating it gives $\nabla f(\mathbf{v}) = \mathbf{H}\mathbf{v}$ and because of the steepest descent minimisation the force is $-\nabla f(\mathbf{v}) = -\mathbf{H}\mathbf{v}$. Adding the constraint $\mathbf{v}^\top \mathbf{v} = 1$ the trajectory of the minimisation is

$$\dot{\mathbf{v}} = -\mathbf{H}\mathbf{v} + \langle \mathbf{v}, \mathbf{H}\mathbf{v} \rangle \mathbf{v}$$

which is essentially equation 6.4b.

If we replace the second ODE with the Rayleigh optimisation in our GAD algorithm then we speed up the rate at which the algorithm "cools" or relaxes toward a separatrix and, in turn, a saddle point. Thus, within each GAD iteration, we do many iteration steps to solve the Rayleigh optimisation at a point and one step to go to the next point as shown by algorithm 6.3 below.

The Rayleigh optimisation is not necessarily easy to do so any other calculation that yields the minimum eigenvector is fine for this step. However, caution should be taken because the matrix $\mathbf{H}$ is not necessarily positive definite to solve the resulting $\nabla f(\mathbf{v}) = 0$ as required by many algorithms. As a result, methodologies such as the Generalised Minimal Residual (GMRES) Method which do not need this requirement are desirable. Eigenvalue and eigenvector finding algorithms can also be used.

Testing this algorithm results the surface contours with trajectories shown in figures 6.10 and 6.11.

The tests on this algorithm show that it behaves well for cases where the initial conditions satisfy the Desirable initial Conditions. However, when the *Desirable Initial Conditions* are not satisfied, the convergence of the algorithm is peculiar because it converges at a point close to the minimum that is not a saddle point. This can be seen from the GAD solution trajectories shown in figure 6.10. This point, however, is not as random as it seems. We know that at a fixed point the value of $\dot{\mathbf{x}}$ from the GAD ODEs is 0. During the tests it was observed that the value of $\dot{\mathbf{x}} \approx 0$. This makes sense because the initial conditions are outside the basin of attraction (the *Desirable Initial Conditions* are not satisfied) and the odd point is the "most similar" to a saddle points in relation to the GAD ODEs. This result goes to validate the *Desirable Initial Conditions* defined before.

Algorithm 6.3 Rayleigh GAD Numerical Algorithm

Require: Initial point $\mathbf{x_0}$.
Require: Function $V(\mathbf{x})$.
Require: Force $\mathbf{F}(\mathbf{x})$ and Hessian $\mathbf{H}(\mathbf{x})$.
Require: Tolerance parameters τ_1, τ_2 and τ_3
Require: Time-step Δt
Require: Set $k = 1$

1: **repeat**

At point x_k, calculate $\mathbf{v}_k$ satisfying

$$\begin{aligned} \underset{\mathbf{v}}{\text{minimise}} \quad & \mathbf{v}^\top \mathbf{H}(x_k)\mathbf{v} \\ \text{subject to} \quad & \mathbf{v}^\top \mathbf{v} = 1. \end{aligned} \tag{6.6}$$

Move to next point using 1st ODE with minimiser v_k from above.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{F}(\mathbf{x_k}) \cdot \Delta t - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v} \rangle \mathbf{v} \cdot \Delta t$$

Calculate parameters for termination conditions

$$\epsilon_1 = V(\mathbf{x_{k+1}}) - V(\mathbf{x_k})$$

$$\epsilon_2 = ||\mathbf{F}(\mathbf{x_k}) - 2\langle \mathbf{F}(\mathbf{x_k}), \mathbf{v} \rangle \mathbf{v}||$$

$$\epsilon_3 = ||\mathbf{F}(\mathbf{x_k})||$$

Set flag for next iteration

$$k = k + 1$$

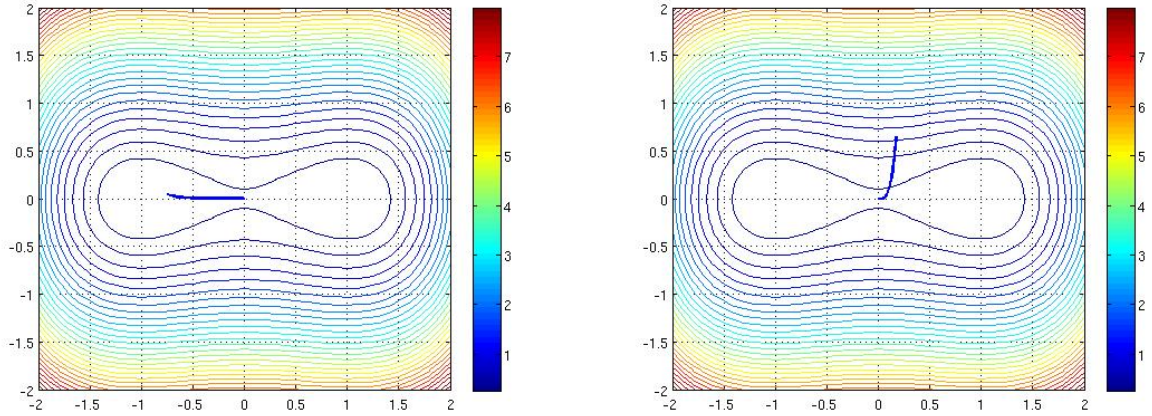
2: **until** ($\epsilon_1 > \tau_1$ & $\epsilon_2 > \tau_2$ & $\epsilon_3 > \tau_3$)

So, this algorithm can be used to test if the initial point satisfied the Desirable Initial Conditions. If the algorithm terminates at a saddle point, we know that the Desirable Initial Conditions were satisfied initially.

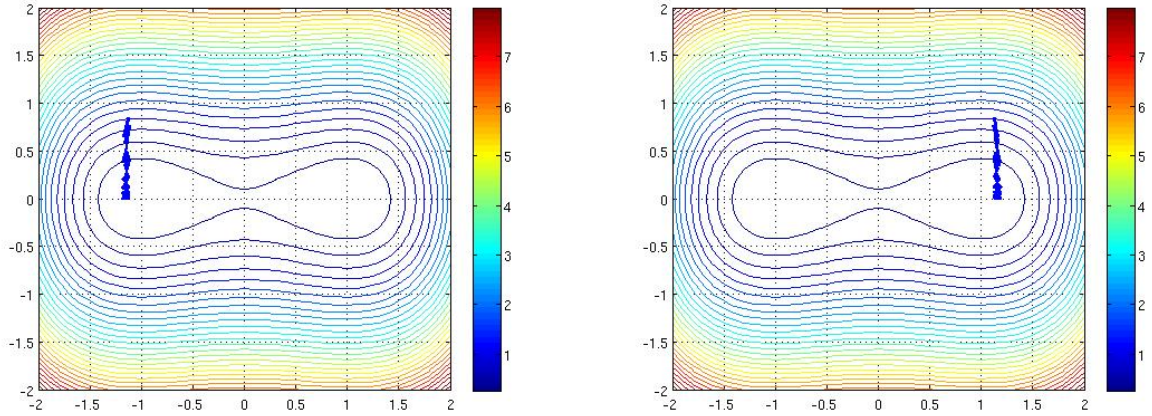
This absurd convergence is less likely as the number of fixed points on a function increases. It should be noted the toy example is only used to illustrate the aspects of Morse theory we want to exploit while solving the GAD ODEs. In reality, the we want to use this algorithm on a non-convex multimodal function that has numerous maxima and minima. In such cases, this peculiar result is highly unlikely. As can be seen in 6.11, this algorithm is well behaved as long as the *Desirable Initial Conditions* are satisfied, as is the case in the realistic non-linear multidimensional and multimodal functions we are interested in.

It is worth adding that because of the inclusion of an optimisation process in every iteration, this algorithm is more computationally demanding than its counterpart in the previous section. Also, as we had set out to do, the Rayleigh GAD algorithm relaxes to the desired direction at a much faster rate because it aggressively finds the most unstable direction at each point in the trajectory.

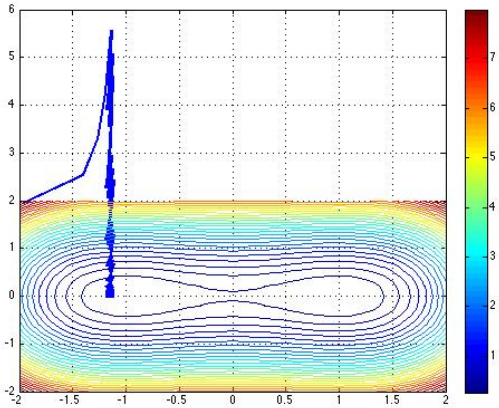
This algorithm behaves in the same way as the Natural GAD algorithm in how it responds to



(a) Initial points satisfying the Desirable Initial Conditions



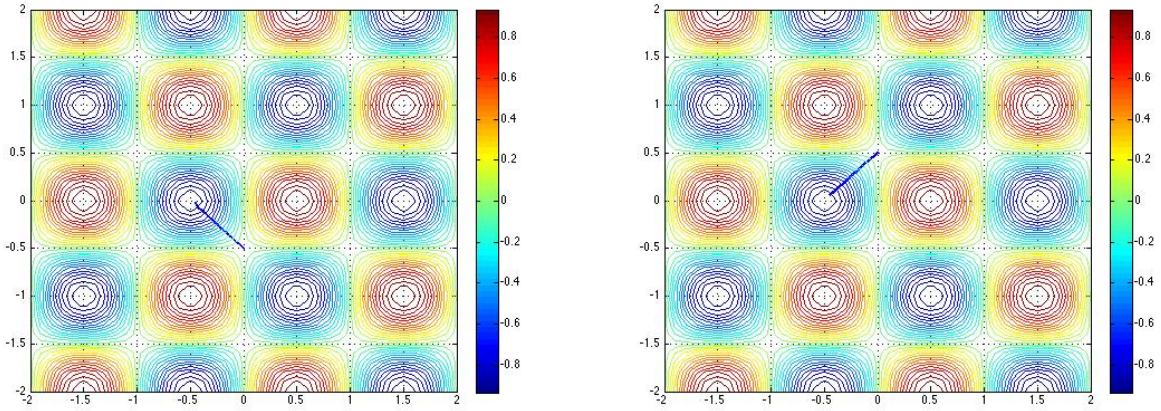
(b) Initial points NOT satisfying the Desirable Initial Conditions - outside GAD's basin of attraction



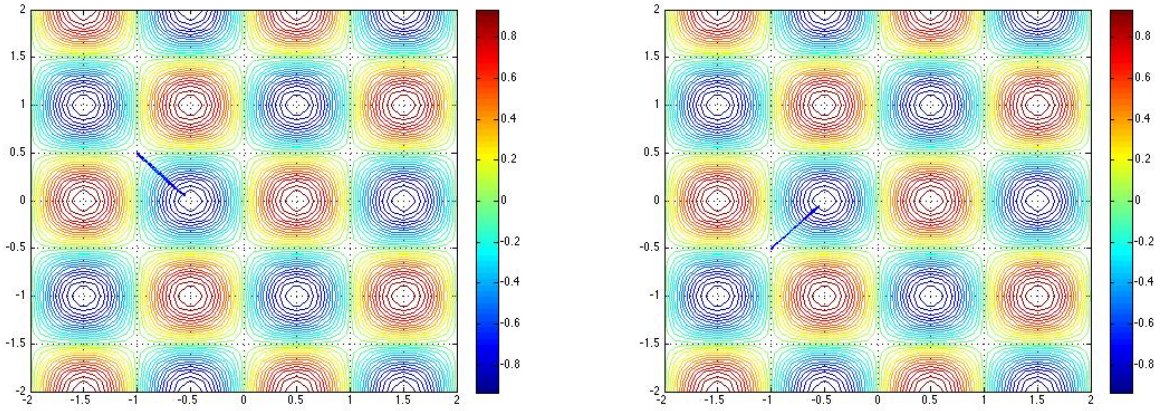
(c) Initial point much farther out and NOT satisfying the Desirable Initial Conditions

Figure 6.10.: *Solution Trajectories for the Rayleigh GAD Numerical Algorithm on* $V(x,y) = \frac{1}{4}(x^2-1)^2 + \frac{3}{2}y^2$.

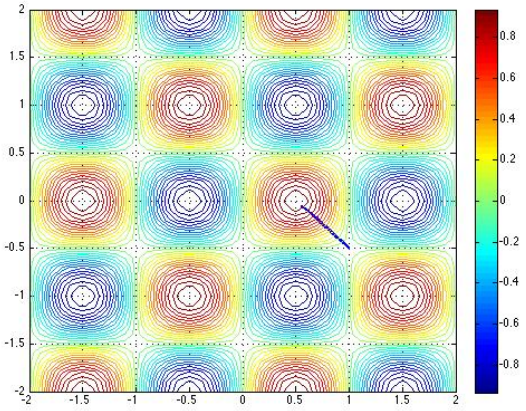
the set step-sizes. That is, if the step-size is small enough, the Rayleigh GAD algorithm will converge at the closest index-1 saddle point. On the other hand, if the step-size is not large enough, the algorithm will converge at the closest saddle point regardless of its degree of instability. We must note, however, that because of the aggressive nature of this algorithm the relative magnitudes



(a) Initial points near minimum satisfying the Desirable Initial Conditions



(b) Initial points near minimum satisfying the Desirable Initial Conditions



(c) Initial point near maximum satisfying the Desirable Initial Conditions

Figure 6.11.: *Solution Trajectories for the Rayleigh GAD Numerical Algorithm on* $V(x, y) = sin(x)cos(y)$.

of the step-sizes that might be deemed as "small enough" might be different for both algorithms.

6.6. Step-Size Strategy

As stated earlier, the true nature of GAD is a gradual process whose solution trajectory must first find a separatrix before it can converge toward an index-1 saddle point. Therefore, for a GAD-based algorithm to work as specified by the GAD mathematical formulation, the step-size used in traversing the trajectory must be very small. Any step-size that is larger might be too aggressive and cause the trajectory to cross the separatrix and not converge.

As a consequence of this, the matter of step-sizes becomes crucial in determining the speed convergence of the algorithms. To deal with this we need a step-size strategy that takes into account both the importance of convergence and the *speed* of convergence. In other words, a step size strategy is required in order to ensure that we get the fastest possible convergent GAD-based algorithm.

A majority of traditional step-size strategies often use function-value based minimisations. This approach will not work for GAD-based algorithms that we have devised because some steps along the solution trajectory may be ascent steps and others may be descent steps. To combat this, we need to use a more neutral objective metric to determine how far in a direction we should move per iteration.

Firstly, we realise that the overall goal of the is to find a fixed point with $\dot{\mathbf{x}} = 0$. We also know that

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v}\rangle \mathbf{v},$$

so we can define a residual-based construct that we want to be reduced at every iteration. Loosely speaking, a residual is the error in a result of an approximation of a function value. In this case, our goal is to get $\dot{\mathbf{x}} = 0$ which is equivalent to saying $||\dot{\mathbf{x}}|| = 0$. Therefore, we can define a residual function R as

$$R(\mathbf{x}) = ||\mathbf{F}(\mathbf{x}) - 2\langle \mathbf{F}(\mathbf{x}), \mathbf{v}\rangle \mathbf{v}||.$$

Since we now have a function we can minimise in order to determine the appropriate step-size in every iteration, we need to think about the constraints that we must set on this optimisation for it converge at the fastest rate. To do this, we remember that there are some iterations that the direction does not minimise the residual R and have a step size of 0. For these cases, we need to set the step-size to the minimal value which we will call α_{min}. For the cases where the direction in an iteration does manage to minimise the residual function then we need to restrict the upper bound of the step-size in order to prevent it from being "too aggressive". Let's call this upper bound α_{max}. For a direction $\mathbf{d}$ determined during an iteration, the resulting step-size strategy has the formulation

$$\begin{aligned} \underset{\alpha}{\text{minimise}} \quad & r(\alpha) = R(\mathbf{x} + \alpha \mathbf{d}) \\ \text{subject to} \quad & \alpha_{min} \leq \alpha \leq \alpha_{max}. \end{aligned} \tag{6.7}$$

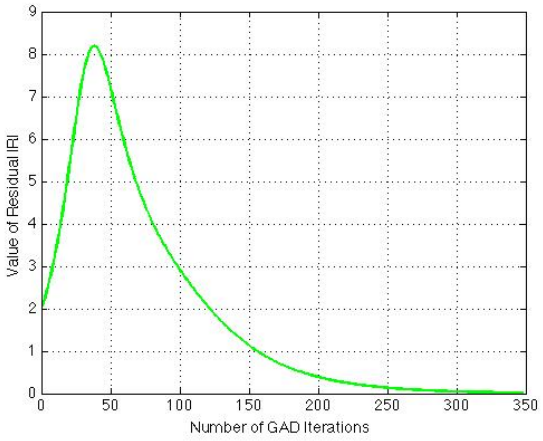
Experimentation of this strategy showed that its efficiency is dependent on the bounds α_{min} and α_{max}. As can be seen from the first two plots in figures 6.12 and 6.13, for both the Natural GAD and Rayleigh GAD algorithms, a fixed step-size yielded a smooth graph that eventually converged to having a residual of 0. This is how the GAD paradigm behaves in its essence. The initial section with the ascending graph shows the part where the algorithm is seeking out the separatrix. After

this separatrix is found we notice that the graph smoothly descends as the residual minimises until it becomes zero.

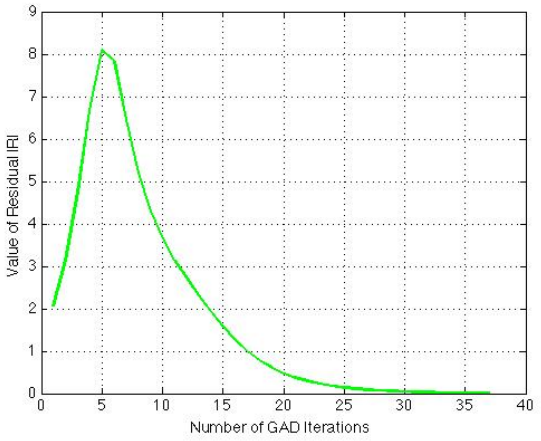
Comparing case with the smaller step-size and the one with the larger step-size show that the case with the larger step-size yielded a lower number of iterations. This shows that the correct parameterisation of the step-size can greatly improve the speed of convergence of the algorithms.

Testing the implementation of the step-size strategy showed interesting results too. From the respective plots in figures 6.12 and 6.13, it is apparent that a step-size with the appropriate parameterisation of the lower and upper bound step-sizes does further improve the speed of convergence. Another minor but intriguing observation is that when the step-size strategy is used the resulting plot is jagged by having interleaving sections with ascending and descending values of the residual. We know that the various sections with the ascending graph correspond to the search for a separatrix and the sections with the descending graph happen after a separatrix is found and an attempt to converge toward a saddle point is made. Since the search of a saddle point after a separatrix is found uses a much larger step-size it tends to be aggressive and the solution trajectory leaves the separatrix. The subsequent iterations then seek out a path back toward the separatrix which causes the residual to increase. This causes the jagged nature of the plots. This just illustrates the importance of tuning the correct upper and lower bounds of the step-size parameters in this strategy.

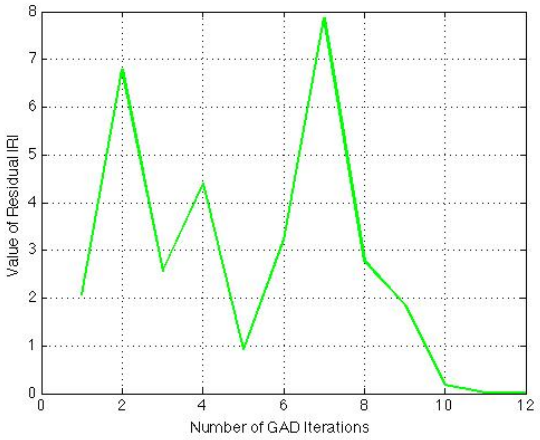
As a final note, we have used exact line-search principles to illustrate the step-size strategy approach for the GAD-based algorithms that we have formulated. However, inexact line searches can be used to implement this strategy but they must adhere to the principles outlined in this section.



(a) Natural GAD with smaller fixed step-size

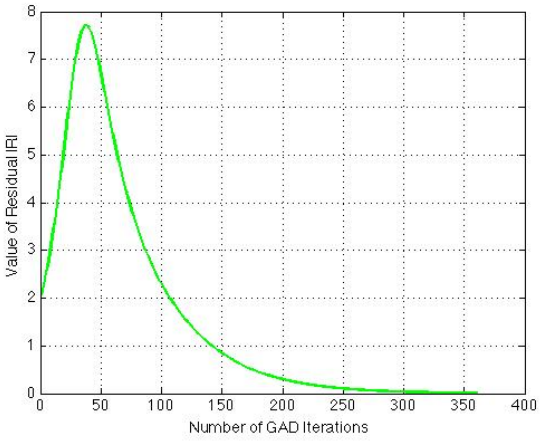


(b) Natural GAD with larger fixed step-size

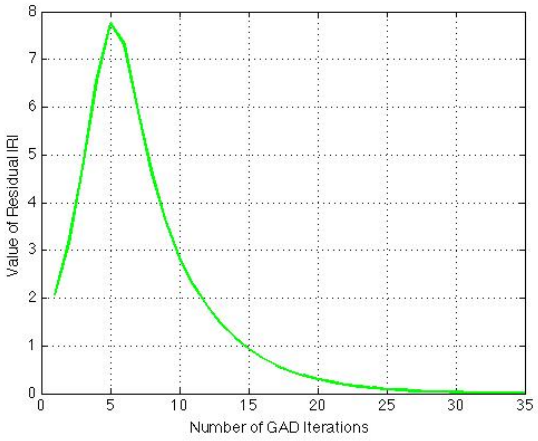


(c) Natural GAD using step-size strategy

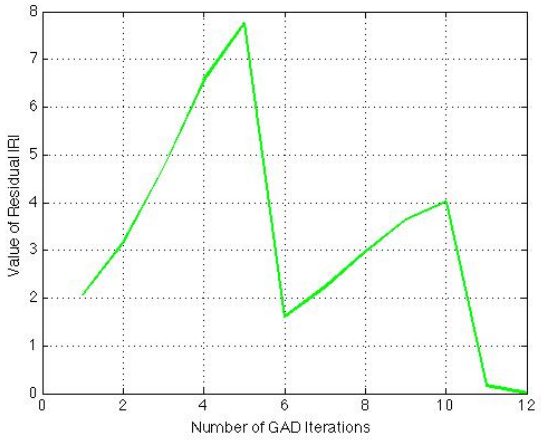
Figure 6.12.: *Residual vs. Iteration plots for Natural GAD to test effect of various sizes and the Step-size Strategy*



(a) Rayleigh GAD with smaller fixed step-size



(b) Rayleigh GAD with larger fixed step-size



(c) Rayleigh GAD using step-size strategy

Figure 6.13.: *Residual vs. Iteration plots for Rayleigh GAD to test effect of various sizes and the Step-size Strategy*

6.7. Convergence, Complexity and Implementation Issues

The *Basic* and *Natural* GAD algorithm shown in algorithm 4.4 has an optimistic *big-Oh* complexity of $O(n)$. This shows that in their naive form GAD-based algorithms are relatively cheap algorithms as E and Zhou pointed out in [7]. However, the reality is that when we add the calculations of the gradient, Hessian, eigenvalues, eigenvectors, optimal step-sizes, and Rayleigh minimisers in every iteration of the algorithm the story is different. The complexity of the implemented algorithm will be dependent on the methods used to calculate the aforementioned values. Because the algorithms formulated in temselves give free reign to the implementer on which methods to use, it is advisable to use the most efficient means possible for users who seek the most efficient algorithm.

On the matter of convergence, the error tolerance and step-size strategy that are used determine the overall convergence-centric aspects of the algorithm.

6.8. Conclusion

In this chapter we designed stable and convergent numerical algorithms based on the Gentlest Ascent Dynamics (GAD) to find index-1 saddle points and formulated the *Desirable Initial Conditions* required to always achieve this. We started by constructing a *basic* and *natural* GAD-based numerical algorithm that is initialised by an initial point and direction. However, this algorithm suffers from convergence based on initial conditions where only small fraction of the whole configurational space for the initial directions lead to stable dynamics converging at index-1 saddle points.

To solve this problem, two distinct algorithms were proposed as improvements on this basic one. The first algorithm is inspired by observations in molecular physics where energy changes during deformation processes proceed via saddle points. Using notions from dynamical systems theory and molecular physics we formulated two impositions on the *initial conditions* that are *desirable* to ensure the GAD algorithms always converge toward a saddle point. The conditions are:

1. The initial point $\mathbf{x_0}$ must be close to at least two local fixed point or close to a saddle point;
2. The initial direction $\mathbf{v_0}$ must correspond to the eigenvector with the smallest eigenvalue of the Hessian at the point $\mathbf{x_0}$ close to the local fixed point.

The second algorithm is a modification of the first by incorporating *Rayleigh optimisation* into the algorithm to ensure quick relaxation of the algorithm to the desired directions and faster convergence onto the separatrix. This algorithm has the advantage of never being unstable but still needs to satisfy the *Desirable Initial Conditions* in order to find a saddle point.

7

Finding All the Saddle Points On Surfaces Using the GAD Numerical Algorithm

7.1. Introduction

In this chapter we take the next step in our quest to develop a global optimisation algorithm. At this point, we have a numerical algorithm that is able to find saddle points and in this chapter we will further improve on this algorithm such that, given enough time, it can find all the saddle points on a generic surface.

7.2. Problem Formulation

In order to develop an algorithm that finds all the saddle points on a generic n-dimensional surface we need to find an abstract way to formulate this problem. This is to make our task easier and more susceptible to treatment by conventional computer science and mathematical formalisation.

For the purpose of this discussion, it will be assumed that we have a multimodal surface with multiple maxima and minima, and the goal is to find all the saddle points on this surface. Ideally, what we want is the ability to exhaustively explore the surface by having an algorithm starting at one point and then finding all the saddle points. We can think of this as a classical graph exploration problem where goal is to be able to visit all nodes on a graph.

One way to abstract a graph out of this problem is to use topologically-inspired notions where basins and saddle points are connected. On a surface with multiple minima the space between the minima will have a point where the surface folds in two opposite directions. As already described in previous chapters, that is how a saddle point is formed. A good real-life example of this is a mountain pass. Therefore, using this line of reasoning, the nodes in this case will be the minima and the saddle points where the a saddle point is connected to an adjacent minimum point and vice

versa. Figure 7.1 below illustrates an example of an incomplete graph representing an abstraction of the exploration of a multidimensional surface that we have described.

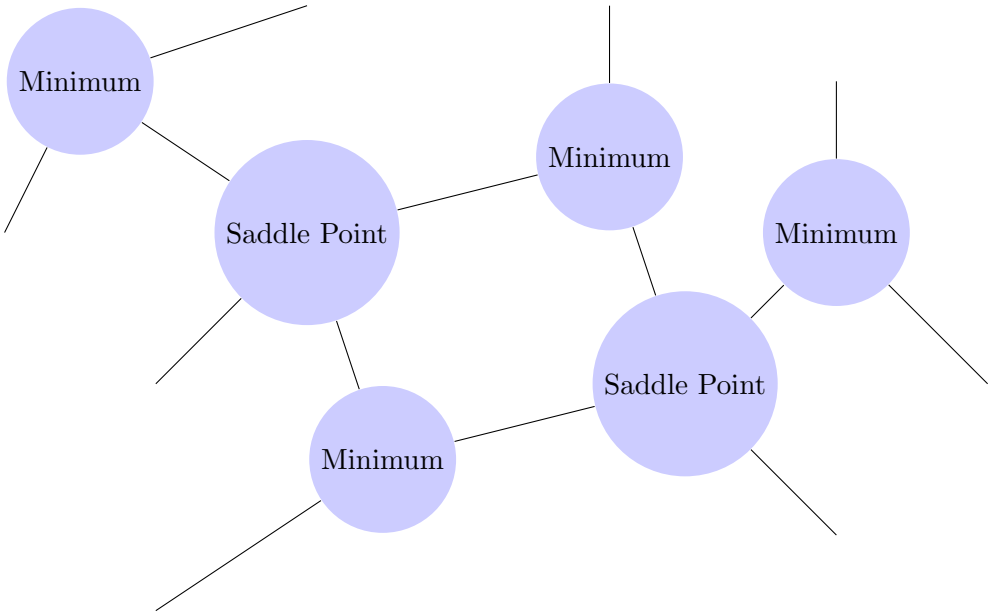


Figure 7.1.: *Example of an incomplete exploration graph abstracted from a multidimensional surface.*

In summary, the problem at hand is to explore an unknown graph with nodes that are saddle points and minima. As a result, solving this problem involves building the graph while it is being explored.

7.3. The GAD "Chaining" Algorithm

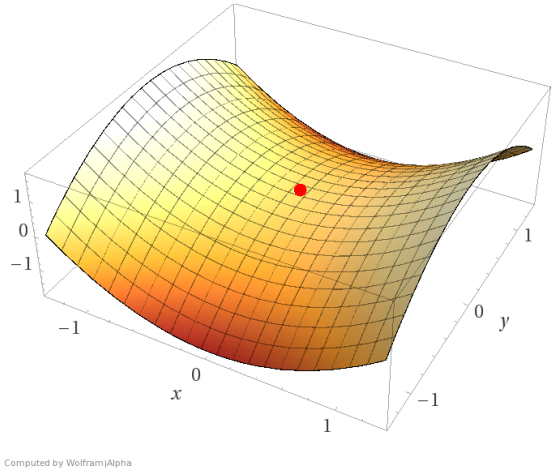
Since we have the decided to abstract the saddle-point-finding problem using a graph exploration formalisation, developing an algorithm is a less daunting task. Why is this so? It's because we can pretend we are exploring a simple graph rather than a complex and intimidating multidimensional surface.

The easiest way to go about the problem is to use the Breadth First Search (BFS) inspired approach where we start at a random point and first find a saddle point using the GAD algorithm. From the saddle point we can then find all the minima associated with this saddle point using a simple local minimisation algorithm and so on.

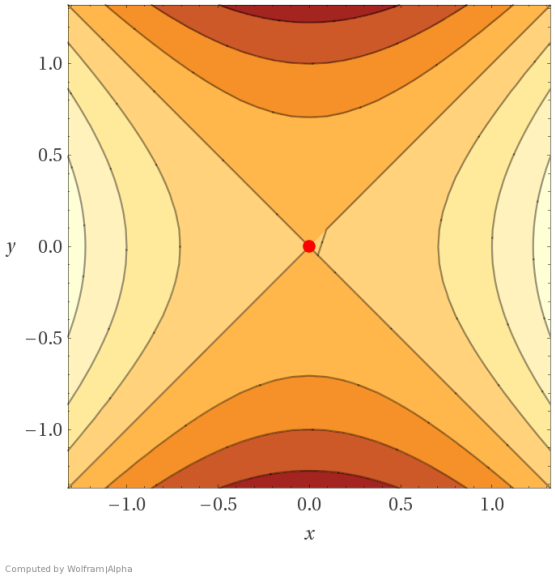
Of course, this is not as simple as it sounds because there are many things to worry about. Firstly, although this fact is trivial, the minima and saddle points are fixed points so both the GAD and local minimisation algorithms may get stuck there. Therefore, we need a way to perturb the these points in order to generate new starting points for the algorithm to proceed. For example, after finding the first saddle point, how do we determine where to start the local minimisation algorithms in order that they can find all the associated minima? Also, at the minima, how can we determine where to start the GAD algorithm so that we can find the next saddle points? The answer here is to slightly perturb these fixed points in suitable directions to generate new points. Using this

paradigm, we can explore the whole surface.

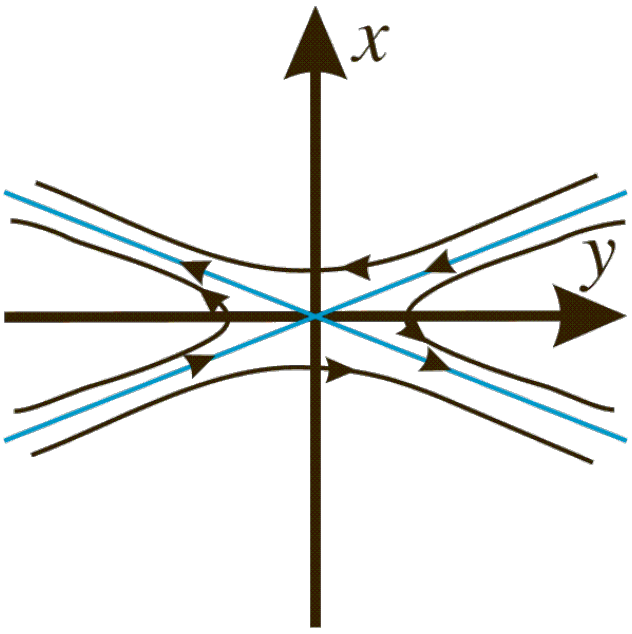
So what are these suitable directions? How do we find them? In order answer these questions we need to get an intuitive understanding of this problem. To do so we will use a 2 dimensional example shown in figure 7.2.



(a) Graph of function $z = x^2 - y^2$



(b) Contour plot of function $z = x^2 - y^2$



(c) Phase plot of a generic 2 dimensional saddle point (Source: Wikimedia Commons)

Figure 7.2.: *A graph and contour plot of $z = x^2 - y^2$, and a generic phase plot of a 2 dimensional saddle point.*

Figure 7.2a shows a plot of the function $z = x^2 - y^2$ which has a saddle point at the origin marked by the red dot. From the diagram it is easy to notice that the surface folds upwards in one direction and downwards in another. This is also evident from the corresponding contour plot in figure 7.2b. Therefore, it is logical to surmise that the direction to look for the minima is the one corresponding

to the to the part where the surface folds downwards and the function value decreases.

So how do we find this direction where the function value is decreasing and the surface is folded downwards? To generalise the approach of how we'll answer this question let us consider a phase plot of a generic saddle point shown in figure 7.2c. Take note that this is not the phase plot of the function $z = x^2 - y^2$. In this plot, we notice several interesting things. Firstly, this point is an attractor for particular trajectories and at the same time repeller for others. Secondly, for each attracting direction the negation of this direction is also an attracting direction; this also holds for the repelling directions. With respect to the saddle point, the trajectories heading away from the saddle point are heading towards a sink or, more commonly, a minimum. This direction is unstable with respect to the saddle point. From stability theory, we know that the unstable directions are the eigenvectors corresponding to the negative eigenvalues of the Hessian. At an index-1 saddle point in this case, we are guaranteed that one eigenvalue will be negative. The positive and negative eigenvector corresponding to this negative eigenvalue is what we need to follow to find a minimum. Therefore, this is the direction we need to perturb in order to get a good starting point for a local minimisation algorithm that finds a minimum.

At a minimum, the perturbation scheme is simpler since we attempt to distribute the new staring points for the GAD algorithm evenly about the point. Therefore, the best way to do this most effectively is to do it in the positive and negative directions of all the eigenvectors of the Hessian.

The resulting algorithm is referred to as GAD "Chaining" because intermittent runs of the GAD algorithm are "chained" together using local minimisation algorithms. The algorithm is shown in figure 7.1.

This algorithm starts at a point $\mathbf{x_k}$ and keeps track of starting points using a queue $\mathbf{Q}$. We first find the nearest saddle point $\mathbf{x_{SP}}$ using the GAD algorithm and then evaluate the unstable eigenvectors of the Hessian there which are simply the directions corresponding to the negative eigenvalues. We then find new starting points for a local minimisation algorithm by perturbing in the positive and negative unstable directions. From these points we start a local minimisation algorithm to find the associated minima. At the minima, we get the complete eigen-space of the Hessian and use it to find the new starting points for the GAD algorithm in the next iteration and we add them to the queue $\mathbf{Q}$.

Of course, we need more graph-theoretic data structures to keep track of what is going on. We want to be able to know what stationary points we have found so far and their adjacency. As a result, we will need the following data structures:

1. **Node List**: Keeps track of what stationary points that have been found so far.
2. **Node Type List**: Keeps track of the types of the stationary points — whether they minima or saddle points — that have been found so far.
3. **Adjacency Matrix**: Keeps track of what stationary points are adjacent to each other.

Many other data structures can be incorporated into the algorithm depending on what the user wants to keep track of. For example, a user may want to also keep track of the function value at the

Algorithm 7.1 GAD 'Chaining' Algorithm

Require: Node List **N**
Require: Node-type List **NT**
Require: Adjacency List **A**
Require: Queue **Q**
Require: Start point $\mathbf{x_0}$
Require: Add $\mathbf{x_0}$ to **Q**

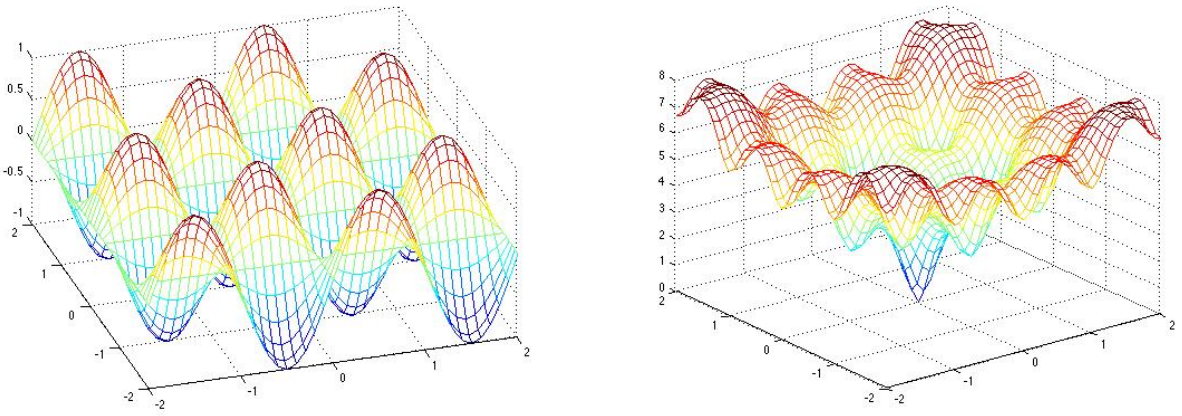
1: **loop**
Get item at front of **Q** as start point $\mathbf{x_k}$ of GAD;
Get nearest saddle point $\mathbf{x_{SP}}$;
Add saddle point to Node List **N** and enter the node-type in **NT**;
Get Hessian at saddle point;
Find directions corresponding to negative eigenvalues of Hessian — unstable directions;
Obtain start points obtained after "perturbing" the saddle points in positive and negative unstable directions;
Use starting points for local minimisation to get minima;
Add minima to Node List **N** and enter the node-types in **NT**;
Add adjacency to Adjacency List **A**;
Get Hessian at the minima;
Find eigenspace of the minima;
Find new start points obtained after "perturbing" the minima in positive and negative directions of the all directions in eigenspace;
Add new start points to **Q**;

stationary point. To do this one simply adds a new data structure to keep track of this information.

So does this algorithm work? For illustration, the algorithm was tested on the following 2 dimensional non-linear multi-modal functions:

1. **Sinusodal Function**: $f(x,y) = sin(\pi x)cos(\pi y)$
2. **Ackley function**: $f(x,y) = -20e^{(-0.2\sqrt{(\frac{1}{2}(x^2+y^2))})} - e^{(\frac{1}{2}(cos(2\pi y)+cos(2\pi y)))} + 20 + e$

The plots of these functions are shown in figure 7.3.



(a) Sinusodal Function (b) Ackley function

Figure 7.3.: *Graphs of test functions for the GAD "Chaining" Algorithm.*

The solution trajectories resulting from the runs of the GAD "Chaining" Algorithm on the functions is shown in figures 7.4 and 7.5. From the illustrations, it is easy to see that the algorithm works well in exploring a surface searching for minima and index-1 saddle points. It must be noted that the searching sequence is determined by the order of the starting points in the queue **Q**. As a result of this, in order to modify the way that the algorithm explores the surface, one must modify the way starting points are selected from **Q**.

This algorithm, given enough time, can explore an entire surface. In fact, if the surface of exploration is constrained and bounded, then it is guaranteed to execute and explore that whole surface in finite time. We will now proceed to discuss the practical and implementation aspects of this algorithm in the next section.

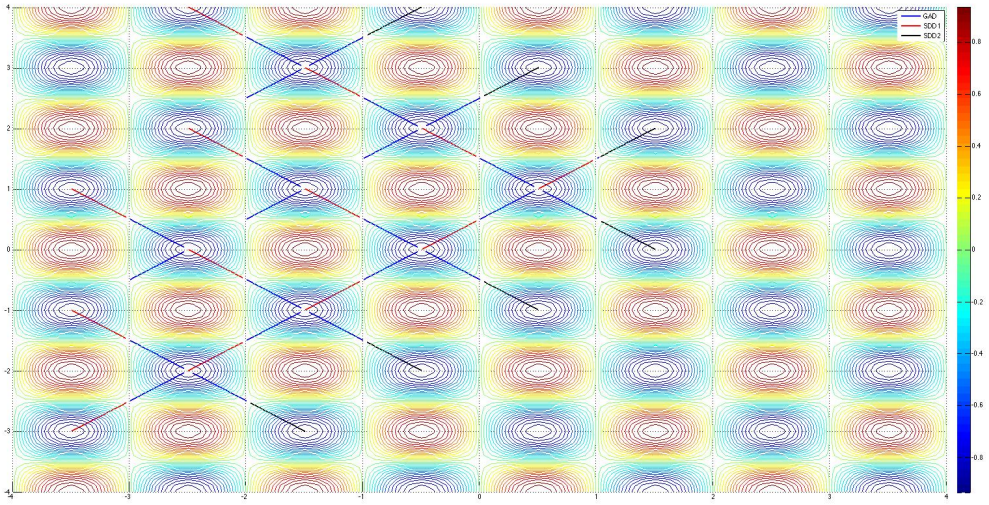


Figure 7.4.: *Solution Trajectories of the GAD "Chaining" Algorithm on the Sinusodal Function.*

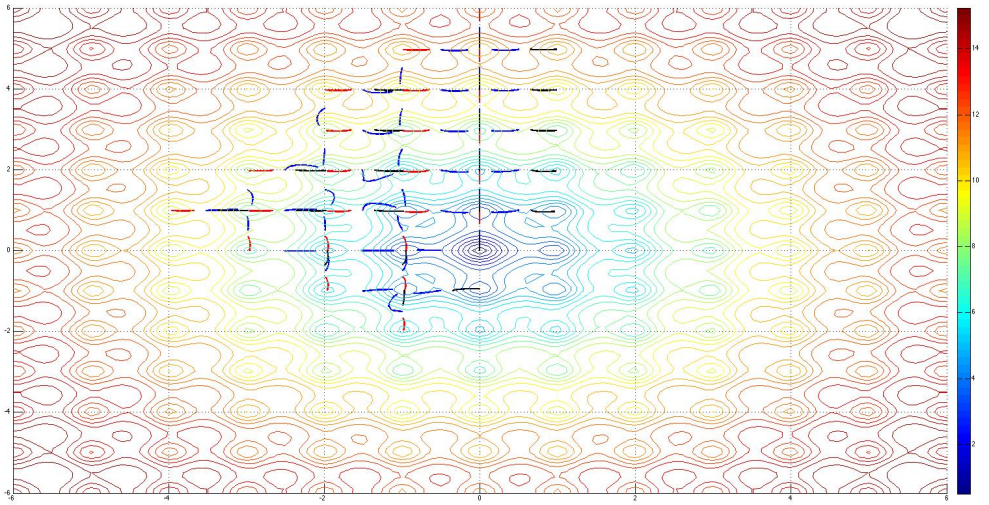


Figure 7.5.: *Solution Trajectories of the GAD "Chaining" Algorithm on the Ackley Function.*

7.4. Complexity and Implementation Issues

The algorithm proposed in this chapter involves keeping track of numerous amount of information as its execution progresses. Because this algorithm is is explorative in nature, it is important to also include a method for checking if an item already exists in a data structure. This way, a check is made to see if an item already exists in the data structure before inserting it. Doing this prevents the data structures from containing duplicates and consequently makes it easier to manipulate and make use of them.

The GAD "chaining" algorithm has 3 main ingredients: the GAD algorithm, a local minimisation algorithm and a perturbation scheme. As we have already covered the implementation issues in both the GAD algorithm and local optimisation algorithm in previous chapters, we will only look at the implementation issues relating to the perturbation scheme in this section.

Perturbation simply displaces a point with small noise in a particular direction. Therefore, in the simple form, we can define a perturbation scheme as shown below.

Definition 7.4.1. (Perturbation Scheme)
Given a point $\mathbf{x} \in \mathbb{R}^n$ *with a displacement direction* $\mathbf{d} \in \mathbb{R}^n$ *and noise constant* ϵ*, then the Perturbation Scheme* $\mathbf{P}$ *that generates a new point* $\mathbf{P}(\mathbf{x}, \mathbf{d}, \epsilon) \in \mathbb{R}^n$ *is given by*

$$\mathbf{P}(\mathbf{x}, \mathbf{d}, \epsilon) = \mathbf{x} + \epsilon\mathbf{d} \tag{7.1}$$

The above definition is simple in itself because it succinctly describes what we want perturbation to achieve. The value of the noise constant ϵ is problem dependent but it is recommended that it be made small enough to allow the generated start point be displaced enough from the original critical point to allow the succeeding algorithm to converge at the next critical point. This constant may be very small for some functions and larger in others. In most cases, it is important to consider the truncation errors associated with the numerical calculations that may indadvertedly cause the algorithm to be trapped by the original critical point.

As a final note, we make a comment on how to prevent the trajectory from visiting the same saddle point more than once within successive iterations. There are many ways to do this but the recommended method is to use a penalty function where the algorithm keeps track of the parent saddle point x^*_{sp} in the preceding iteration. Using this saddle point the algorithm ensures that GAD trajectories emanating from points perturbed from the minimisers are penalised by a penalty function $P(x)$ that is defined in terms of the distance from the current point to the saddle point x^*_{sp}.

7.5. Conclusion

In this chapter we have designed a numerical algorithm that is able to explore a generic surface and find all the saddle points on it given enough time. This algorithm had been called GAD "Chaining" because it interleaves intermittent runs of the GAD algorithm with local minimisation runs. This algorithm starts at a point x_k and finds the nearest saddle point from which various local minimisation runs are used to find all the minima associated with the saddle point, and so on. A

Morse theoretic approach is used to deal with the issue of ensuring all the associated minima are found.

The problem as a whole is formulated as a graph-theoretic exploration problem where the goal is to explore an unknown graph whose nodes are saddle points and minima. As a result, solving this problem involves building the graph while it is being explored and the solution trajectories follow a breadth first search paradigm.

8

Global Optimisation using the Gentlest Ascent Dynamics and the Global Descent Policy

8.1. Introduction

At this point we have developed an effective algorithm that is able to explore and find all the saddle points on a function. This algorithm, in itself, is very important and can lend itself to the many uses in areas where one may need to find saddle points. A good example of such an area is molecular physics. In this area, atomic interactions have energy levels that traverse via saddle points of the associated potential energy surfaces.

However, the ability find saddle points on a function has an even a deeper implication. Since saddle points are located at points where the hyper-surfaces fold in different directions, they are points that give enough information about a function's curvature and critical point distribution in that vicinity. In fact, it suffices to say that if one is able to find all the saddle points on a function then one knows how the function behaves everywhere.

In this chapter we want to use our newly developed saddle point finding ability to find the global optimum of an objective function.

8.2. Problem Formulation and Solution Strategy

It is obvious that if we let the GAD "Chaining" algorithm developed in the previous chapter to run indefinitely it will explore the whole function and it will develop a complete graph representation of it. One can modify this algorithm to always keep track of the best minimiser found so far and by the time the algorithm terminates it can return the global minimiser. However, this approach

has the drawback that it can run forever on an unbounded function and is thus computationally infeasible. In other words, we want an algorithm that converges to the global minimum as fast as possible in a practical time frame.

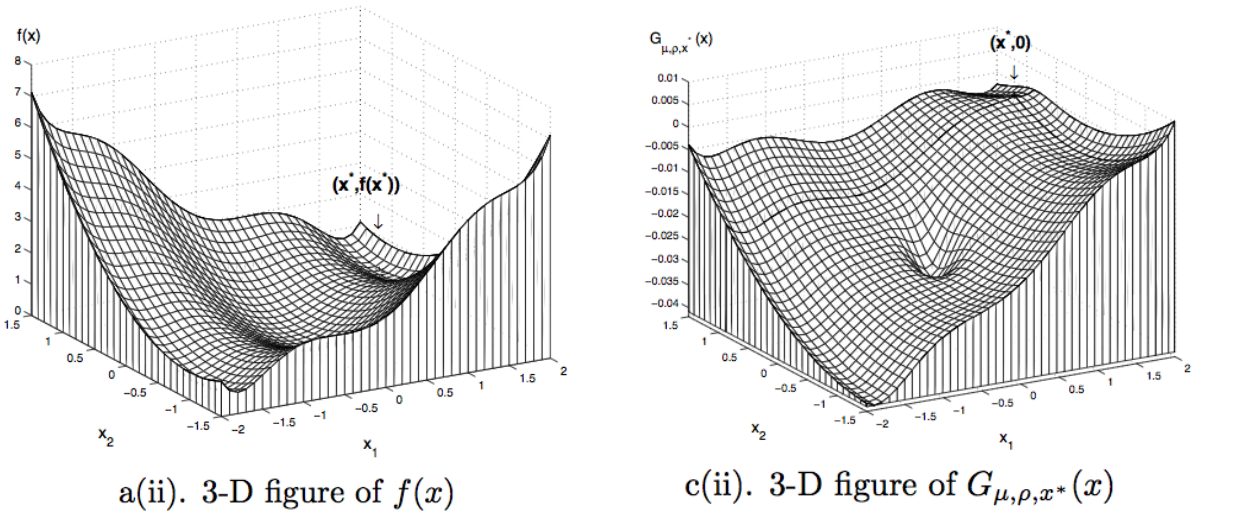
So how can we ensure that we find a globally optimal point with the saddle point finding strategy?

The most obvious approach is to ensure that we always find better minimisers at every iteration. Let us call this strategy the **Global Descent Policy**. In this chapter we develop two methods that follow this strategy. Both methods make use of saddle points as transitional points to find the global minimiser of a function. We will look at each method in turn.

8.3. Global Optimisation Approach 1: Using Global Descent Functions at Saddle Points

We currently have the following tools in our arsenal: the GAD algorithm and the global descent policy. If we were somehow able to explore a surface in such a way that we always find a saddle point located at a lower elevation at every step then we would indirectly be finding better minimisers.

One of the tools we can make use of is the global descent function (from [5]) that we described in section 4.5. As a quick reminder, a global descent function G of an objective function f at a local minimiser x^* is a type of filled function where this point x^* exists as a maximiser on G and that if there is a better minimiser x^{**} of f then it will exist on G along a decent direction line from the maximiser x^*. Figure 8.1 below illustrates this on the three-hump camelback function. In other words, the global descent function G can be used to find a new starting point for a local minimisation algorithm to a better minimiser.



(a) Function $f(x)$ with minimiser at $(x^*, f(x^*))$ (b) Global Descent Function $G(x)$ with maximiser at $(x^*, 0)$

Figure 8.1.: *Illustration of what a Global Descent Function is on a the three-hump camelback function (Source: [5]).*

So we can ask an interesting question: Can we use a global descent function at a saddle point x_{sp} of a function f to find a better saddle point? One may ask why we want to do this and the answer is quite straightforward. As stated before, a lower saddle point implies a lower minimiser. Therefore, if we interleave runs of the GAD algorithm with uses of the global descent function at the saddle points we are indirectly be seeking out a better minimisers at every iteration.

However, we still need to think about how to make the global descent function to work for saddle points because its theoretical construction is based on minimisers. We know that a saddle point partially behaves like a minimiser by being a stationary point (since $\nabla f(x_{sp}) = 0$) and that it is stable in some directions around it. Therefore, if we restrict the global descent function to work on a constrained subspace of the area around the saddle point where it behaves like a minimiser then we can solve this problem.

Firstly, the implication of the global descent function at the saddle point working as we want it to is that we will have a way of finding points at a lower basin than our current saddle point. Any of these new points could easily be the minimiser connected to the current saddle point because it it at a lower basin by definition. We don't want this. What we want is a better saddle point that is farther away from the closest minimisers. How do we ensure this? We can make the descent step that we take on the global descent function G be slightly larger. That way, we unsure that we escape the neighbourhood of the current saddle point and search farther away. This condition of a larger step-size is a disparity from the original algorithm in [5].

Secondly, another change we can make to the original algorithm is to deal with the fact that a saddle point behaves like a minimiser only in particular directions. This set is formally called the *stable manifold* of the saddle point. Therefore, searching on the global descent function should be constrained to those directions where it behaves like a minimiser - that is, its *stable manifold*. To ensure this, we can search in the directions corresponding to the positive eigenvalues of the Hessian at the saddle point. This may seem odd at first thought because the stable manifold of a saddle point on a function corresponds to the part of the function in the vicinity of the saddle point where its value is increasing (folded upwards). However, one needs to remember that this part that is folded upwards on the original function f will be folded downwards on the global descent function G, and it is this descent search space that we want to access.

Thirdly, an important thing to note is that we are searching for a global minimiser. Therefore, at every step that we find a better saddle point, we should find the better minimiser associated with that saddle point. As a result, the algorithm will generate a pair of critical points (a saddle point and a minimiser) at every iteration.

The resulting algorithm is shown in algorithm 8.1; for easier reading, a more compact version of the algorithm is shown in algorithm 8.2.

So what is going on in algorithm 8.1? As can be seen by the compact version in algorithm 8.2, we start at a point $\mathbf{x_k}$ and use GAD to find the closest saddle point $\mathbf{x^*_{sp}}$ and a local minimisation algorithm to find the closest minimiser $\mathbf{x}^{**}$. From here, our surface exploration proceeds by forming

Algorithm 8.1 Global Optimisation using GAD and Global Descent Function

Require: Critical Point Storage List **L**

Phase 0: Initialisation

(a) Choose a function v_μ satisfying conditions (V1) and (V2).

(b) Choose $\rho_{ini} > 0$ (the initial value of ρ), $0 < \mu_{ini} < 1$ (the initial value of μ), $\mu_L > 0$ (the lower bound of ρ), $0 < \hat{\rho} < 1$(a fraction for the reduction of ρ), $0 < \hat{\rho} < 1$ (a fraction for the reduction of μ), $\epsilon > 0$ (the radius of a small neighbourhood of saddle point x^*_{sp}), $\kappa > 0$ (a small tolerance), and $\lambda_U > 0$ (the maximum step-size for a line search).

(c) Set $\rho := \rho_{ini}$ and $\mu := \mu_{ini}$.

(d) Choose/generate an initial point $x_{ini} \in \mathbb{X}$ for problem

$$\min \{f(x) : x \in \mathbb{X}\}$$

and set the current iterative point $x_{cur} := x_{ini}$ and the iteration count $N_{iter} := -1$.

(e) Perform the local search in Phase 1.

Phase 1: Local Search for Closest Saddle Point and Minimiser

(a) Starting from x_{cur}, use any local minimisation method to search for a local minimiser x^{**} of f over $\mathbb{X}$ and the GAD algorithm to find the closest saddle point x_{sp}. Update the current saddle point $x^*_{sp} := x_{sp}$ and the iteration count $N_{iter} := N_{iter} + 1$.

(b) Store current minimiser x^{**} and saddle point x^*_{sp} in list L.

(c) If x^{**} is NOT better than previous minimisers then **exit**;

(d) Perform the global search in Phase 2.

Phase 2: Global Search

(a) Find stable directions at x^*_{sp}. Generate a set of m initial points distributed around saddle point x^*_{sp} but outside the immediate neighbourhood of $\{x^*_{sp}$: $x^{ini}_{(i)} \in \mathbb{X} \backslash N_\epsilon(x^*_{sp}) : i = 1, 2, ..., m\}$ perturbed in stable directions by step-size ϵ (radius of a small neighbourhood). Set $i := 1$.

(b) Set the current iterative point $x_{cur} := x^{ini}_{(i)}$

(c) If $f(x_{cur}) < f(x^*)$, then perform the local search in Phase 1.

(d) If

$$||\nabla G_{\mu,\rho,x^*_{sp}}(x_{cur})|| < \kappa \text{ and } (x_{cur} - x^*)^\top \nabla G_{\mu,\rho,x^*_{sp}}(x_{cur}) \geq 0,$$

then choose a positive integer l such that $\mu_l := \mu^l \mu$ and

$$||\nabla G_{\mu,\rho,x^*_{sp}}(x_{cur})|| \geq \kappa \text{ or } (x_{cur} - x^*)^\top \nabla G_{\mu,\rho,x^*_{sp}}(x_{cur}) < 0.$$

Update $\mu := \mu_l$.

(e) Choose a descent direction D of $G_{\mu,\rho,x^*_{sp}}$ at x_{cur}. Find a new x along descent direction D by a line search method such that $G_{\mu,\rho,x^*_{sp}}$ can reduce to a certain extent and the step-size of the line search $\lambda \leq \lambda_U$ (this step-size must be large enough to escape the basin with the adjacent minimiser); then set $x_{cur} := x$ and go to Phase 2(c). However, if x attains the boundary of $\mathbb{X}$ during minimisation, then go to Phase 2(f);

(f) Set $i := i + 1$. If $i \leq m$, then go to Phase 2(b);

(g) Set $\rho := \hat{\rho}\rho$ and reset $\mu := \mu_{ini}$. If $\rho \geq \rho_l$, then go to Phase 2(a). Otherwise, the algorithm is incapable of finding a minimiser of f better than the current local minimiser x^* starting from the initial points $x_{(i)} : i = 1, 2, ..., m$. The algorithm stops, and x^* is taken as a (putative) global minimiser.

a global descent function at the saddle point $\mathbf{x^*_{sp}}$, seeking out a better starting point located in a lower basin in stable descent directions at that saddle point, and then finding the closest saddle point and minimiser to this starting point, and so on. This algorithm does indeed seek out better

minimisers at every iteration. Concerning termination conditions, this algorithm terminates in two cases:

1. When the global descent problem-dependent parameters hit their lower bounds (as specified in [5]),
2. When no better minimiser can be found.

As one can imagine, the dependence on problem specific parameters could lead to situations where the algorithm may terminate with a minimiser that may not necessarily be a true global optimiser. We will address this issue in section 8.5.

Algorithm 8.2 Compact Global Optimisation using GAD and Global Descent Function

Require: Critical Point Storage List $\mathbf{L}$
Require: Starting point $\mathbf{x_0}$
Require: $k = 0$
Require: Initialise start point to $\mathbf{x_k}$
Require: Initialise Global Descent Function parameters
1: **loop**
  Get nearest saddle point $\mathbf{x^*_{sp}}$ to starting point $\mathbf{x_k}$;
  Get nearest minimum point $\mathbf{x}_{**}$ to starting point $\mathbf{x_k}$;
2: **if** $\mathbf{x^*_{sp}}$ and $\mathbf{x}^{**}$ are the lowest lying saddle point and minimiser found so far **then**
  Store the current saddle point $\mathbf{x^*_{sp}}$ and minimiser $\mathbf{x}^{**}$ in list $\mathbf{L}$;
  Use the global descent function at $\mathbf{x^*_{sp}}$ and search the *stable manifold* of $\mathbf{x^*_{sp}}$ to get a new starting point at a lower basin;
  Set $\mathbf{x_{k+1}}$ as new better starting point;
  $k = k + 1$;
3: **else**
  Take current saddle point x^*_{sp} and minimiser x^{**} as the lowest lying critical points;
  Take minimiser as putative global minimiser;
  break;

To quickly summarise, as stated before, the key to making this algorithm work is to ensure that the step-size of the line search on the Global Descent algorithm is large enough for the next point to be far enough from original saddle point. This is to ensure that we escape the saddle point's GAD basin of attraction. Secondly, searching on the global descent function defined at a saddle point should be constrained to those directions where it behaves like a minimiser - that is, its *stable manifold*. Both these conditions are a disparity from the original algorithm.

8.4. Global Optimisation Approach 2: Using GAD "Chaining" and Unbounded Line Searches

The algorithm described in the previous section relies on global descent functions which have problem they need the correct specification of problem-dependent parameters in order to work properly. These parameters may be difficult to estimate and, in some cases, their appraisement may be as hard as the original problem. In this section we devise a simpler method that achieves the same goal as the global descent functions above.

Of course, as a prerequisite, we want our approach to use saddle points and follow the Global Descent Policy. As a result, the method we are formulating will be a result of modifying the GAD "Chaining" Algorithm to follow the Global Descent Policy.

To recap, the GAD "Chaining" Algorithm iteratively finds a saddle point and the minimisers connected to this saddle point. Therefore, if we can find a way of being able to identify a lower basin from a saddle point without resorting to the global descent function then we would have a second approach. Fortunately, this is possible.

We know, from definition, that a saddle point is a point where a surface folds in different directions. As stated earlier in the chapter, they are points that give enough information about a function's curvature and critical point distribution in that vicinity.

The suggested modification to the GAD "Chaining" is as follows: when we reach a saddle point, our goal is to find another saddle point that is in a lower basin. To do this, we perform a line search in the direction of the unstable manifold at the point in order to find the step-size that minimises the function the most in that direction. To reiterate, the direction in this case is that of the negative eigenvalue; that is, the direction where the minimum connected to this saddle point is. So, for a function f with unstable direction d at a saddle point x_{sp}, we try to find the step-size λ that minimises the function the most in direction d we do

$$\min\, g(\lambda) = \min\, f(x_{sp} + \lambda \cdot d). \tag{8.1}$$

Since adjacent minima will always be along this line with respect to the saddle point we will always be guaranteed to get the lowest basin in that direction. The worst possible case is getting the minimum adjacent to the saddle point and the best case is getting a point lower than the minimum next to the current saddle point. In either case we will be finding better minimisers because minima within a certain radius are likely to be along this direction and this find the best one. From this lowest point, we can find the closest minimum and in turn find the next saddle point, and so on. The algorithm showing the described algorithm is shown in algorithm 8.3.

It is worth pointing out that usual line searches place a constraint on the step-size λ such as $0 \leq \lambda \leq N$, where usually $N = 1$. However, the line search we do in algorithm 8.3 above and shown in equation 8.1 is a one dimensional unconstrained minimisation with $\lambda \in \mathbb{R}$. This is why the line search in the algorithm is referred to as *unbounded*.

Algorithm 8.3 Global Optimisation using GAD "Chaining" and Unbounded Line Search Algorithm

Require: Objective function f
Require: Node List **N**
Require: Node-type List **NT**
Require: Queue **Q**
Require: Start point $\mathbf{x_0}$
Require: Add $\mathbf{x_0}$ to **Q**

1: **loop**
Get item at front of **Q** as start point $\mathbf{x_k}$ of GAD;
Get nearest saddle point $\mathbf{x_{sp}}$;
Add saddle point to Node List **N**;
Get Hessian at saddle point;
Find directions corresponding to negative eigenvalues of Hessian — unstable directions, d;
Do a line search in unstable directions and find step-size λ^* that minimises function value the most;

$$\min g(\lambda) = \min f(x_{sp} + \lambda \cdot d).$$

Obtain start points obtained after "perturbing" the saddle points in positive and negative unstable directions with step-size from previous step;
Use starting points for local minimisation to get minima;
2: **if** current minimisers are better minimisers than other minimisers found so far **then**
Add minima to Node List **N** and enter the node-types in **NT**;
Get Hessian at the minima;
Find eigenspace of the minima;
Find new start points obtained after "perturbing" the minima in positive and negative directions of the all directions in eigenspace;
Add new start points to **Q**;
3: **else**
Take lowest lying minimisers as putative global minimisers;
break;

8.5. Convergence, Implemenation and Complexity Issues

Both variants of the global optimisation algorithms described in the previous sections terminate when they cannot find a better minimiser from the current point of an iteration. Therefore, all the global optimisers they find are putative. For the cases with multiple global minimisers, different initialisations of the algorithm can be done.

For the first algorithm which uses the global descent function, another possible reason for termination may be due to the problem dependent parameters reaching their lower bounds. The global descent function works well with a predicator-corrector approach to fine tune the parameters. Therefore, it is possible that at termination better minimisers could have been missed. An effective way to deal with this when implementing the algorithms shown in 8.1 to 8.3 is to wrap the algorithm in another function that reinitialises the algorithm for the global minimiser. This wrapper algorithm is shown in algorithm 8.4.

Another implementation issue worth noting one more time is that the degree of accuracy that the minimisers and saddle points are estimated is dependent on the termination conditions specified

Algorithm 8.4 Global Optimisation Algorithm Wrapper Function

Require: Starting point x_k
Require: Global minimiser storage list **G**
1: **loop**
  Run Global minimisation algorithm from x_k to get global minimiser x_g^*;
2: **if G** is not empty and x_g^* is better than all minimisers in **G then**
  Set $x_k = x_g^*$;
  continue;
3: **else**
  Set x_g^* to be the best minimiser in **G**;
  break;
  Return global minimiser x_g^*;

for the underlying GAD and minimisation algorithms. It is recommended that the underlying algorithms use the residual, gradient and point change differentials as part of the termination conditions to get good results.

Of course, the the eigenvalue finding mechanisms that are employed have a huge impact on the performance of the algorithm as the dimensionality grows. Therefore, trade off have to be made to ensure the best performance. For example, one may resort to use the 'Natural' GAD variant for high dimensional cases in order to avoid a large number repetitive computation of eigenvalues and eigenvectors in every iteration to sacrifice the rate of convergence for efficient use computational resources.

As a final note, it must be mentioned that the modification of the use of the global descent function G at a saddle point where the search directions on G are restricted to the critical points unstable manifold can be relaxed. This may be useful in terms of efficiency where one may need to reduce the number of times eigenvalue/eigenvector calculations are done. The relaxation allows one to randomly distribute new starting points around the saddle point or any other method the implementer prefers.

8.6. Conclusion

In this chapter we devised two global optimisation algorithms that make use of saddle points and employ a policy that ensures that better minimisers are found at every iteration. The first algorithm makes use of the global descent function from [5] with some modifications to make it work at saddle points; the second algorithm is a novel approach to global optimisation that makes use of a saddle point's unstable manifolds to search for better minimisers.

9
Numerical Results

God made the natural numbers; all else is the work of man.

– Leopold Kronecker, (1823-1891) [39]

9.1. Introduction

The proposed algorithms in the previous chapters encompassing the main body of this work have been implemented in MATLAB. The aim of this chapter is to present the numerical experience from using the proposed algorithms.

9.2. Design of the Computational Experiments

The algorithms have been implemented in MATLAB Release 2013a and run on an iMac with a 2.7 GHz Intel Core $i5$ processor and 8 GB 1600 MHz DDR3 memory. The tests are split in two parts:

1. Convergence Theory of the GAD and Global optimisation algorithms;
2. Measuring performance of the GAD and Global optimisation algorithms.

The set of problems which were used for the tests are shown below:

- Problem 1: Ackley function (with $n = 2, 3, 5, 10, 20$);
- Problem 2: Rastrigin function (with $n = 2, 3, 5, 10, 20$); and
- Problem 3: Schwefel function (with $n = 2, 3, 5, 10, 20$).

More information on the test functions is in appendix A. Table 9.1 summarises the important aspects of the problems stated above.

For the part of the tests dealing with the convergence theory (convergence rate), the model we will use is the Quotient-convergence (Q-convergence) model. This model uses the quotient between two

Table 9.1.: *Problem Statistics*

Problem				
Name	No. of variables	Hypercube of Definition	Global Minimiser x^*	$f(x^*)$
1 (Ackley)	2	$x_i \in [-32.768, 32.768]$	(0,...,0)	0
	3	$x_i \in [-32.768, 32.768]$	(0,...,0)	0
	5	$x_i \in [-32.768, 32.768]$	(0,...,0)	0
	10	$x_i \in [-32.768, 32.768]$	(0,...,0)	0
	20	$x_i \in [-32.768, 32.768]$	(0,...,0)	0
2 (Rastrigin)	2	$x_i \in [-5.12, 5.12]$	(0,...,0)	0
	3	$x_i \in [-5.12, 5.12]$	(0,...,0)	0
	5	$x_i \in [-5.12, 5.12]$	(0,...,0)	0
	10	$x_i \in [-5.12, 5.12]$	(0,...,0)	0
	20	$x_i \in [-5.12, 5.12]$	(0,...,0)	0
3 (Schwefel)	2	$x_i \in [-500, 500]$	(420.9687, 420.9687)	−837.9658
	3	$x_i \in [-500, 500]$	(420.9687, 420.9687, 420.9687)	−1256.9487
	5	$x_i \in [-500, 500]$	(420.9687, ..., ..., 420.9687)	−2094.9145
	10	$x_i \in [-500, 500]$	(420.9687, ..., ..., 420.9687)	−4189.829
	20	$x_i \in [-500, 500]$	(420.9687, ..., ..., 420.9687)	−8379.658

successive terms which is given by:

$$\lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|^q} = R \tag{9.1}$$

where L is the number that the sequences x_k converge toward, q is the convergence order and R is the rate of convergence [40, 41, 42].

For $q = 1$, if the sequence converges, and

- $R = 0$, then the sequence is said to converge **super-linearly**,
- $R = 1$, then the sequence is said to converge **sub-linearly** [40, 41, 42].

For $q > 1$, if the sequence converges to L, with $R < \infty$ then

- q = 2 is called **quadratic** convergence,
- q = 3 is called **cubic** convergence,
- q = 4 is called **quartic** convergence,
- q = 5 is called **quintic** convergence,
- etc [40, 41, 42].

Therefore, the easiest way to analyse the rate of convergence is by plotting a graph of the error $|x_{k+1} - L|$ vs. the number of iterations. To get the the rate we study the logarithmic scale plot of the same graph and it will give the value of q; that is, if the corresponding logarithmic scale plot is linear, then $q = 1$, if it is quadratic, then $q = 2$, etc.

For the other part of the test dealing with the accuracy and performance of the algorithms we will define a metric to measure performance. The performance of all the proposed GAD algorithms

was evaluated using the following performance index:

$$PI = \frac{f(x^*_{ini}) - f(x^*_{sp})}{f(x^*_{ini}) - f(x^*_{actual\_sp})}, \tag{9.2}$$

where x^*_{ini} is the initial point, x^*_{sp} is the saddle point returned by the algorithm and $x^*_{actual\_sp}$ is the true saddle point.

Furthermore, the performance of all the proposed global optimisation algorithms was evaluated using the following performance index:

$$PI = \frac{f(x^*_{ini}) - f(x^*_{final})}{f(x^*_{ini}) - f(x^*_{global})}, \tag{9.3}$$

where x^*_{ini} is the initial point, x^*_{final} is the final minimiser returned by the algorithm and x^*_{global} true global minimiser.

Thus, $0 \leq PI \leq 1$. For the two extreme cases, $PI = 1$ means that the algorithm succeeds in finding the concerned critical point and $PI = 0$ means that no improvement from the initial point has been made.

The computational results for the performance and accuracy tests are given in tables 9.2, 9.3, 9.4 and 9.5. For each entry, 2 experiments were performed, the results averaged and entered in the tables where the notation is as follows:

- Problem = notation of the problem;
- Name = name of the objective function;
- n = number of decision variables;
- N_{iter} = average number of iterations;
- $N_{gradient}$ = average number of gradient evaluations;
- $N_{hessian}$ = average number of Hessian evaluations;
- N_{eig} = average number of eigenvalue-eigenvector evaluations;
- T_{exec} = average CPU execution time in seconds;
- N_{PI} = the average performance indices;

In the next section we present the the relevant results from these computational experiments on the prescribed problems. They are presented as graphs and tables. For more information on the test functions, implementation-centric parameters used, and the complete set of the results refer to appendices A, B and C, respectively. The analyses of these results will be done in chapter 11, the discussion chapter.

9.3. Results of Computational Experiments

This section contains the results of computational experiments. However, a majority of the graphical plots for the convergence rate have been omitted and only the relevant ones in high dimensions

have been included. This is because of their voluminous nature. For the complete set of results, refer to appendix C. The results now follow:

Table 9.2.: *Results and solution statistics for Natural GAD*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	46	47	47	2	0.9323	1
	3	77	78	78	2	2.3589	1
	5	20	21	21	2	1.1036	1
	10	11	12	12	2	1.6169	1
	20	19	20	20	2	6.5468	1
2 (Rastrigin)	2	41	42	42	2	2.5714	1
	3	112	113	113	2	9.4933	1
	5	38	39	39	2	8.4202	1
	10	23	24	24	24	10.2352	1
	20	36	37	37	37	31.8601	1
3 (Schwefel)	2	144	145	145	2	8.6552	1
	3	151	152	152	2	13.6435	1
	5	107	108	108	2	43.2022	1
	10	123	124	124	2	37.4938	1
	20	567	568	568	2	331.9135	1

Table 9.3.: *Results and solution statistics for Rayleigh GAD*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	10	11	11	11	2.3566	1
	3	51	52	52	52	10.0858	1
	5	65	66	66	66	22.0817	1
	10	103	104	104	104	80.6100	1
	20	19	19	19	19	40.4132	1
2 (Rastrigin)	2	92	93	93	93	8.3496	1
	3	247	248	248	248	28.2447	1
	5	137	138	138	138	27.5040	1
	10	103	104	104	104	46.1129	1
	20	224	225	225	225	225	1
3 (Schwefel)	2	157	158	158	158	14.9944	1
	3	172	173	173	173	25.2781	1
	5	167	168	168	168	43.3048	1
	10	174	174	174	174	86.7005	1
	20	185	186	186	186	228.4269	1

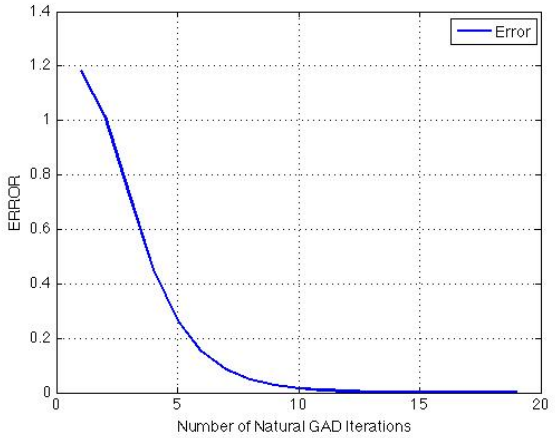
Table 9.4.: *Results and solution statistics for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	12	625	701	152	14.7518	1
	3	9	2935	2857	116	86.8983	1
	5	5	2626	6188	52	92.9138	1
2 (Rastrigin)	2	1	29	31	4	2.4126	1
	3	1	60	62	4	6.6653	1
	5	1	15	17	4	3.9685	1
	10	1	31	33	4	11.5332	1

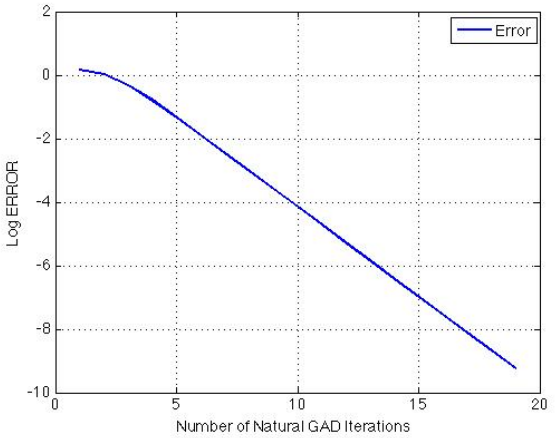
Table 9.5.: *Results and solution statistics for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	10	173	199	199	30.2412	1
	3	4	238	246	246	32.7275	1
	5	6	439	465	465	106.5899	1
2 (Rastrigin)	2	10	333	353	50	23.0188	1
	3	12	232	256	60	26.1143	1
	5	30	1016	1076	150	163.9899	1
	10	46	8588	8680	20	3798.9	1

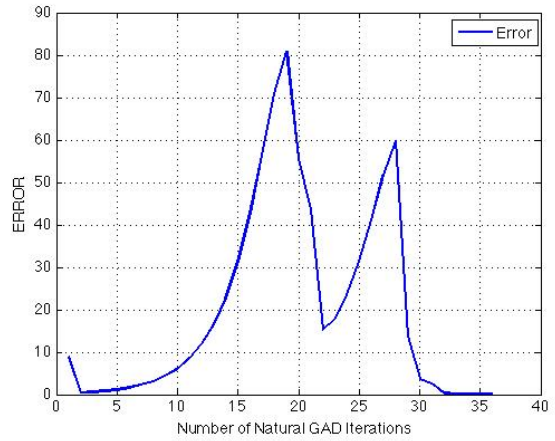
Figure 9.1.: *Rate of Convergence Results for Natural GAD on Problems 1,2 and 3 for* $n = 20$



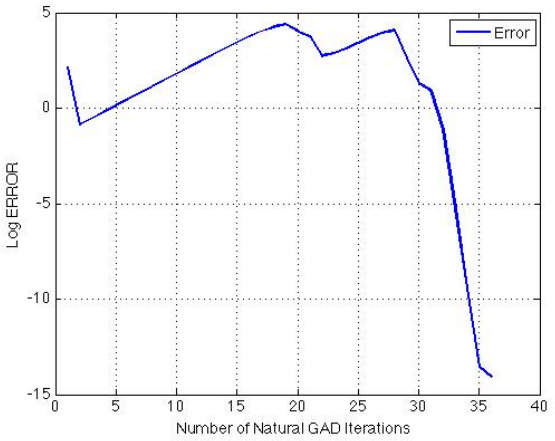
(a) Ackley Function, $n = 20$: Error vs. Iterations



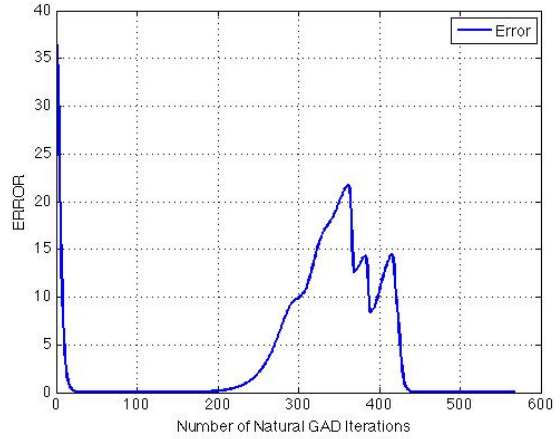
(b) Ackley Function, $n = 20$: Log Error vs. Iterations



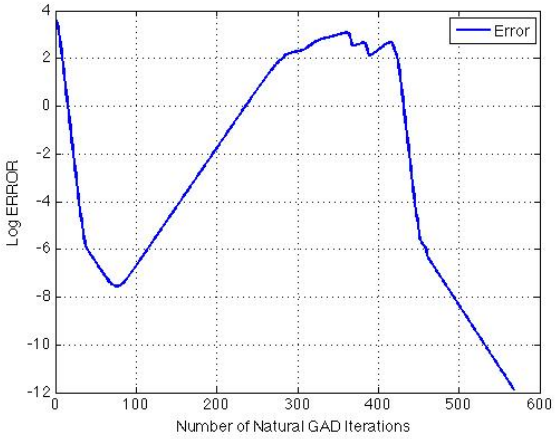
(c) Rastrigin Function, $n = 20$: Error vs. Iterations



(d) Rastrigin Function, $n = 20$: Log Error vs. Iterations

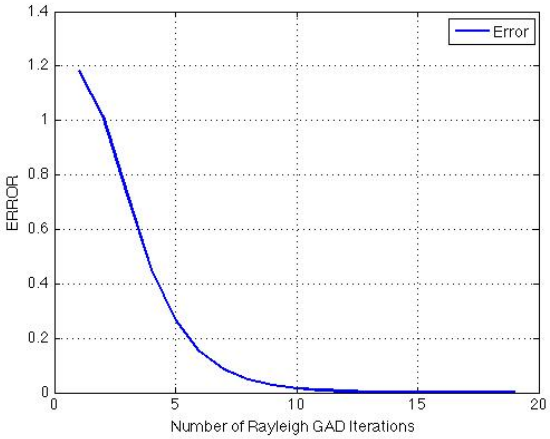


(e) Schwefel Function, $n = 20$: Error vs. Iterations

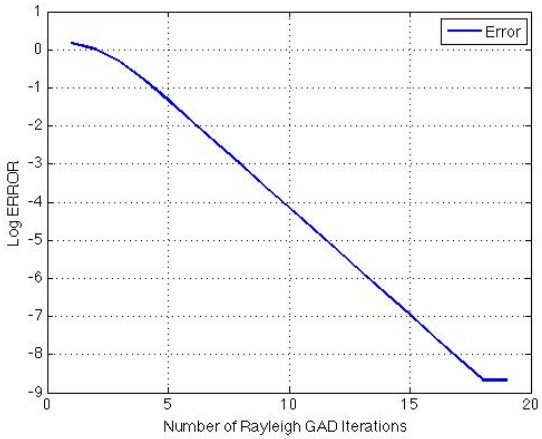


(f) Schwefel Function, $n = 20$: Log Error vs. Iterations

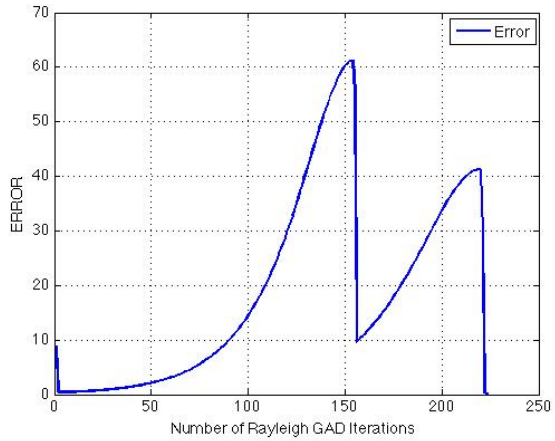
Figure 9.2.: *Rate of Convergence Results for Rayleigh GAD on Problems 1,2 and 3 for* $n = 20$



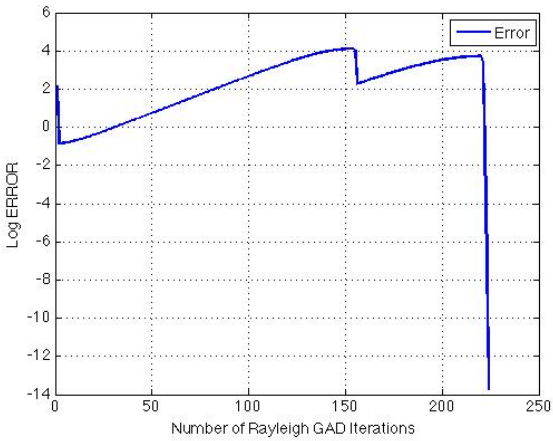
(a) Ackley Function, $n = 20$: Error vs. Iterations



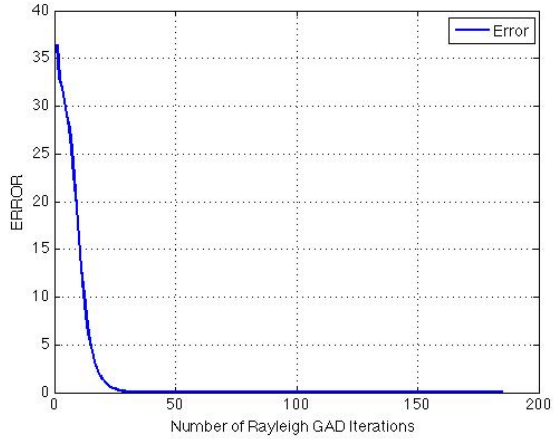
(b) Ackley Function, $n = 20$: Log Error vs. Iterations



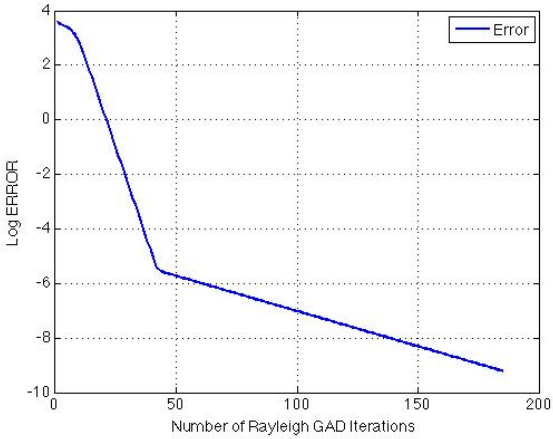
(c) Rastrigin Function, $n = 20$: Error vs. Iterations



(d) Rastrigin Function, $n = 20$: Log Error vs. Iterations

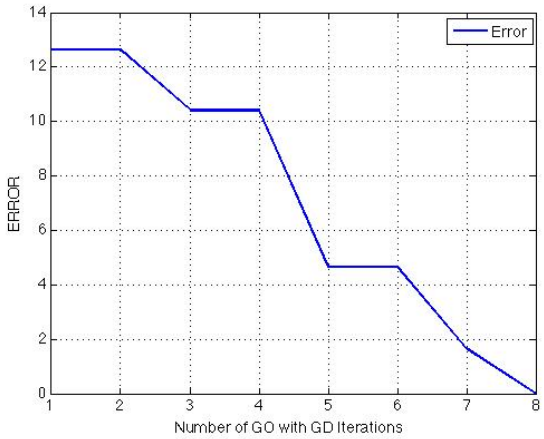


(e) Schwefel Function, $n = 20$: Error vs. Iterations

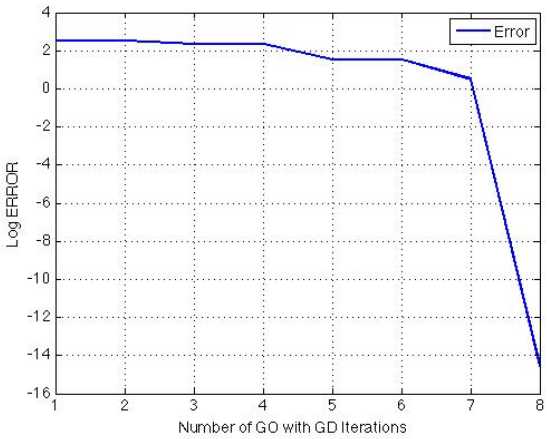


(f) Schwefel Function, $n = 20$: Log Error vs. Iterations

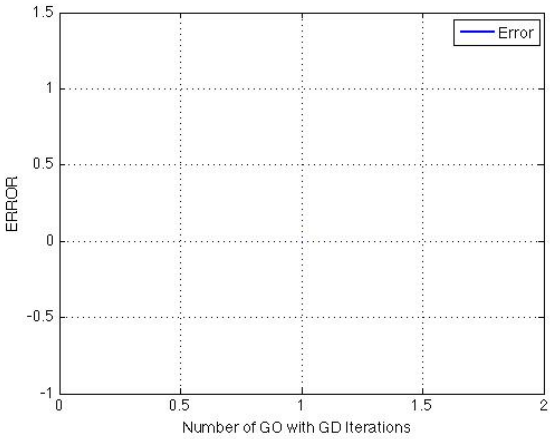
Figure 9.3.: *Rate of Convergence Results for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions on Problems 1 and 2 for* $n = 5, 10$



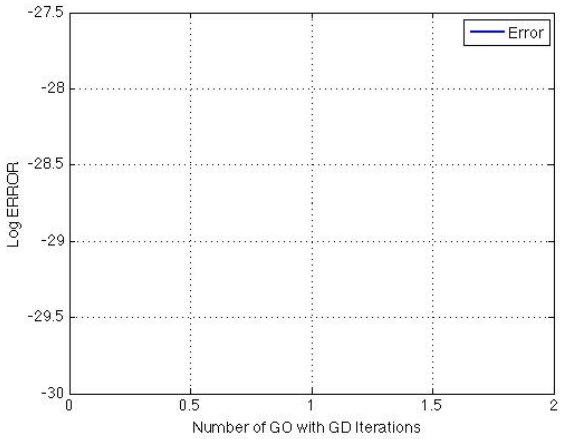
(a) Ackley Function, $n = 5$: Error vs. Iterations



(b) Ackley Function, $n = 5$: Log Error vs. Iterations

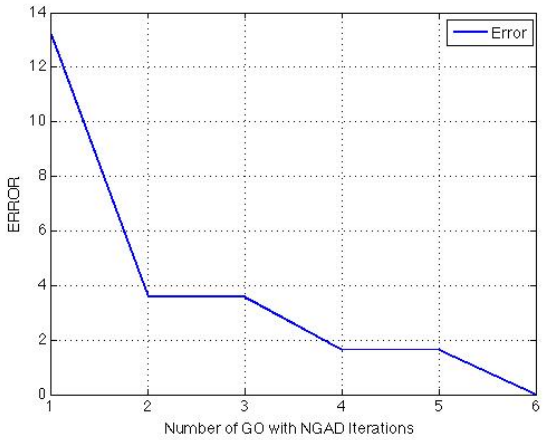


(c) Rastrigin Function, $n = 10$: Error vs. Iterations (Done in 1 iteration)

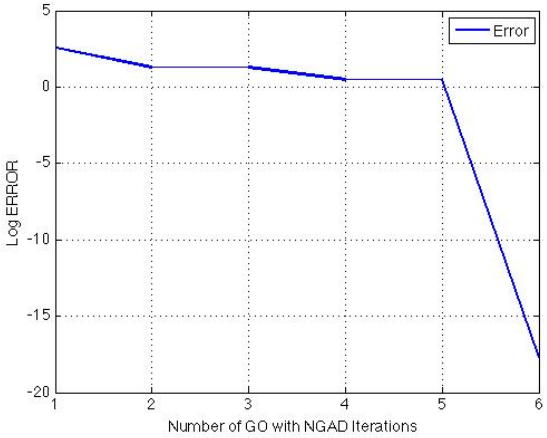


(d) Rastrigin Function, $n = 10$: Log Error vs. Iterations (Done in 1 iteration)

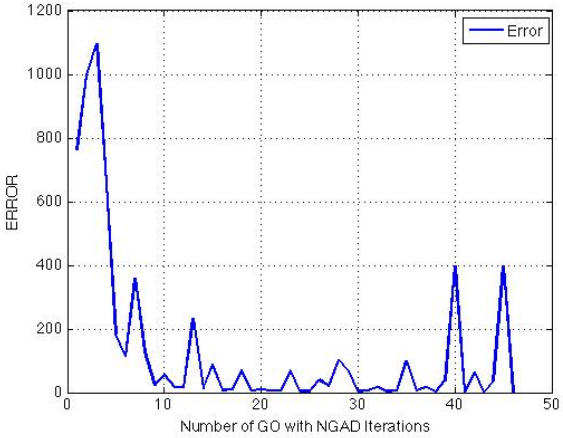
Figure 9.4.: *Rate of Convergence Results for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches on Problems 1 and 2 for* $n = 5, 10$



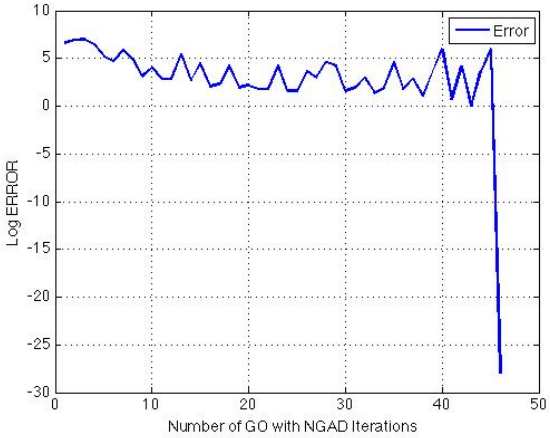
(a) Ackley Function, $n = 5$: Error vs. Iterations



(b) Ackley Function, $n = 5$: Log Error vs. Iterations



(c) Rastrigin Function, $n = 10$: Error vs. Iterations



(d) Rastrigin Function, $n = 10$: Log Error vs. Iterations

9.4. Summary

In this chapter we have presented the results of numerical experimentation on the algorithms developed in this thesis. We first presented problems from literature and carried out experiments that were split in two parts:

1. Convergence Theory of the GAD and Global optimisation algorithms;
2. Measuring performance of the GAD and Global optimisation algorithms.

The set of problems which were used for the tests are shown below:

- Problem 1: Ackley function (with $n = 2, 3, 5, 10, 20$);
- Problem 2: Rastrigin function (with $n = 2, 3, 5, 10, 20$); and
- Problem 3: Schwefel function (with $n = 2, 3, 5, 10, 20$).

More information on the test functions and implementation details is in appendices A and B, respectively. An analysis of these results is done in chapter 11.

10

Example Problem: Global Optimisation of Lennard-Jones Clusters

10.1. Introduction

We now apply our algorithms to a practical problem that has attracted both theoretical and computational research. This problem is the minimisation of the potential energy function of Lennard-Jones atomic clusters. There are two main reasons why this is an interesting problem.

Firstly, the practical importance of discovering the low energy configurations of a cluster of atoms is important in molecular conformation research [43]. For example, in chemical physics, researchers are interested in finding the lowest energy configurations of a macro-molecular structure [44, 45]. A specific application of this is in protein folding where the native structure is related to the global minimum of its potential energy function [44, 45].

Secondly, the minimisation of the Lennard-Jones potential energy function is an easy to state problem and yet it poses enormous difficulties for any unbiased global optimisation algorithm [43]. Therefore, it serves as a good test system for global optimisation algorithms.

10.2. Problem Formulation

The Lennard-Jones (L-J) potential, first proposed in 1924 by John Lennard-Jones, is given by

$$E = 4\epsilon \sum_{i<j} \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{10.1}$$

or more commonly as

$$E = \epsilon \sum_{i<j} \left[\left(\frac{r_m}{r_{ij}} \right)^{12} - 2 \left(\frac{r_m}{r_{ij}} \right)^{6} \right] \tag{10.2}$$

where ϵ and $r_m = 2^{1/6}\sigma$ are the pair equilibrium well depth and separation, respectively [43] [45]. For our treatment as in many others, we will employ reduced sizes $\epsilon = \sigma = 1$ throughout to give

$$E = \sum_{i<j} \Big[\Big(\frac{1}{r_{ij}}\Big)^{12} - \Big(\frac{1}{r_{ij}}\Big)^{6} \Big]. \tag{10.3}$$

To get the intuition behind the equation above, we remember that molecular conformation problems involve finding the global minimum of the potential energy function which depends on the relative positions of the atoms. In the L-J model, all atoms are considered equal and only pairwise interactions are included in the definition of the potential energy [43, 45]. Let us take $N \geq 2$ be an integer representing the total number of atoms. The L-J pairwise potential energy function is defined as follows: if the distance between the centres of a pair of atoms is r, then their contribution to the total energy is defined to be

$$v(r) = \frac{1}{r^{12}} - \frac{2}{r^{6}} \tag{10.4}$$

and the L-J potential energy which we will call E of the molecule is defined as

$$E(X) = E(X_1, X_2, ..., X_N) = \sum_{i<j} v(||X_i - X_j||) \tag{10.5}$$

where $X_i \in \mathbb{R}^3$ represents the coordinates of the centre of the i-th atom and the norm is used is the usual Euclidean one [43, 45]. The optimal L-J configuration $X^* = \{X_1, X_2, ..., X_N\}$ is defined as the solution to the global optimisation problem

$$LJ_N = E(X^*) = \min_{X \in \mathbb{R}^3} E(X). \tag{10.6}$$

It is this problem that we will be solving with our algorithm in this chapter.

10.3. Design of the Computational Experiments

As before, the algorithms have been implemented in MATLAB Release 2013a and run an iMac with a 2.7 GHz Intel Core $i5$ CPU with 8 GB 1600 MHz DDR3 memory. The goal of the tests is to measure the performance and accuracy of the Global optimisation algorithms on the Lennard-Jones clusters.

The set of problems which were used for the tests are shown below:

- Problem 1: L-J potential energy function (with $N = 2$);
- Problem 2: L-J potential energy function (with $N = 3$);

More information on the test functions is in appendix A. The table below summarises the important aspects of the problems stated above.

Table 10.1.: *Problem Statistics*

N	energy (E_{min})	reference
2	-1.0000	[44, 46, 47]
3	-3.0000	[44, 46, 47]

As was done with the tests in the numerical experience chapter, we will define a metric to measure performance. The performance of all the proposed global optimisation algorithms was evaluated using the following performance index:

$$PI = \frac{f(X^*_{ini}) - f(X^*_{final})}{f(X^*_{ini}) - f(X^*_{global})}, \tag{10.7}$$

where X^*_{ini} is the initial configuration, X^*_{final} is the final minimising configuration's energy returned by the algorithm and X^*_{global} true global minimising configuration.

Thus, as before, $0 \leq PI \leq 1$. For the two extreme cases, $PI = 1$ means that the algorithm succeeds in finding the concerned critical point and $PI = 0$ means that no improvement from the initial point has been made.

The computational results for the performance and accuracy tests are given in tables 10.2 and 10.3. For each entry, 2 experiments were performed, the results averaged and entered in the tables where the notation is as follows:

- Problem = notation of the problem;
- Name = name of the objective function;
- N = Number of atoms;
- N_{iter} = average number of iterations;
- $N_{gradient}$ = average number of gradient evaluations;
- $N_{hessian}$ = average number of Hessian evaluations;
- N_{eig} = average number of eigenvalue-eigenvector evaluations;
- T_{exec} = average CPU execution time in seconds;
- N_{PI} = the average performance indices;

In the next sections we present the results of these tests; for information on implementation-centric parameters used refer to appendix B. An analysis of these results is done in chapter 11.

10.4. Results of Computational Experiments

In this section we present the results of the computational experiments carried out on L-J clusters with 2 and 3 atoms. These results are shown in the tables that follow.

Table 10.2.: *Solution statistics for Global Optimisation Approach 1 (Global Descent Functions) on L-J Clusters*

Problem							
Name	N	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
L-J Potential Energy Function	2	1	70	72	4	49.3584	1
	3	1	3	5	4	1.8206	1

Table 10.3.: *Solution statistics for Global Optimisation Approach 2 (Unbounded Line Searches) on L-J Clusters*

Problem							
Name	N	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
L-J Potential Energy Function	2	1	71	75	7	23.9450	1
	3	2	13	25	18	12.6516	1

10.5. Summary

In this chapter we have presented the results of numerical experimentation of the global optimisation algorithms developed in this thesis tested on the L-J clusters problem.

The set of problems which were used for the tests are shown below:

- Problem 1: L-J potential energy function (with $N = 2$);
- Problem 2: L-J potential energy function (with $N = 3$);

More information on the test functions and implementation details is in appendices A and B, respectively. An analysis of these results is done in chapter 11.

11 Discussion

11.1. Introduction

The purpose of this chapter is to discuss and analyse the results of the experiments and numerical experience in the preceding two chapters. The goal is to critically examine these numerical results in the light of the previous state of the subject matter in this thesis, and includes judgments on what has been learnt in this work.

11.2. Analysis of Results on the GAD-based Saddle-Point-Finding Algorithms

One of the major aims of this work was to develop a computationally efficient, numerically stable and globally convergent GAD-based saddle-point-finding algorithm. We started out by postulating the *Desirable Initial Conditions* that were used in the *Natural* and *Rayleigh Optimisation* versions of the GAD-based algorithms.

Based on the GAD equations, we know that these algorithms have a two-phased approach for finding a saddle point. The first phase involves finding the separatrix associated with a saddle point and once this is achieved the second stage is converging toward the saddle point itself. With our devised algorithms, we observed that each of these stages required its own optimal step-size where the separatrix hunting phase usually needed a smaller step-size in contrast to the final stage that required a much larger and more optimistic one. Therefore, the best results were achieved with an optimal step-size strategy that took the needs of both phases into account. As a result, the first stage where the solution trajectory is finding a separatrix involves a larger fraction of the algorithm's iterations in comparison to the subsequent phase.

As can been seen from each of the *Iterations vs Error* plots in chapter 9, the separatrix find-

ing stage involved an increasing error; once the separatrix was found, the error decreased sharply until the solution trajectory converged at a saddle point. The corresponding *Iterations vs* **log** *Error* showed that each of these phases had linear convergence. This means that the overall convergence rate of the devised GAD-based saddle-point-finding algorithms is **linear**.

Concerning the reasons for the different versions of the algorithms, the motivation for developing the Rayleigh GAD algorithm was to find a way of designing an algorithm that converges towards the separatrix at a much faster rate than its counterpart following the natural approach. From the performance results, it is evident that it achieves this goal by reaching the separatrix and eventually finding the saddle point in less iterations. However, it does so in more time than the natural approach. This shows the tradeoff that needs to be made by implementers of the paradigms that have been proposed in this work. Of course, it is evident that faster implementations of the techniques used within the Rayleigh GAD method could make it a much superior algorithm.

Amidst all this success of the devised algorithms are the *Desirable Initial Conditions*. The results in the tables and plots show that the importance of these conditions are in intertwined in two aspects. Firstly, it ensures that the generated GAD solution trajectory is guaranteed to find a separatrix. Secondly, it prevents unstable trajectories.

During the numerical experience, it was discovered that the parameters used by the algorithm such as the error tolerances and step-sizes were highly dependent on the geometry of the underlying function. For example, in higher dimensional Ackley functions, a correctly set constant step-size was more efficient than using a line search step-size strategy. Furthermore, the Rastrigin function required step-sizes with much smaller orders in comparison with the Schwefel function that needed step-sizes with upper bounds as high as 1 to 1.5.

So, on the whole, did this work manage to achieve the task of creating a computationally efficient, numerically stable and globally convergent GAD-based saddle-point-finding algorithm? Let us answer this question in terms of its constituent parts.

In terms of computational efficiency, in as much as the performance of the algorithm is quite impressive, the GAD-related statistics shown in tables 9.2 and 9.3 are intriguing. The number of gradient, Hessian and eigenvalue/eigenvector calculations in comparison to the number of algorithm iterations is quite alarming. Despite the general $1:1$ ratio of these numbers to the number of iterations, these calculations are expensive operations in themselves. Therefore, it is desirable to perform as few of them as possible. Furthermore, for higher dimensional cases, this could become a bottleneck for the algorithm and may, in turn, affect its scalability. This is made even worse in cases where one may need to set very small step-sizes that may invoke an even higher number of iterations. In the same vein, the issue of computational complexity is entangled with the issue of *time* versus *number of iterations* trade-off aspects of the algorithm.

Are the algorithms globally convergent? Not necessarily. Because the algorithms need to abide by the *Desirable Initial Conditions* to ensure convergence, the *global* convergence aspect is lost. However, as already mentioned, these conditions can easily be achieved and the question of *global*

convergence is not such an important one for practical high dimensional cases.

The question of numerical stability is also connected to that of global convergence and the *Desirable Initial Conditions*. Was it achieved? Of course. With the correct parameters, the algorithm will always find the closest saddle point. Even so, the saddle point found will not necessarily be an index-1 saddle point. Therefore, the pool of saddle points includes those of all indices as long as they are in proximity of the initial conditions. But, as already stated earlier, for very small step-sizes, the GAD trajectory will find an index-1 saddle point.

As a final note on the issue of finding saddle points, researchers in this space are always seeking to formulate methods that target particular types of saddle points. The GAD paradigm was derived in order to seek out the index-1 saddle points. Unfortunately, for our algorithm to achieve the task of finding this particular type of saddle points in higher dimensional spaces, its computational efficiency suffers.

11.3. Analysis of Results on the Global Optimisation Algorithms

The other major goal in this dissertation was to design and construct a global optimisation algorithm that uses saddle points as transitional points. Two algorithms were developed using this paradigm. The first used the global descent function at saddle points and the other made use of unbounded line searches. Both proposed algorithms were based on the GAD "Chaining" algorithm and were very successful in the regard of using saddle points to find global minimisers.

Firstly, these global optimisation algorithms were developed around the policy that at every iteration better minimisers should be found. This policy ensured that each transition through a saddle point served as an improvement on the current minimiser. However, this way of approaching the global optimisation problem has the fault of being dependent on the geometry of the function. The hurdles of using this approach are in two aspects. To start with, the first stumbling block of using this prescribed approach is that if the next best minimiser is not in a bounded vicinity of the current transitional saddle point then the algorithm will fail to improve the current minimiser. The second drawback is that the approach may be ineffective on geometrically deceptive functions. As previously noted, this is because of the approach's dependence on the objective function's geometry. Since the resulting solution trajectory hops between minimisers with the overall goal of always improving on the preceding one, the paradigm essentially assumes that the global minimiser will be in the vicinity of other good minimisers. However, for cases where the global minimiser does not exist in close proximity with other better minimisers the algorithm may face difficulties. To succeed, the algorithm may need to be initialised at a point in the domain space that can enable it to eventually transition though a saddle point whose unstable directions are oriented in the direction of the global minimiser. And, as one can imagine, for higher dimensional cases this may take considerable time to happen for the worst scenarios.

It is for the reasons outlined in the previous paragraph that a **multi-start** approach to the global optimisation algorithm may be beneficial. Strategic distribution of starting points around the domain of definition of the objective function would help a great deal in overcoming this geometry

dependence.

Moreover, because of this topological and Morse-theoretic dependence of the transitions through saddle points, the question of termination conditions becomes a very imperative one. Throughout the development of the global optimisation methods in this dissertation, the termination condition used was designed to stop if a better minimiser could not be found. But with the geometric-dependence it may be necessary to allow the algorithm to make "mistakes" by finding some worse minimisers in the hope that the subsequent transition process can yield a better minimiser overall. This way the algorithm can hopefully correct itself depending on the geometry of the objective function in question.

Based on the test results shown in chapter 9, the devised algorithms work. However, from the graphical plot in figures , and those in appendix C, the convergence rate of all the global optimisation algorithm is **linear**. Furthermore, it should be noted that the a large fraction of the time during the execution of the algorithm was spent finding saddle points. Therefore, using a faster saddle-point-finding method can greatly improve the algorithm's speed.

The global optimisation algorithms developed in this work are not without their shortcomings. The most prominent drawback is the *parametric* nature of the algorithms. This is the best described by the multiple scenarios where an optimal set of parameters needs to be set in order for the algorithm to be most efficient. In as much as it is understandable that this is the problem with practically any numerical algorithm, it is worth pointing out in this case. The types of parameters include and are not limited to error parameters, termination-centric tolerances and even the number of times to allow the algorithm to make "mistakes".

11.4. Analysis of Results from the Tests on Lennard-Jones Clusters

The global optimisation algorithms devised in this thesis were applied to the global minimisation of the potential energy functions of L-J clusters with 2 and 3 atoms. From the results in the previous chapter, the algorithm was successfully able to determine the most optimal configurations of atoms with the lowest energy levels.

It must be mentioned that this problem has a large number of local and global minimisers. Therefore, performance depends on the initial conditions and the transitional points used in the higher demential space. The problems used with 2 and 3 atoms have dimensionality of 6 and 9, respectively.

As stated in the previous section, the phases involving finding the saddle points were the most expensive and therefore better saddle-point-finding methods may be useful.

11.5. Summary

In this chapter we discuss and analyse the results of the experiments and numerical experience in the preceding two chapters. The developed GAD-based saddle-point-finding algorithms are effective

in their task except from concerns related to computational efficiency in higher dimensional spaces. This concern is in relation to the number of times the algorithm calculates the gradient, Hessian and eigenvector/eigenvalue pairs each run. For the global optimisation algorithms using saddle points, they are equally effective except that their dependence on the geometry of the objective function in order to work well could be a serious hurdle. To mitigate for this problem a multi-start solution is suggested. The last section of the chapter briefly outlines the results of applying the global optimisation algorithms to the L-J clusters problem.

12
Conclusions

In opposition to the foolish "ignorabimus" our slogan shall be: We must know — we will know!

– David Hilbert, 1930 [48]

12.1. Summary of Work

The work in this treatise had dual aims. The first was the numerical distillation of the GAD in order to develop a numerically stable and globally convergent index-1 saddle-point-finding algorithm. The second aim was to construct a global optimisation algorithm that took advantage of the Morse-theoretic properties of saddle points by using them as transitional points to find better minimisers.

We started out by first developing two effective and linearly convergent GAD-based saddle-point-finding algorithms that make use of the postulated *Desirable Initial Conditions* to ensure convergence. On one hand, these algorithms are different because the first one follows the *natural* essence of the GAD autonomous ODEs and the other ensures that convergence toward a saddle point takes place in less iterations. It does so by infusing a *Rayleigh optimisation* into the algorithm. On the other hand, these algorithms are also complimentary in that the aggressive nature of the second approach makes it harder for it to find lower index saddle points and may be ineffective for some functions. These algorithms are important because they can now be part of the repertoire of algorithms that are useful in finding saddle points. This is especially useful in the analysis of transitional points of minimum energy paths in areas not limited to computational physics and chemistry but also in the atomistic simulation of rare events [9].

The GAD "Chaining" Algorithm was developed based on GAD-algorithms. This algorithm has the goal of finding all the saddle points on a non-linear differentiable function. It abstracts the

search for saddle points as a graph exploration problem. The nodes on this graph are saddle points and minimisers. The algorithm uses the breadth first search approach to explore the function for saddle points. The name *GAD "Chaining"* comes from the fact that intermittent runs of the GAD algorithm are interleaved with local minimisation runs. This algorithm, in itself, is very useful because knowing all the saddle points on a function means that one knows how the function behaves everywhere.

The next phase in the development of this work was the proposition of the two global optimisation algorithms. The global optimisation algorithms that were developed are essentially modifications of the GAD "Chaining" algorithm except that they follow a policy where they seek out better minimisers at every iteration. The first was dependent on a modified use of the global descent function at saddle points where the function exploration is restricted in such a way that only better saddle points from a current saddle point are explored. The role of the global descent function is to find a new starting point for the GAD and local minimisation algorithm to find a better minimiser and saddle point. The second approach replaces the use of the the global descent function with an unbounded line search in the unstable directions at the saddle points to find a new starting point for the GAD and local minimisation algorithm to find a better minimiser and saddle point. The resulting global optimisation algorithms are both linearly convergent.

12.2. Problems to Overcome, Research Issues and Future Directions

Firstly, the major area where the work presented in this thesis requires improvement is that of efficiency. Finding more efficient ways to simulate the Rayleigh optimisation would be a major boost to the strength of this work. Furthermore, research into ways of of reducing the overhead of calculating the gradient, Hessian and eigenvalue/eigenvalue pairs in the algorithms would greatly improve their efficiency. Outside of this, an interesting approach would be to find ways of simulating GAD without excessive use of the gradient and Hessian.

Secondly, the dependence of the proposed global optimisation algorithms on the geometry of the objective functions is an interesting problem in itself. As suggested earlier, multi-start approaches are one way to combat this. It would be interesting to explore possible multi-start paradigms for global optimisation algorithms that suffer from this problem. Additionally, finding other ways apart from multi-start methods to combat this 'geometric' dependence is an intriguing area of research considering that most global optimisation algorithms need to be biased in one way or another for them to be able to optimise geometrically deceptive functions.

Thirdly, research work that could use stochastic versions of GAD as suggested by Weinan E and Samanta in [9] on the proposed global optimisation algorithms would be interesting. This would give a theoretical handle on stochastic saddle-point-finding methods. Additionally, it would also aid in a comparison and analysis of the cost-benefit examination of using stochastic methods versus using deterministic methods for saddle-point-centric global optimisation algorithms.

Lastly, research into better saddle-point-finding methods is imperative. An exploration of us-

ing the Dimer method or another approach to find saddle points instead of the Gentlest Ascent Dynamics (GAD) is an interesting prospect. Would those methods be more efficient? Would these other algorithms be faster? Would they involve less calculations of the gradient, Hessian and eigenvalues/eigenvectors? Moreover, exploring different ways of using saddle points to do global optimisation present a fascinating and compelling area of research.

12.3. Conclusion

The main result in this treatise is not necessarily the proposed set of effective saddle point finding and global optimisation algorithms, although these proposed linearly convergent methods have been analysed and their performance discussed. The major contribution of this work is the postulation of the premise that saddle points can be important transitional points in the area of global optimisation. All in all, the work presented in this treatise has achieved all its objectives and has posed interesting questions on the place that saddle points have in the global optimisation sphere.

A

Test Functions

A.1. Ackley Function

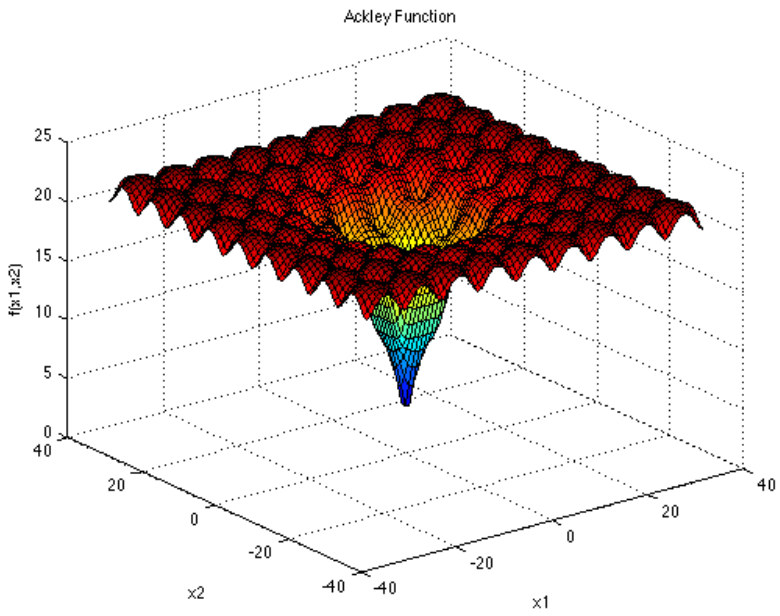


Figure A.1.: *2 dimenstional plot of the Ackley function*

The Ackley function is widely used for testing optimisation algorithms. In its two-dimensional form, as shown in the plot above, it is characterised by a nearly flat outer region and a large hole at the centre. The function poses a risk for optimisation algorithms, particularly hill-climbing algorithms, to be trapped in one of its many local minima [49, 50, 51, 52].

Recommended variable values are: $a = 20, b = 0.2$ and $c = 2\pi$ [49, 50, 51].

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

A.1.1. Input Domain

The function is usually evaluated on the hypercube $x_i \in [-32.768, 32.768]$, for all $i = 1, ..., n$ although it may also be restricted to a smaller domain such as $15 \leq x_i \leq 30$ [49, 50, 51, 52].

A.1.2. Global Minimum

The global minimum: $x^* = (0, ..., .., 0)$, and $f(x^*) = 0$ [49, 50, 51, 52].

A.2. Rastrigin Function

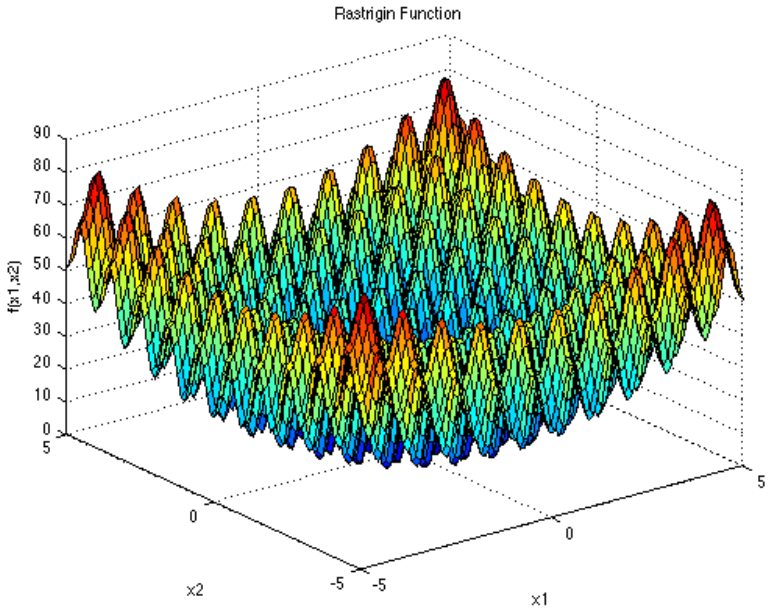


Figure A.2.: *2 dimenstional plot of the Rastrigin function*

The Rastrigin function has several local minima. It is highly multimodal, but locations of the minima are regularly distributed. It is shown in the plot above in its two-dimensional form [52, 53].

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

where d is the dimensionality of the problem.

A.2.1. Input Domain

The function is usually evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all $i = 1, 2, ..., n$ [52, 53].

A.2.2. Global Minimum

The global minimum: $x^* = (0, ..., ..., 0)$, and $f(x^*) = 0$ [52, 53].

A.3. Schwefel Function

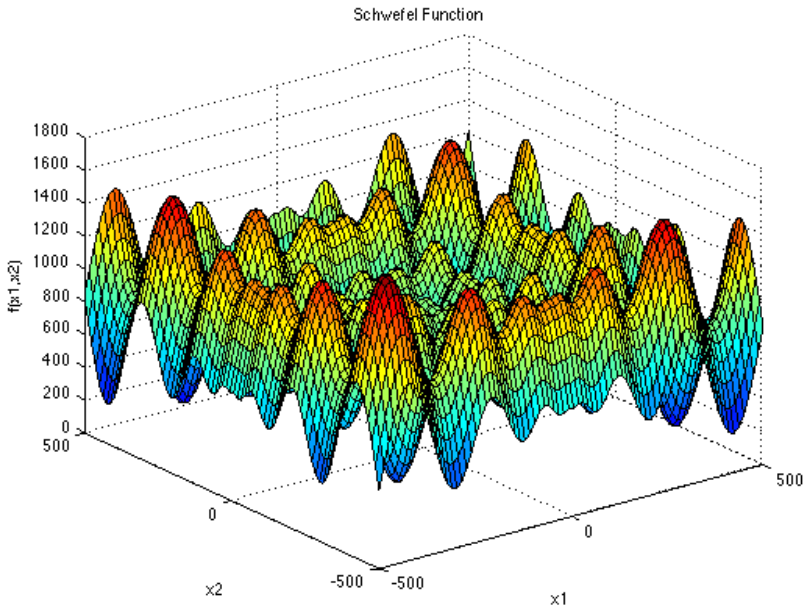


Figure A.3.: *2 dimenstional plot of the Schwefel function*

The Schwefel function is complex, with many local minima. Schwefel's function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction. The plot shows the two-dimensional form of the function [52, 54, 55].

$$f(\mathbf{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$$

A.3.1. Input Domain

The function is usually evaluated on the hypercube $x_i \in [-500, 500]$, for all $i = 1, 2, ..., n$ [52, 54, 55].

A.3.2. Global Minimum

The global minimum: $x^* = (420.9687, ..., ..., 420.9687)$, and $f(x^*) = -n \times 418.9829$, where n is the dimensionality of the function [52, 54, 55].

A.4. Lennard-Jones Potential Function

The Lennard-Jones (L-J) potential, first proposed in 1924 by John Lennard-Jones, is given by

$$E = 4\epsilon \sum_{i<j} \Big[\Big(\frac{\sigma}{r_{ij}} \Big)^{12} - \Big(\frac{\sigma}{r_{ij}} \Big)^{6} \Big] \tag{A.1}$$

or more commonly as

$$E = \epsilon \sum_{i<j} \Big[\Big(\frac{r_m}{r_{ij}} \Big)^{12} - 2 \Big(\frac{r_m}{r_{ij}} \Big)^{6} \Big] \tag{A.2}$$

where ϵ and $r_m = 2^{1/6}\sigma$ are the pair equilibrium well depth and separation, respectively [43] [45]. For our treatment as in many others, we will employ reduced sizes $\epsilon = \sigma = 1$ throughout to give

$$E = \sum_{i<j} \Big[\Big(\frac{1}{r_{ij}} \Big)^{12} - \Big(\frac{1}{r_{ij}} \Big)^{6} \Big]. \tag{A.3}$$

To get the intuition behind the equation above, we remember that molecular conformation problems involve finding the global minimum of the potential energy function which depends on the relative positions of the atoms. In the L-J model, all atoms are considered equal and only pairwise interactions are included in the definition of the potential energy [43, 45]. Let us take $N \geq 2$ be an integer representing the total number of atoms. The L-J pairwise potential energy function is defined as follows: if the distance between the centres of a pair of atoms is r, then their contribution to the total energy is defined to be

$$v(r) = \frac{1}{r^{12}} - \frac{2}{r^{6}} \tag{A.4}$$

and the L-J potential energy which we will call E of the molecule is defined as

$$E = (X) = E(X_1, X_2, ..., X_N) = \sum_{i<j} v(||X_i - X_j||) \tag{A.5}$$

where $X_i \in \mathbb{R}^3$ represents the coordinates of the centre of the i-th atom and the norm used is the usual Euclidean one [43, 45].. The optimal L-J configuration $X^* = \{X_1, X_2, ..., X_N\}$ is defined as the solution to the global optimisation problem

$$LJ_N = E(X^*) = \min_{X \in \mathbb{R}^3} E(X). \tag{A.6}$$

B

Details of Computational Experiments

The algorithms have been implemented in MATLAB Release 2013a and run on an iMac with a 2.7 GHz Intel Core $i5$ CPU and 8 GB 1600 MHz DDR3 memory.

B.1. Introduction

The tables used throughout this chapter contain the implementation details used in the numerical experience documented in this thesis. The notation used is supposed to be interpreted as follows:

- Problem = notation of the problem;
- Name = name of the objective function;
- α = step-size;
- α_f = fixed step-size;
- α_l = lower bound step-size;
- α_u = upper bound step-size;
- τ = Minimal Tolerance Parameter;
- n = number of variables;
- N = number of atoms;
- N_{iter} = average number of iterations;
- $N_{gradient}$ = average number of gradient evaluations;
- $N_{hessian}$ = average number of Hessian evaluations;
- N_{eig} = average number of eigenvalue-eigenvector evaluations;
- T_{exec} = average CPU execution time in seconds;
- N_{PI} = the average performance indices;

B.2. Implementation Details

Table B.1.: *Implementation details for Convergence Theory: Rate of Convergence of Natural GAD*

Problem				
Name	n	Step-Size	Initial Point	τ
1 (Ackley)	2	$\alpha_f = 0.1$	$(0.4, 0.6)$	10^{-26}
	3	$\alpha_f = 0.1$	$(0.23, 8, 5)$	10^{-26}
	5	$\alpha_f = 0.1$	$(0.23, 8, 5, 1, 1)$	10^{-26}
	10	$\alpha_f = 0.1$	$(0.23, 8, 5, 1, 1, 1, 2, 3, 4, 5)$	110^{-26}
	20	$\alpha_f = 0.1$	$(0.23, 8, 5, 1, 1, 1, 2, 3, 4, 5, 0.23, 8, 5, 1, 1, 1, 2, 3, 4, 5)$	10^{-26}
2 (Rastrigin)	2	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, 1)$	10^{-26}
	3	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, 1, 1)$	10^{-26}
	5	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, ..., 1)$	10^{-26}
	10	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, ..., 1)$	10^{-26}
	20	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, ..., 1)$	10^{-26}
3 (Schwefel)	2	$\alpha_l = 0.2; \alpha_u = 1.0$	$(270, 370)$	10^{-26}
	3	$\alpha_l = 0.2; \alpha_u = 1.0$	$(270, 370, 270)$	10^{-26}
	5	$\alpha_l = 0.2; \alpha_u = 1.0$	$(270, 370, 270, 370, 270)$	10^{-26}
	10	$\alpha_l = 0.2; \alpha_u = 1.0$	(1,...,1)	10^{-26}
	20	$\alpha_l = 0.2; \alpha_u = 1.0$	(1,...,1)	10^{-26}

Table B.2.: *Implementation details for Convergence Theory: Rate of Convergence of Rayleigh GAD*

Problem				
Name	n	Step-Size	Initial Point	τ
1 (Ackley)	2	$\alpha_l = 10^{-2}; \alpha_u = 10^{-1}$	$(0.4, 0.6)$	10^{-8}
	3	$\alpha_l = 10^{-4}; \alpha_u = 10^{-2}$	$(0.23, 8, 5)$	10^{-8}
	5	$\alpha_l = 10^{-4}; \alpha_u = 10^{-2}$	$(0.23, 8, 5, 1, 1)$	10^{-8}
	10	$\alpha_l = 10^{-4}; \alpha_u = 10^{-2}$	$(0.23, 8, 5, 1, 1, 1, 2, 3, 4, 5)$	10^{-8}
	20	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$[0.23, 8, 5, 1, 1, 1, 2, 3, 4, 5, 0.23, 8, 5, 1, 1, 1, 2, 3, 4, 5]$	10^{-8}
2 (Rastrigin)	2	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, 1)$	10^{-8}
	3	$\alpha_l = 10^{-4}; \alpha_u = 10^{-1}$	$(1, 1, 1)$	10^{-8}
	5	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, ..., 1)$	10^{-8}
	10	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, ..., 1)$	10^{-8}
	20	$\alpha_l = 10^{-3}; \alpha_u = 10^{-1}$	$(1, ..., 1)$	10^{-8}
3 (Schwefel)	2	$\alpha_l = 0.1; \alpha_u = 0.9$	$(270, 370)$	10^{-8}
	3	$\alpha_l = 0.1; \alpha_u = 0.9$	$(270, 370, 270)$	10^{-8}
	5	$\alpha_l = 0.1; \alpha_u = 0.9$	$(270, 370, 270, 370, 270)$	10^{-8}
	10	$\alpha_l = 0.1; \alpha_u = 0.9$	$(270, 370, ..., ..., ..., 270, 370)$	10^{-8}
	20	$\alpha_l = 0.1; \alpha_u = 0.9$	$(270, 370, ..., ..., ..., 270, 370)$	10^{-8}

Table B.3.: *Implementation details Global Optimisation Approach 1: Global Descent Functions*

Problem				
Name	n	Step-Size	Initial Point	τ
1 (Ackley)	2	—	$(30, 30)$	10^{-3}
	3	—	$(30, 30, 30)$	10^{-3}
	5	—	$(30, ..., ..., 30)$	10^{-3}
2 (Rastrigin)	2	—	$(5, 5)$	10^{-8}
	3	—	$(5, 5, 5)$	10^{-8}
	5	—	$(5, ..., 5)$	10^{-8}
3 (Schwefel)	2	—	$(270, 370)$	10^{-8}
	3	—	$(270, 370, 270)$	10^{-8}
	5	—	$(270, 370, 270, 370, 270)$	10^{-8}

Table B.4.: *Implementation details Global Optimisation Approach 2: Unbounded Line Searches*

Problem				
Name	n	Step-Size	Initial Point	τ
1 (Ackley)	2	—	$(30, 30)$	10^{-3}
	3	—	$(30, 30, 30)$	10^{-3}
	5	—	$(30, ..., ..., 30)$	10^{-3}
2 (Rastrigin)	2	—	$(5, 5)$	10^{-8}
	3	—	$(5, 5, 5)$	10^{-8}
	5	—	$(5, ..., 5)$	10^{-8}
3 (Schwefel)	2	—	$(270, 370)$	10^{-8}
	3	—	$(270, 370, 270)$	10^{-8}
	5	—	$(270, 370, 270, 370, 270)$	10^{-8}

Table B.5.: *Implementation details for Perfomance and Accuracy of Global Optimisation Using Global Descent Functions at Saddle Points – L-J Clusters*

N	energy (E_{min})	reference	Initial point $X_0 \in \mathbb{R}^{3N}$
2	-1.0000	[44, 46, 47]	$(1, 0.2, 3, 0.4, 0.5, 0.6)$
3	-3.0000	[44, 46, 47]	$(1, 0.7, 0.6, 0.3, 1, 0.2, 0.2, 0.3, 1)$
4	-6.0000	[44, 46, 47]	$(-0.3616353090, 0.0439914505, 0.5828840628)$
5	-9.103852	[44, 46, 47]	$(-0.2604720088, 0.7363147287, 0.4727061929)$

Table B.6.: *Implementation details for Perfomance and Accuracy of Global Optimisation Using GAD "Chaining" and Unbounded Line Searches – L-J Clusters*

N	energy (E_{min})	reference	Initial point $X_0 \in \mathbb{R}^{3N}$
2	-1.0000	[44, 46, 47]	$(1, 0.2, 3, 0.4, 0.5, 0.6)$
3	-3.0000	[44, 46, 47]	$(1, 0.7, 0.6, 0.3, 1, 0.2, 0.2, 0.3, 1)$
4	-6.0000	[44, 46, 47]	$(-0.3616353090, 0.0439914505, 0.5828840628)$
5	-9.103852	[44, 46, 47]	$(-0.2604720088, 0.7363147287, 0.4727061929)$

C

Complete Results of Computational Experiments

Table C.1.: *Solution statistics for Natural GAD*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	46	47	47	2	0.9323	1
	3	77	78	78	2	2.3589	1
	5	20	21	21	2	1.1036	1
	10	11	12	12	2	1.6169	1
	20	19	20	20	2	6.5468	1
2 (Rastrigin)	2	41	42	42	2	2.5714	1
	3	112	113	113	2	9.4933	1
	5	38	39	39	2	8.4202	1
	10	23	24	24	24	10.2352	1
	20	36	37	37	37	31.8601	1
3 (Schwefel)	2	144	145	145	2	8.6552	1
	3	151	152	152	2	13.6435	1
	5	107	108	108	2	43.2022	1
	10	123	124	124	2	37.4938	1
	20	567	568	568	2	331.9135	1

Table C.2.: *Solution statistics for Rayleigh GAD*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	10	11	11	11	2.3566	1
	3	51	52	52	52	10.0858	1
	5	65	66	66	66	22.0817	1
	10	103	104	104	104	80.6100	1
	20	19	19	19	19	40.4132	1
2 (Rastrigin)	2	92	93	93	93	8.3496	1
	3	247	248	248	248	28.2447	1
	5	137	138	138	138	27.5040	1
	10	103	104	104	104	46.1129	1
	20	224	225	225	225	190.6497	1
3 (Schwefel)	2	157	158	158	158	14.9944	1
	3	172	173	173	173	25.2781	1
	5	167	168	168	168	43.3048	1
	10	174	174	174	174	86.7005	1
	20	185	186	186	186	228.4269	1

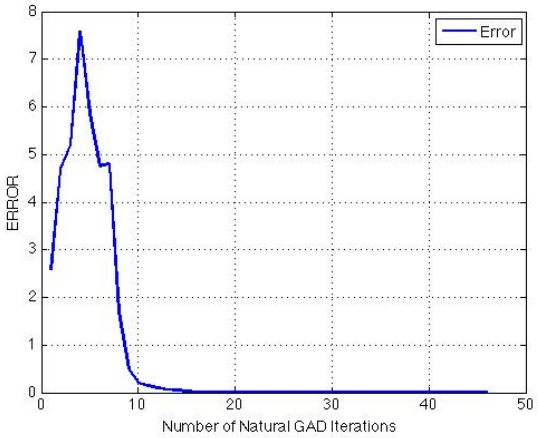
Table C.3.: *Results and solution statistics for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	12	625	701	152	14.7518	1
	3	9	2935	2857	116	86.8983	1
	5	5	2626	6188	52	92.9138	1
2 (Rastrigin)	2	1	29	31	4	2.4126	1
	3	1	60	62	4	6.6653	1
	5	1	15	17	4	3.9685	1
	10	1	31	33	4	11.5332	1

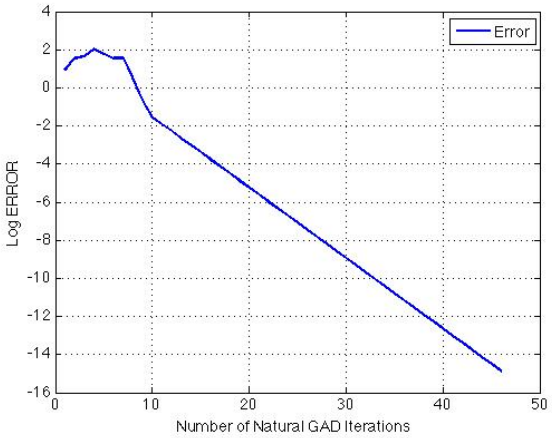
Table C.4.: *Results and solution statistics for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches*

Problem							
Name	n	N_{iter}	$N_{gradient}$	$N_{hessian}$	N_{eig}	T_{exec}	PI
1 (Ackley)	2	10	173	199	199	30.2412	1
	3	4	238	246	246	32.7275	1
	5	6	439	465	465	106.5899	1
2 (Rastrigin)	2	10	333	353	50	23.0188	1
	3	12	232	256	60	26.1143	1
	5	30	1016	1076	150	163.9899	1
	10	46	8588	8680	20	3798.9	1

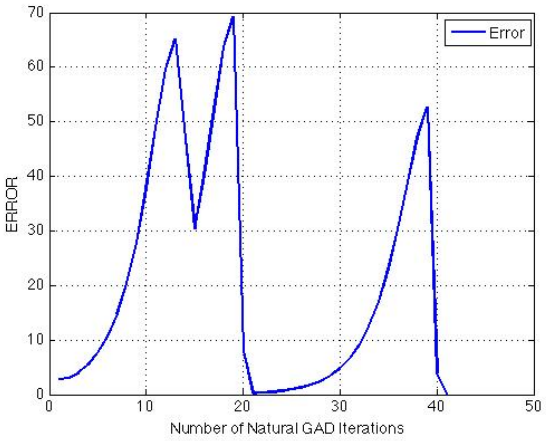
Figure C.1.: *Rate of Convergence Results for Natural GAD on Problems 1,2 and 3 for* $n = 2$



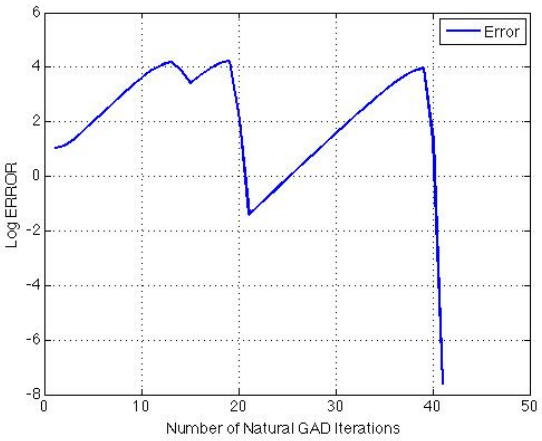
(a) Ackley Function, $n = 2$: Error vs. Iterations



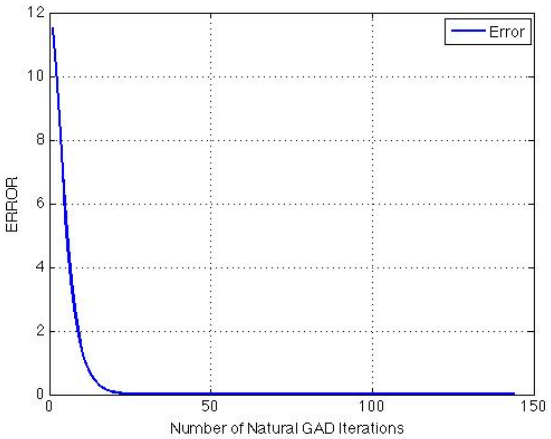
(b) Ackley Function, $n = 2$: Log Error vs. Iterations



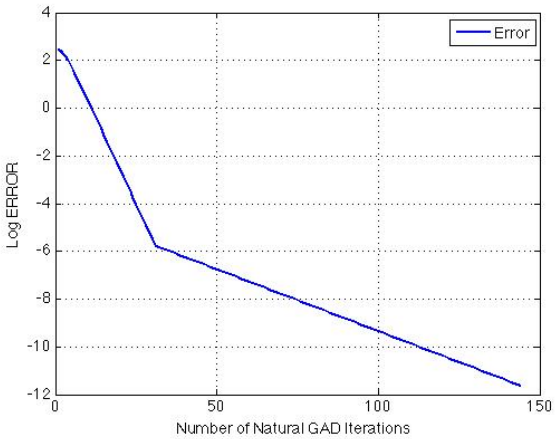
(c) Rastrigin Function, $n = 2$: Error vs. Iterations



(d) Rastrigin Function, $n = 2$: Log Error vs. Iterations

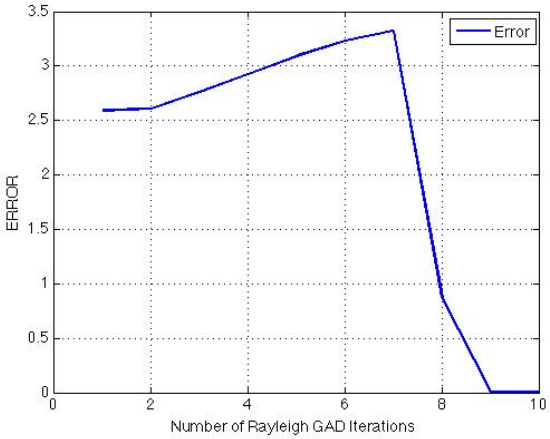


(e) Schwefel Function, $n = 2$: Error vs. Iterations

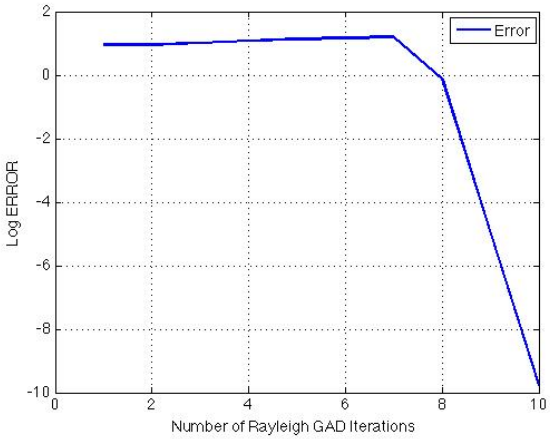


(f) Schwefel Function, $n = 2$: Log Error vs. Iterations

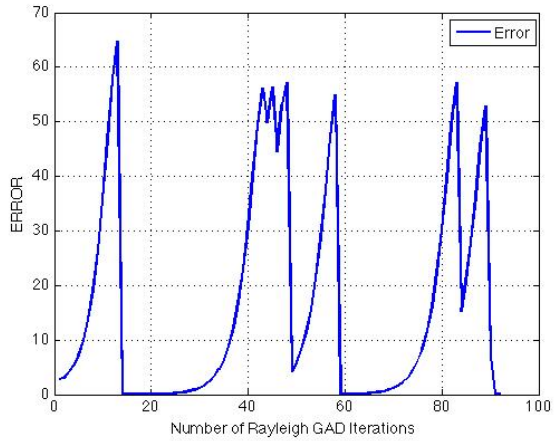
Figure C.2.: *Rate of Convergence Results for Rayleigh GAD on Problems 1,2 and 3 for* $n = 2$



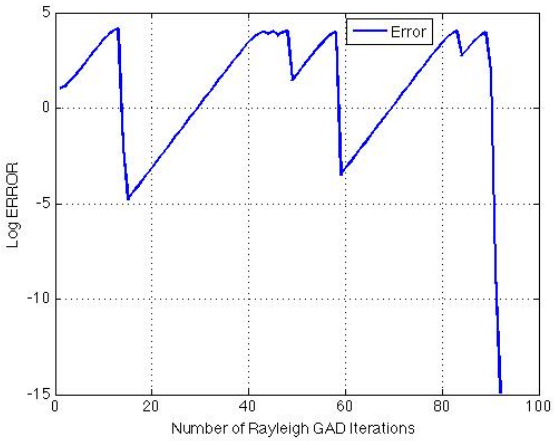
(a) Ackley Function, $n = 2$: Error vs. Iterations



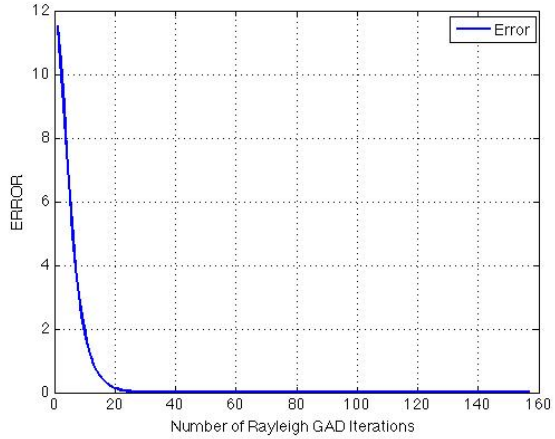
(b) Ackley Function, $n = 2$: Log Error vs. Iterations



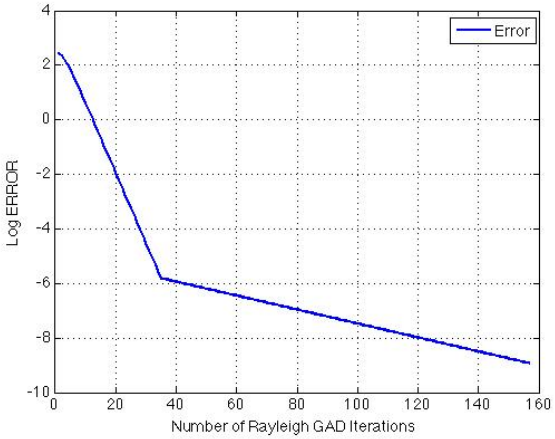
(c) Rastrigin Function, $n = 2$: Error vs. Iterations



(d) Rastrigin Function, $n = 2$: Log Error vs. Iterations

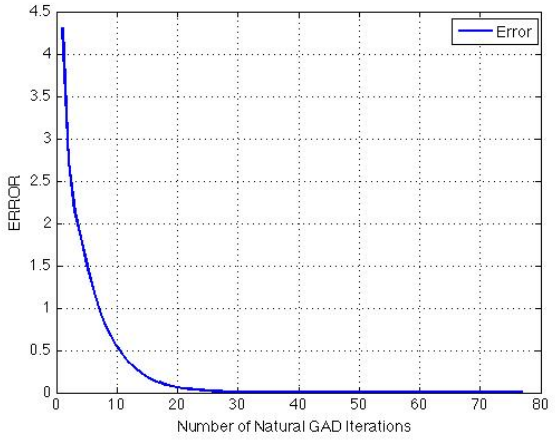


(e) Schwefel Function, $n = 2$: Error vs. Iterations

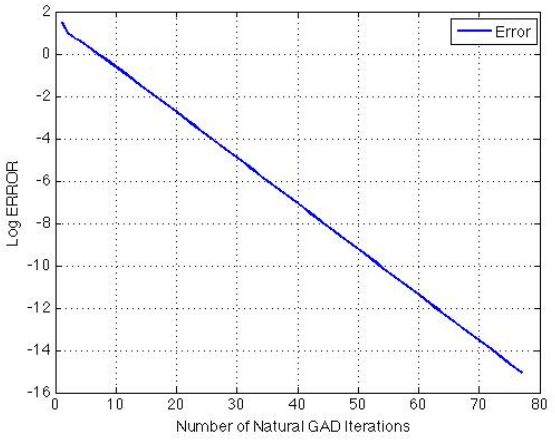


(f) Schwefel Function, $n = 2$: Log Error vs. Iterations

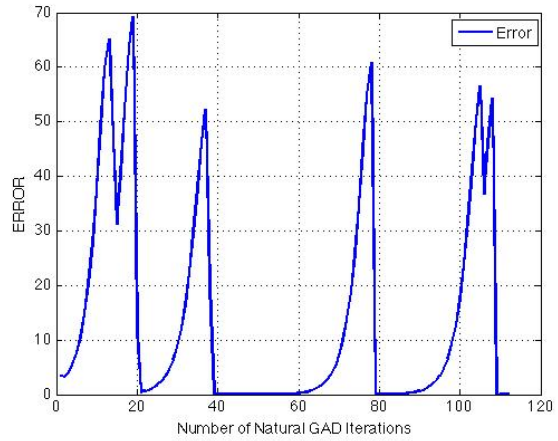
Figure C.3.: *Rate of Convergence Results for Natural GAD on Problems 1,2 and 3 for* $n = 3$



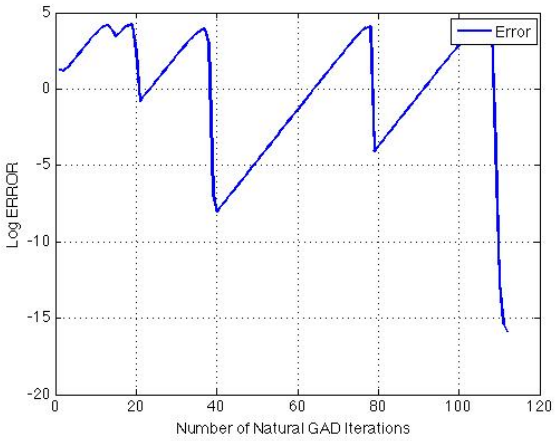
(a) Ackley Function, $n = 3$: Error vs. Iterations



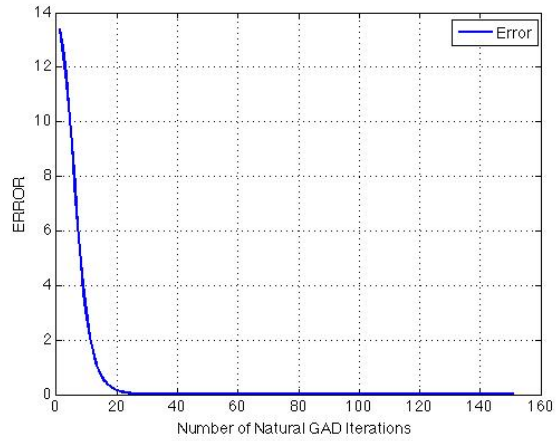
(b) Ackley Function, $n = 3$: Log Error vs. Iterations



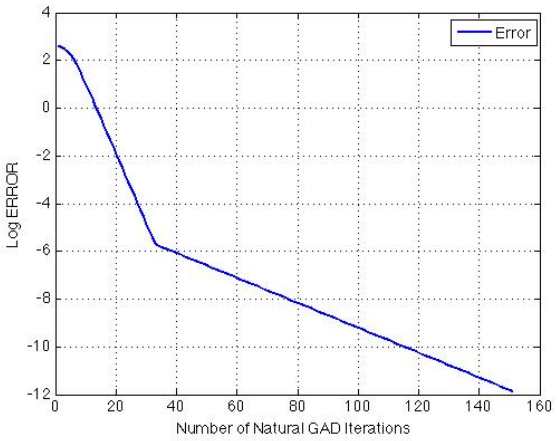
(c) Rastrigin Function, $n = 3$: Error vs. Iterations



(d) Rastrigin Function, $n = 3$: Log Error vs. Iterations

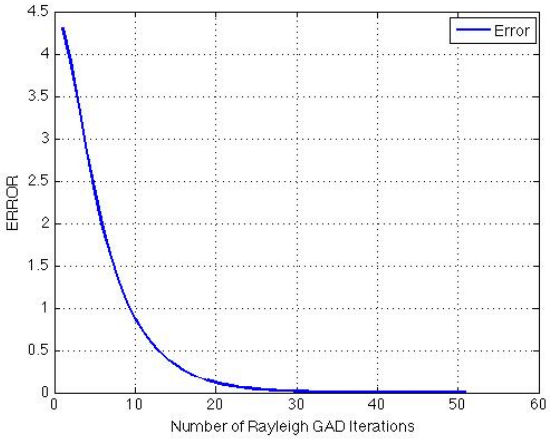


(e) Schwefel Function, $n = 3$: Error vs. Iterations

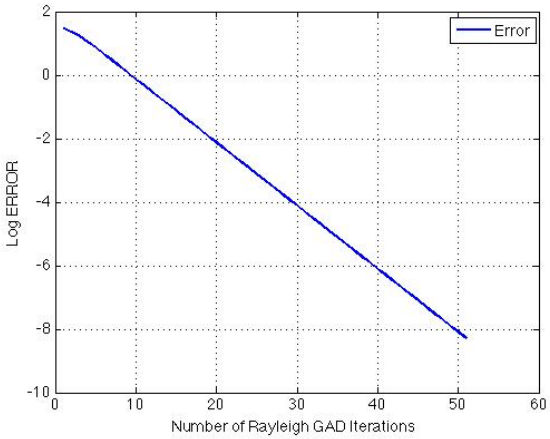


(f) Schwefel Function, $n = 3$: Log Error vs. Iterations

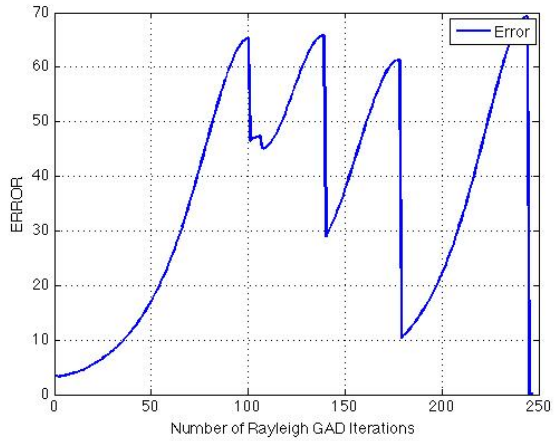
Figure C.4.: *Rate of Convergence Results for Rayleigh GAD on Problems 1,2 and 3 for* $n = 3$



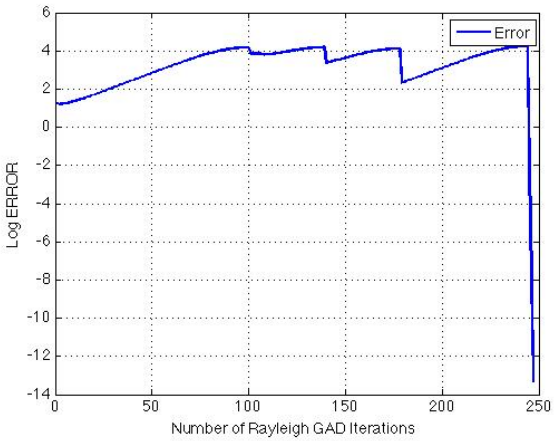
(a) Ackley Function, $n = 3$: Error vs. Iterations



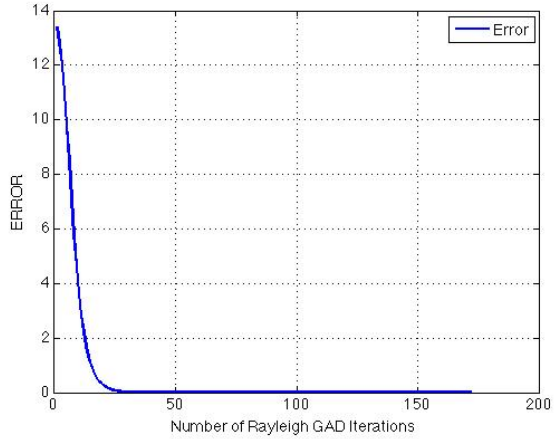
(b) Ackley Function, $n = 3$: Log Error vs. Iterations



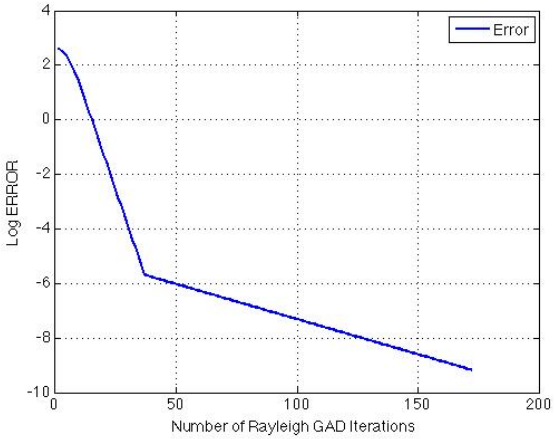
(c) Rastrigin Function, $n = 3$: Error vs. Iterations



(d) Rastrigin Function, $n = 3$: Log Error vs. Iterations

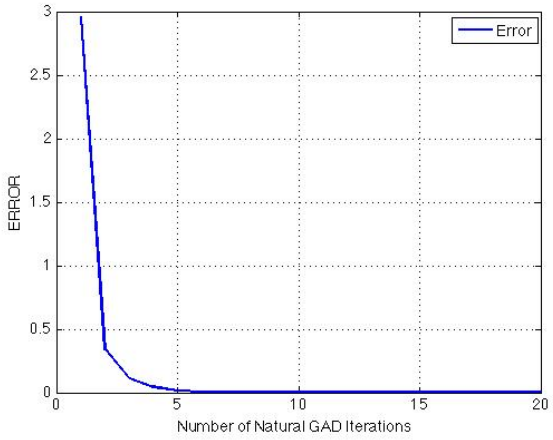


(e) Schwefel Function, $n = 3$: Error vs. Iterations

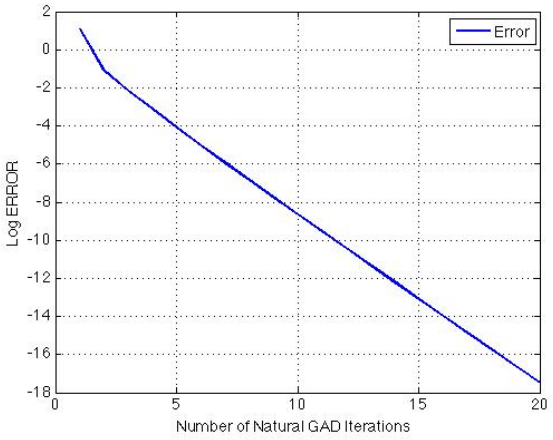


(f) Schwefel Function, $n = 3$: Log Error vs. Iterations

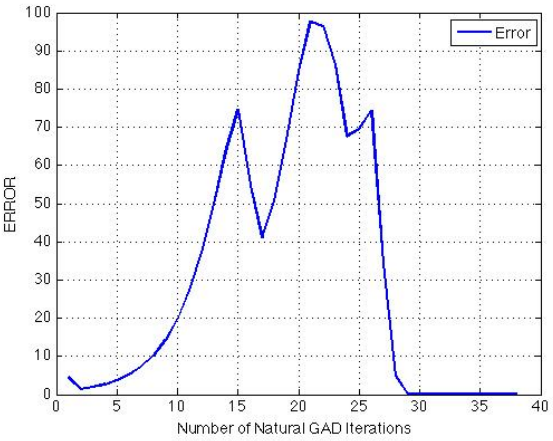
Figure C.5.: *Rate of Convergence Results for Natural GAD on Problems 1,2 and 3 for* $n = 5$



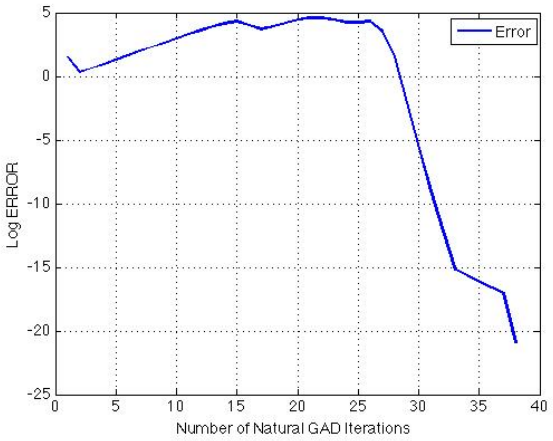
(a) Ackley Function, $n = 5$: Error vs. Iterations



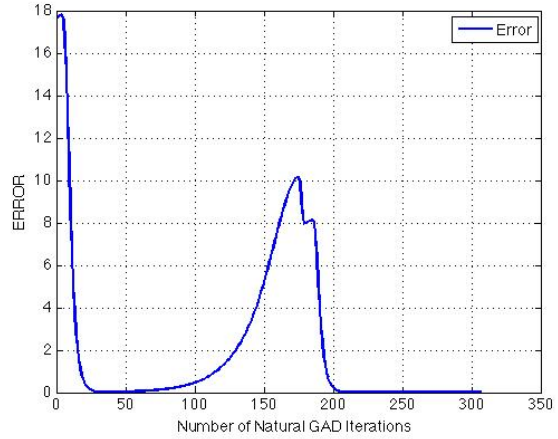
(b) Ackley Function, $n = 5$: Log Error vs. Iterations



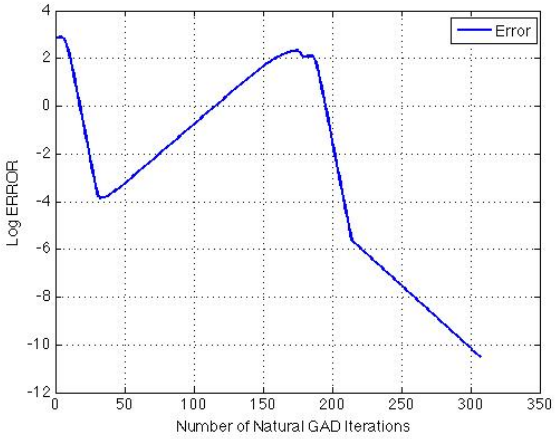
(c) Rastrigin Function, $n = 5$: Error vs. Iterations



(d) Rastrigin Function, $n = 5$: Log Error vs. Iterations

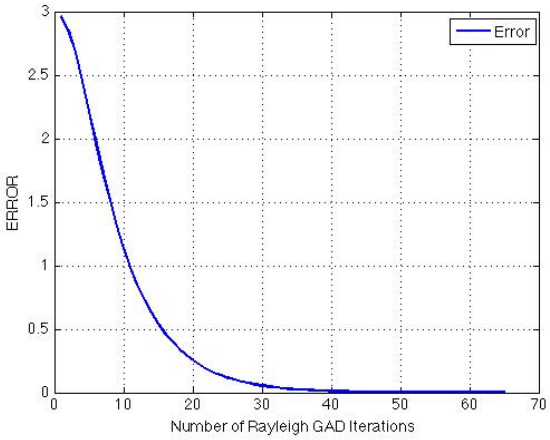


(e) Schwefel Function, $n = 5$: Error vs. Iterations

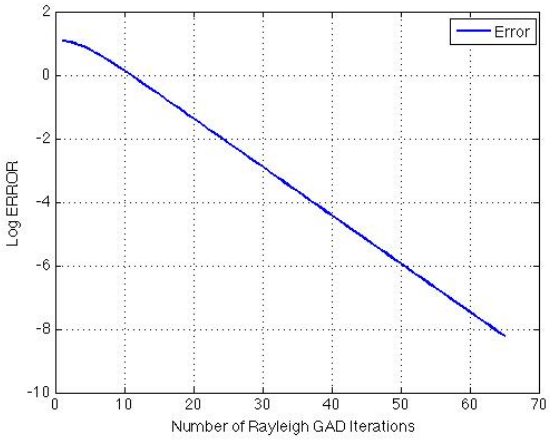


(f) Schwefel Function, $n = 5$: Log Error vs. Iterations

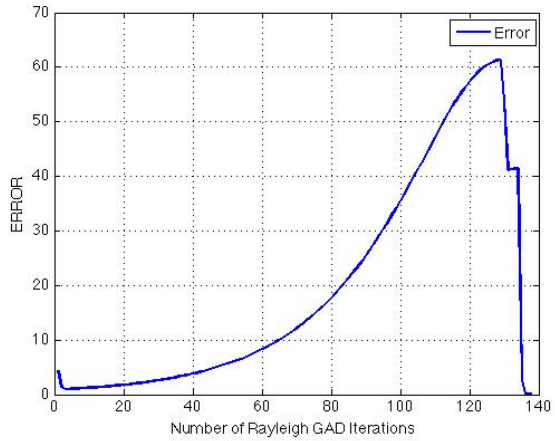
Figure C.6.: *Rate of Convergence Results for Rayleigh GAD on Problems 1,2 and 3 for* $n = 5$



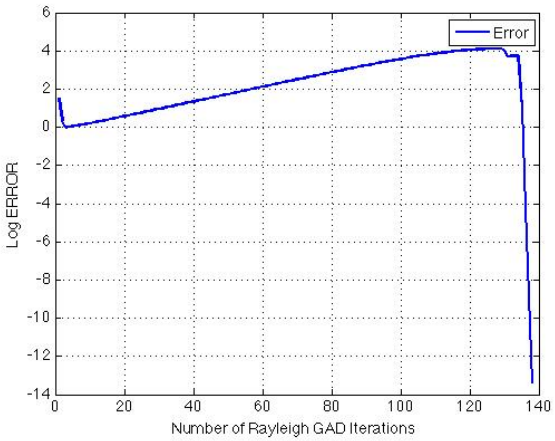
(a) Ackley Function, $n = 5$: Error vs. Iterations



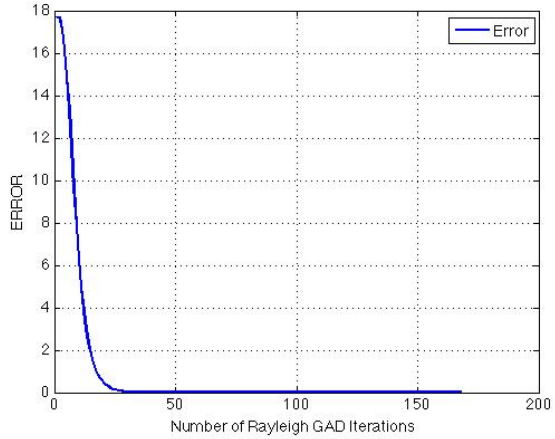
(b) Ackley Function, $n = 5$: Log Error vs. Iterations



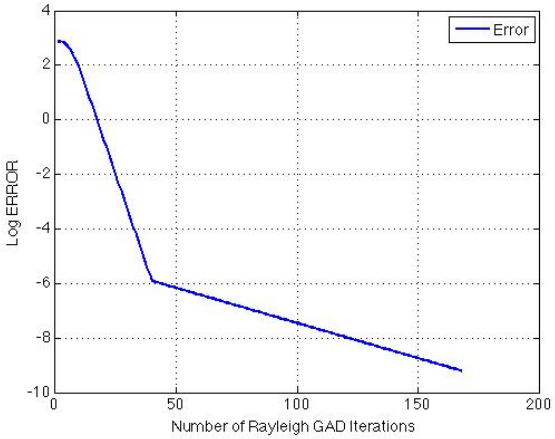
(c) Rastrigin Function, $n = 5$: Error vs. Iterations



(d) Rastrigin Function, $n = 5$: Log Error vs. Iterations

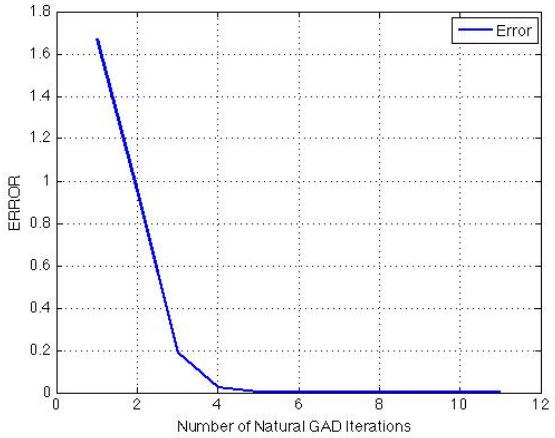


(e) Schwefel Function, $n = 5$: Error vs. Iterations

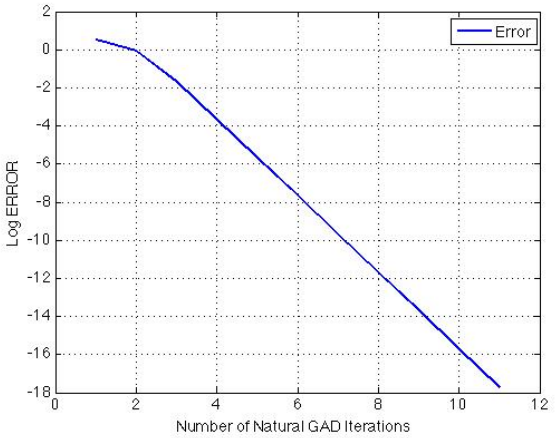


(f) Schwefel Function, $n = 5$: Log Error vs. Iterations

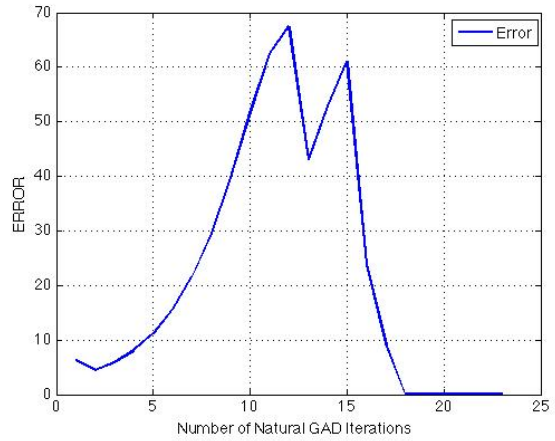
Figure C.7.: *Rate of Convergence Results for Natural GAD on Problems 1,2 and 3 for* $n = 10$



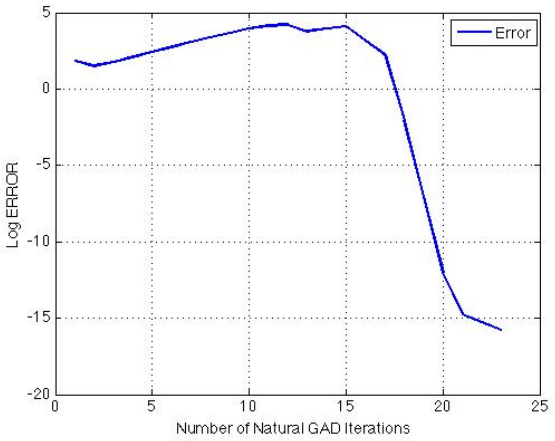
(a) Ackley Function, $n = 10$: Error vs. Iterations



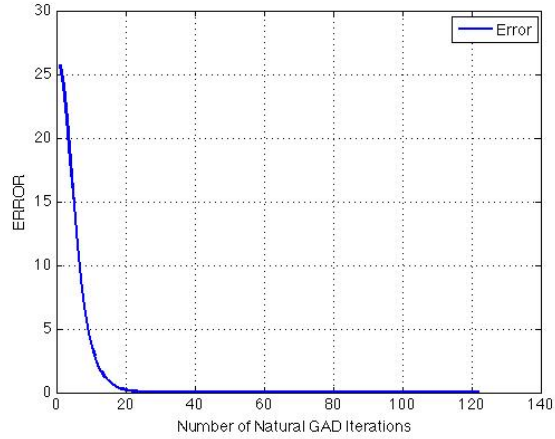
(b) Ackley Function, $n = 10$: Log Error vs. Iterations



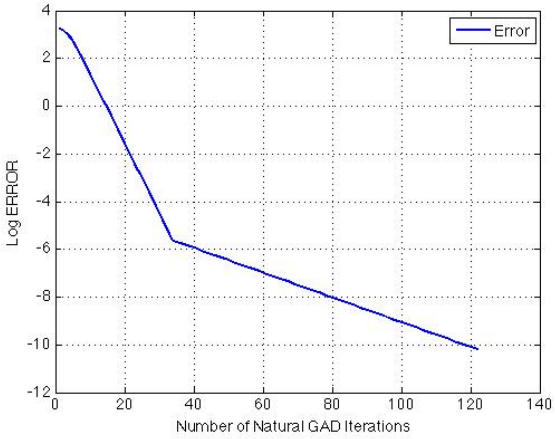
(c) Rastrigin Function, $n = 10$: Error vs. Iterations



(d) Rastrigin Function, $n = 10$: Log Error vs. Iterations

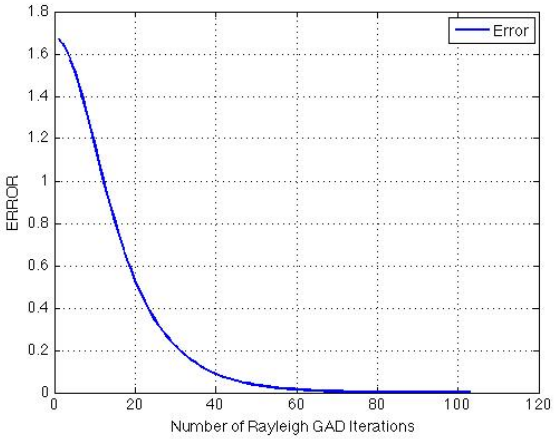


(e) Schwefel Function, $n = 10$: Error vs. Iterations

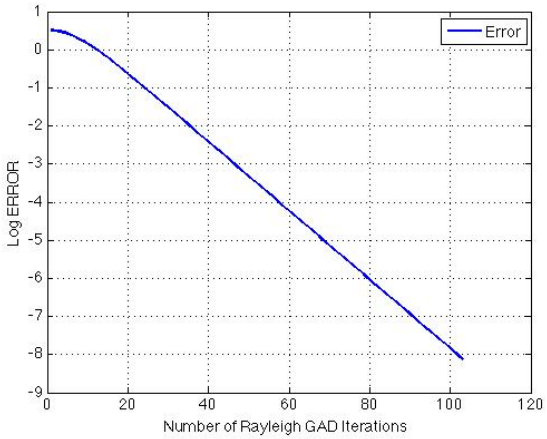


(f) Schwefel Function, $n = 10$: Log Error vs. Iterations

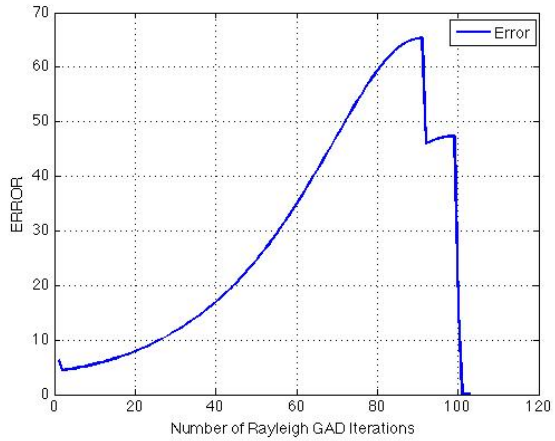
Figure C.8.: *Rate of Convergence Results for Rayleigh GAD on Problems 1,2 and 3 for* $n = 10$



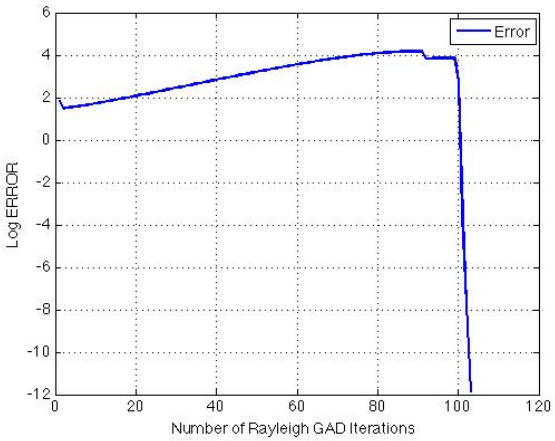
(a) Ackley Function, $n = 10$: Error vs. Iterations



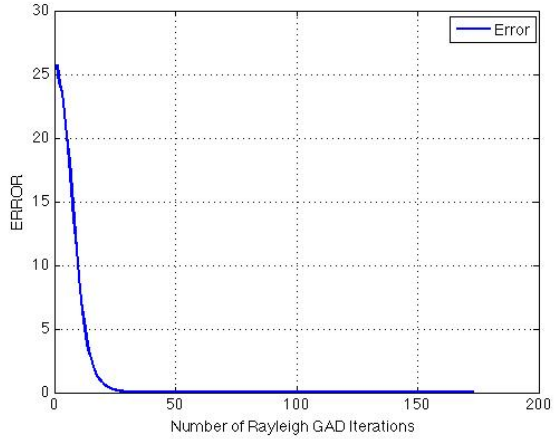
(b) Ackley Function, $n = 10$: Log Error vs. Iterations



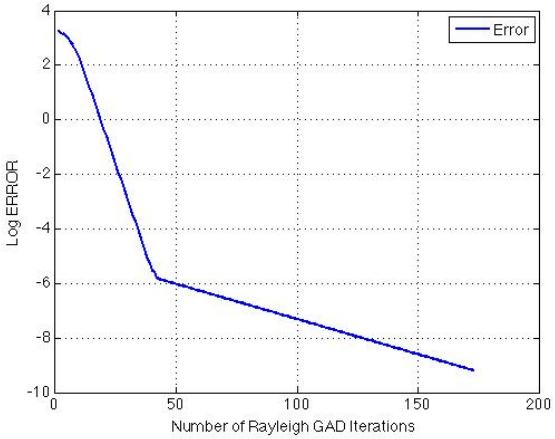
(c) Rastrigin Function, $n = 10$: Error vs. Iterations



(d) Rastrigin Function, $n = 10$: Log Error vs. Iterations

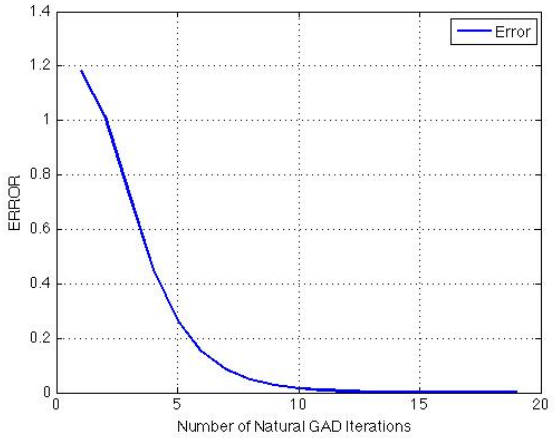


(e) Schwefel Function, $n = 10$: Error vs. Iterations

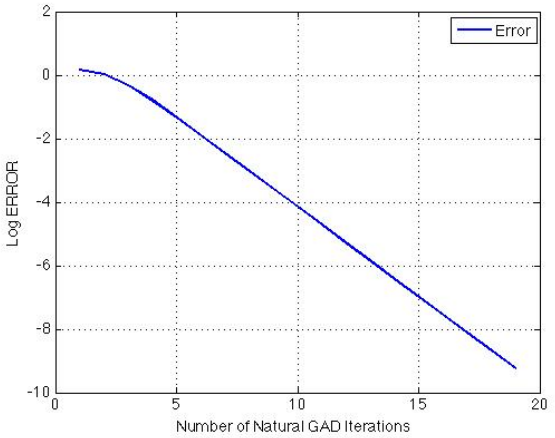


(f) Schwefel Function, $n = 10$: Log Error vs. Iterations

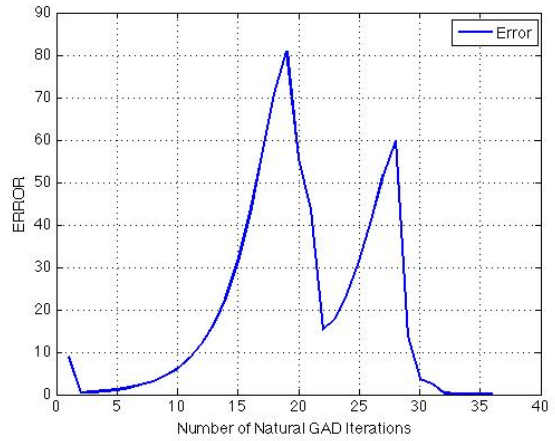
Figure C.9.: *Rate of Convergence Results for Natural GAD on Problems 1,2 and 3 for* $n = 20$



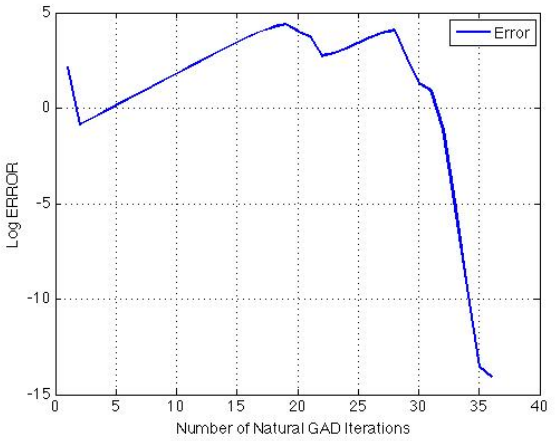
(a) Ackley Function, $n = 20$: Error vs. Iterations



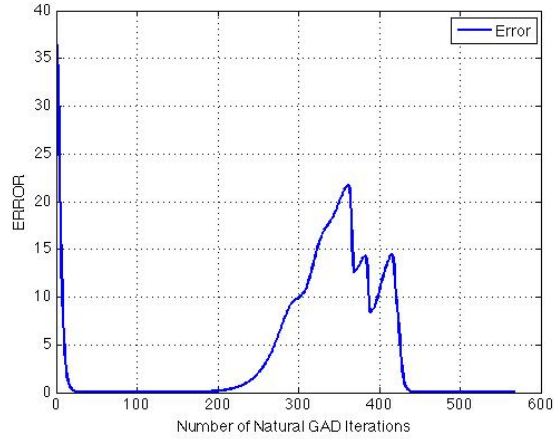
(b) Ackley Function, $n = 20$: Log Error vs. Iterations



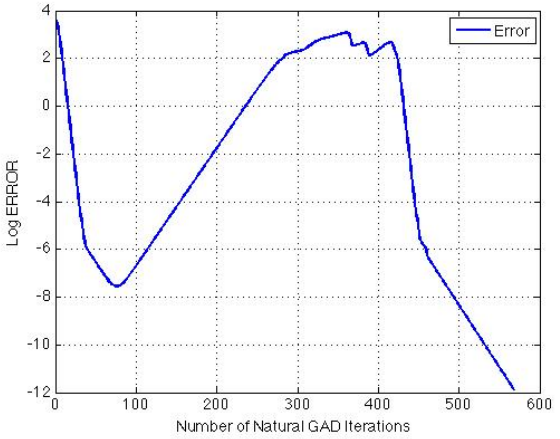
(c) Rastrigin Function, $n = 20$: Error vs. Iterations



(d) Rastrigin Function, $n = 20$: Log Error vs. Iterations

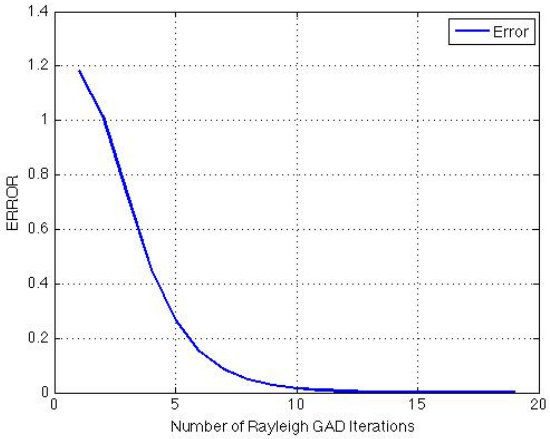


(e) Schwefel Function, $n = 20$: Error vs. Iterations

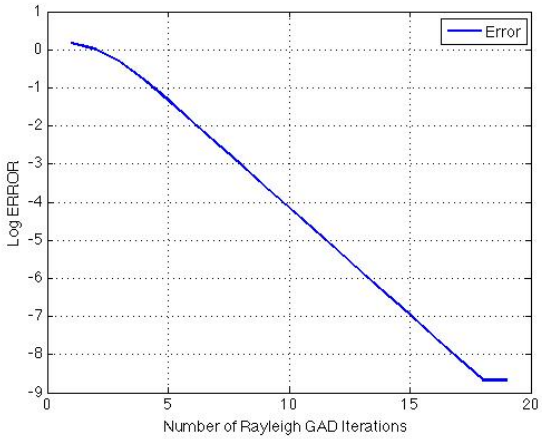


(f) Schwefel Function, $n = 20$: Log Error vs. Iterations

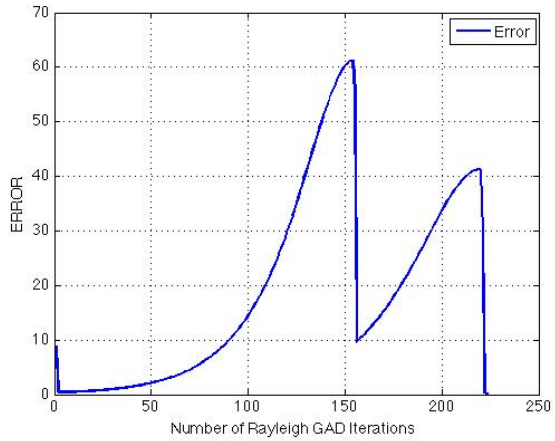
Figure C.10.: *Rate of Convergence Results for Rayleigh GAD on Problems 1,2 and 3 for* $n = 20$



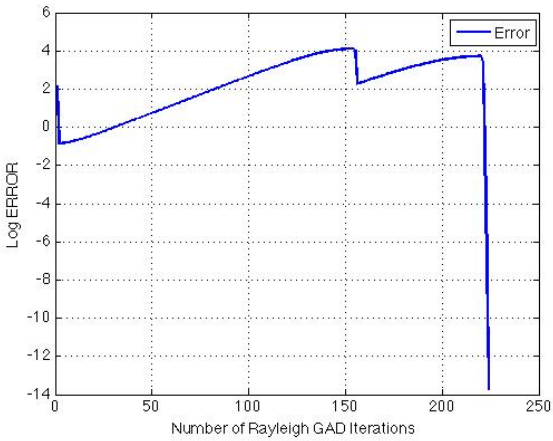
(a) Ackley Function, $n = 20$: Error vs. Iterations



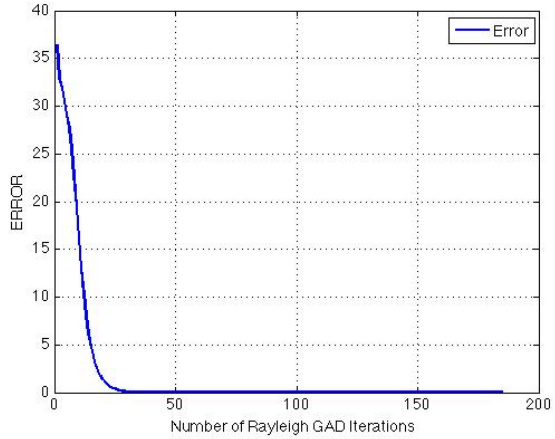
(b) Ackley Function, $n = 20$: Log Error vs. Iterations



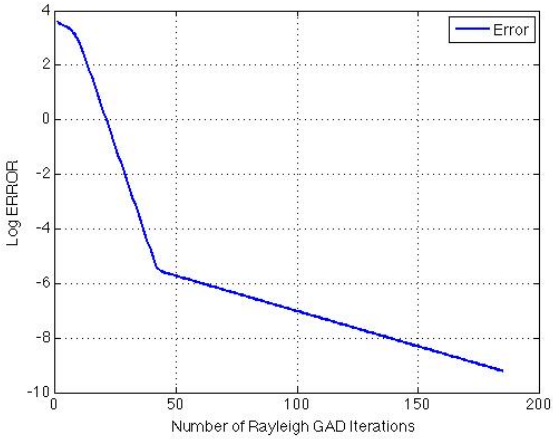
(c) Rastrigin Function, $n = 20$: Error vs. Iterations



(d) Rastrigin Function, $n = 20$: Log Error vs. Iterations

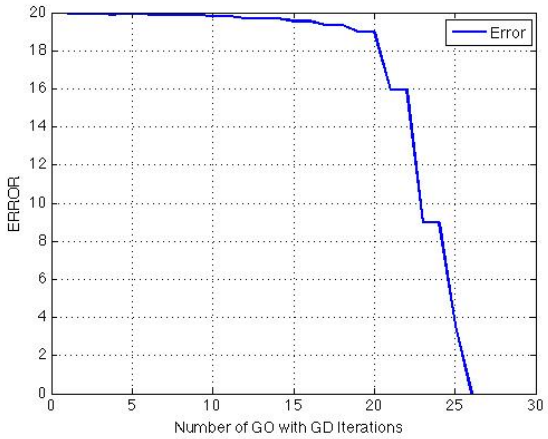


(e) Schwefel Function, $n = 20$: Error vs. Iterations

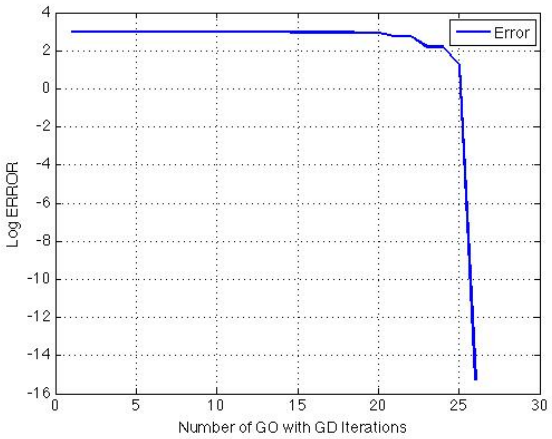


(f) Schwefel Function, $n = 20$: Log Error vs. Iterations

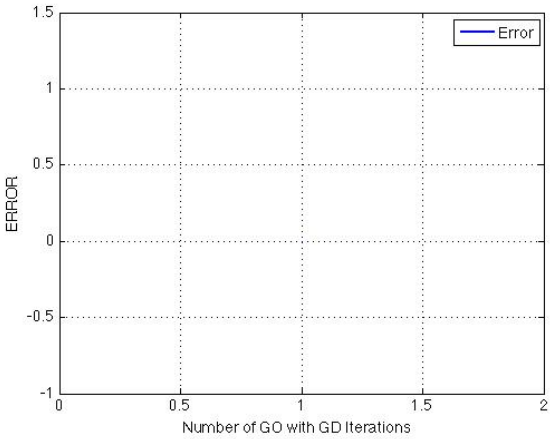
Figure C.11.: *Rate of Convergence Results for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions on Problems 1 and 2 for $n = 2$*



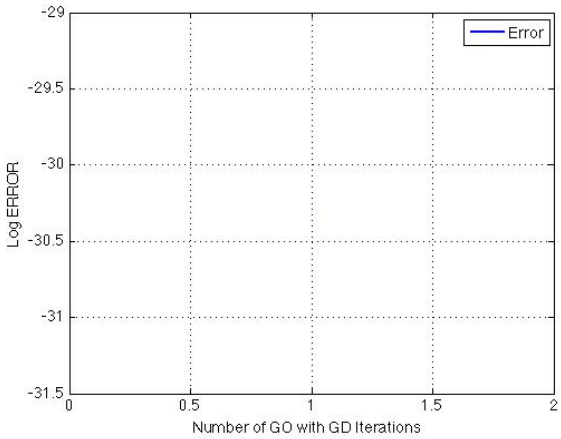
(a) Ackley Function, $n = 2$: Error vs. Iterations



(b) Ackley Function, $n = 2$: Log Error vs. Iterations

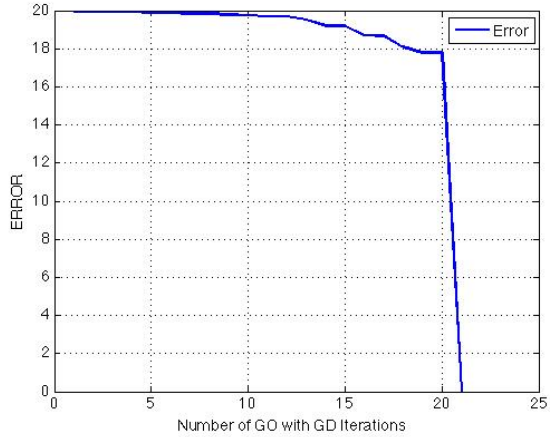


(c) Rastrigin Function, $n = 2$: Error vs. Iterations (Done in 1 iteration)

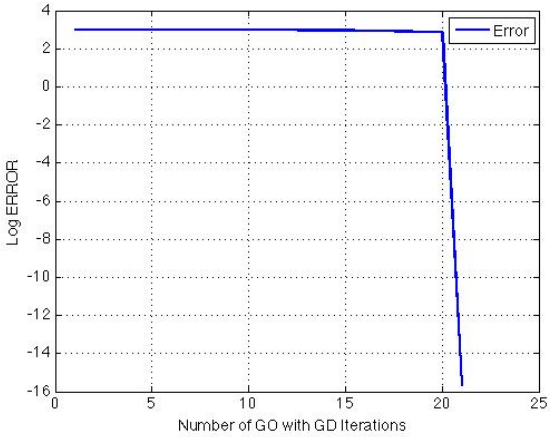


(d) Rastrigin Function, $n = 2$: Log Error vs. Iterations (Done in 1 iteration)

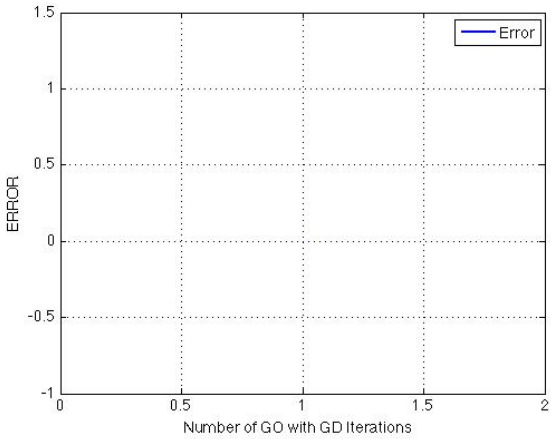
Figure C.12.: *Rate of Convergence Results for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions on Problems 1 and 2 for $n = 3$*



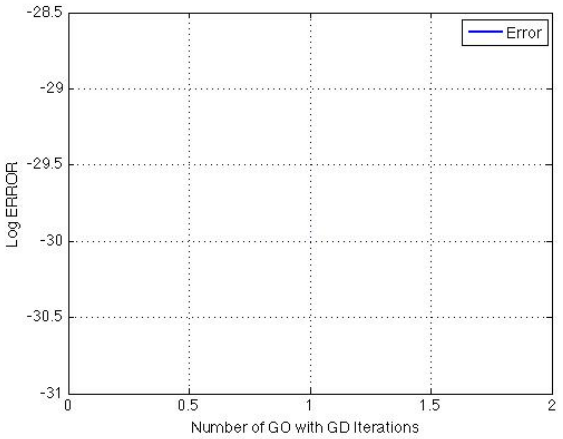
(a) Ackley Function, $n = 3$: Error vs. Iterations



(b) Ackley Function, $n = 3$: Log Error vs. Iterations

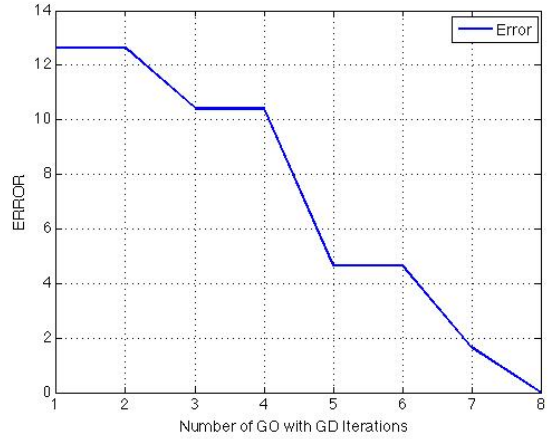


(c) Rastrigin Function, $n = 3$: Error vs. Iterations (Done in 1 iteration)

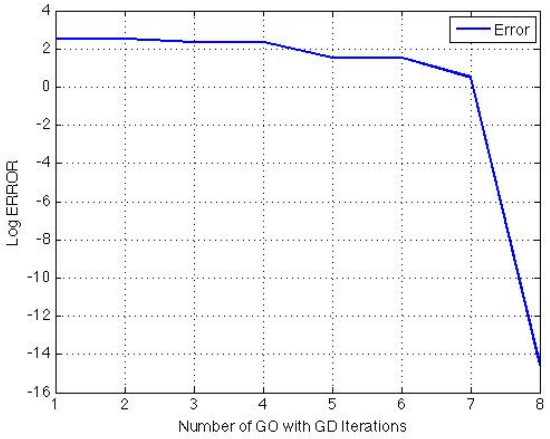


(d) Rastrigin Function, $n = 3$: Log Error vs. Iterations (Done in 1 iteration)

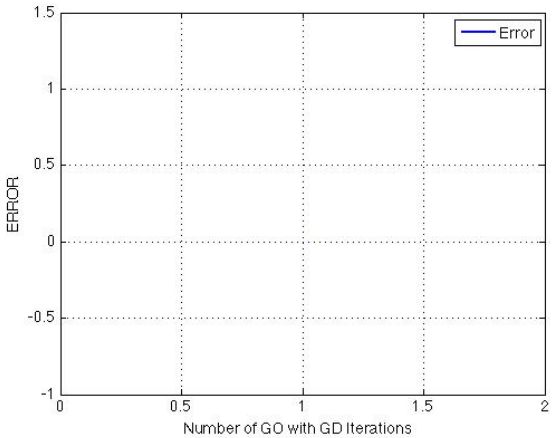
Figure C.13.: *Rate of Convergence Results for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions on Problems 1 and 2 for $n = 5$*



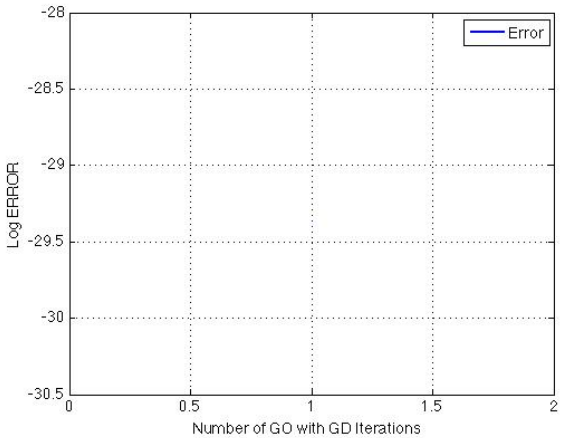
(a) Ackley Function, $n = 5$: Error vs. Iterations



(b) Ackley Function, $n = 5$: Log Error vs. Iterations

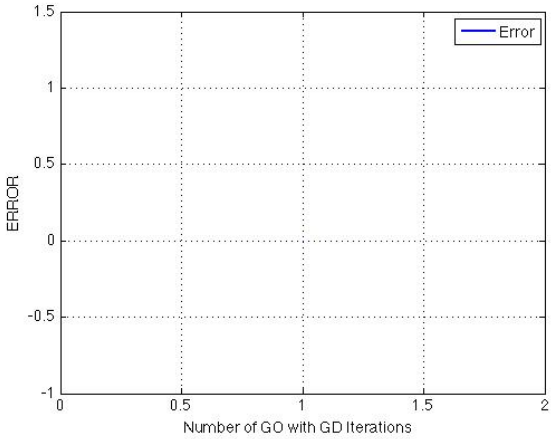


(c) Rastrigin Function, $n = 5$: Error vs. Iterations (Done in 1 iteration)

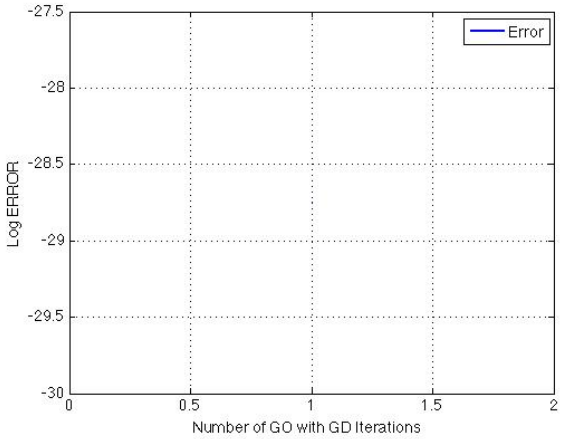


(d) Rastrigin Function, $n = 5$: Log Error vs. Iterations (Done in 1 iteration)

Figure C.14.: *Rate of Convergence Results for Global Optimisation Approach 1 - Saddle Points and Global Descent Functions on Problem 2 for $n = 10$*



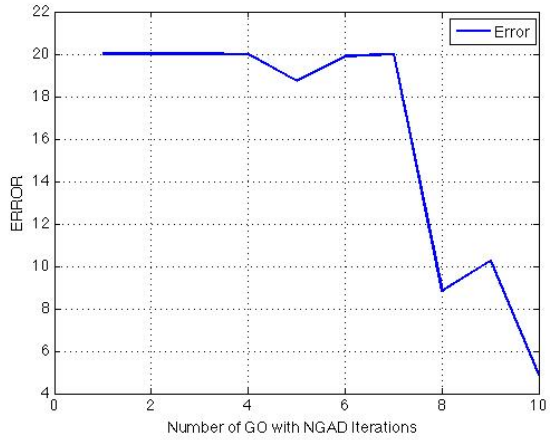
(a) Rastrigin Function, $n = 10$: Error vs. Iterations (Done in 1 iteration)



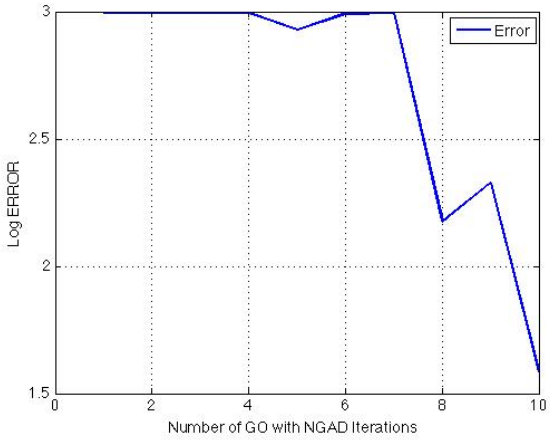
(b) Rastrigin Function, $n = 10$: Log Error vs. Iterations (Done in 1 iteration)

C.1. Other Results

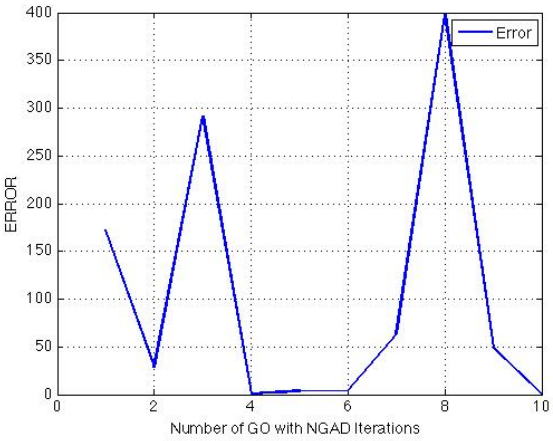
Figure C.15.: *Rate of Convergence Results for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches on Problems 1 and 2 for* $n = 2$



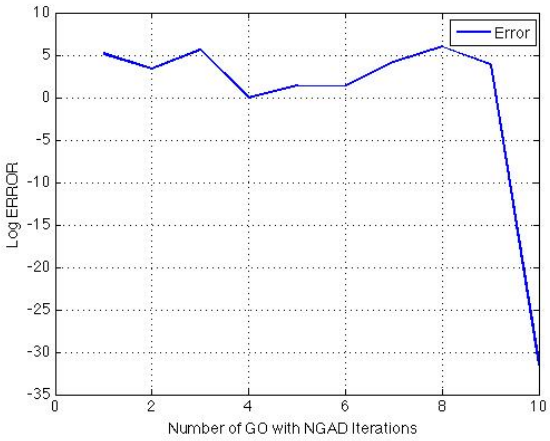
(a) Ackley Function, $n = 2$: Error vs. Iterations



(b) Ackley Function, $n = 2$: Log Error vs. Iterations

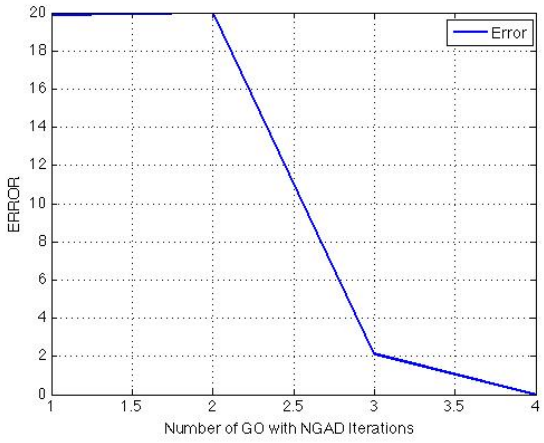


(c) Rastrigin Function, $n = 2$: Error vs. Iterations

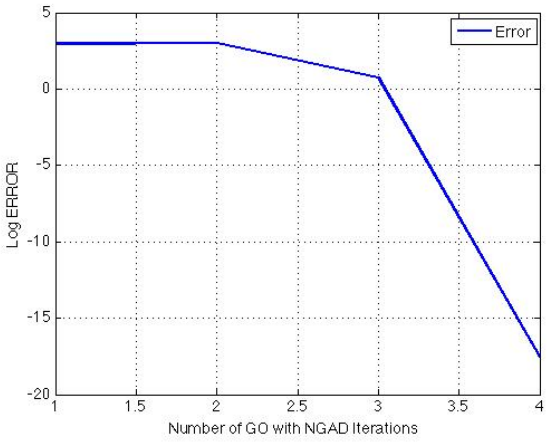


(d) Rastrigin Function, $n = 2$: Log Error vs. Iterations

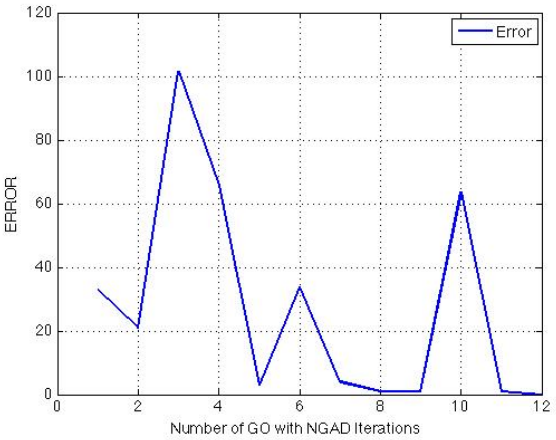
Figure C.16.: *Rate of Convergence Results for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches on Problems 1 and 2 for $n = 3$*



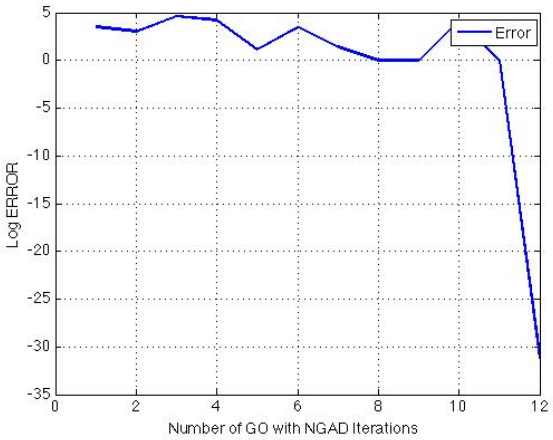
(a) Ackley Function, $n = 3$: Error vs. Iterations



(b) Ackley Function, $n = 3$: Log Error vs. Iterations

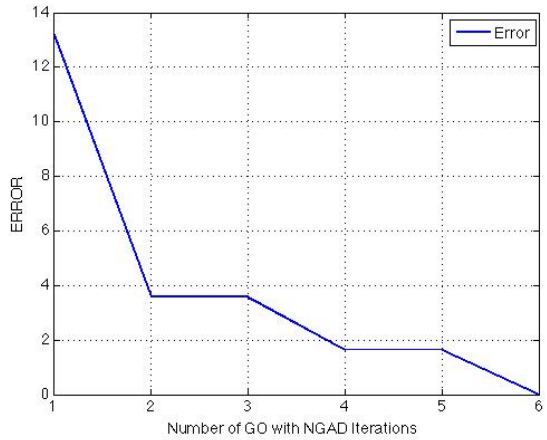


(c) Rastrigin Function, $n = 3$: Error vs. Iterations

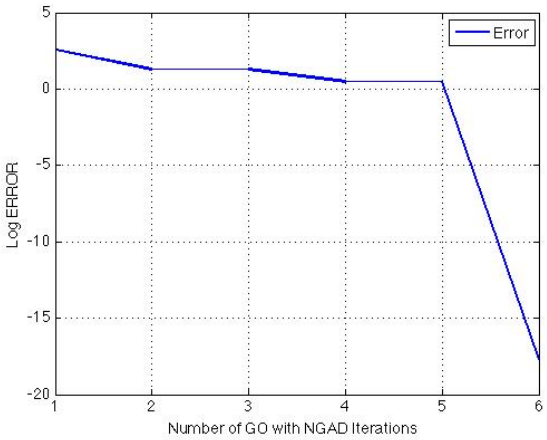


(d) Rastrigin Function, $n = 3$: Log Error vs. Iterations

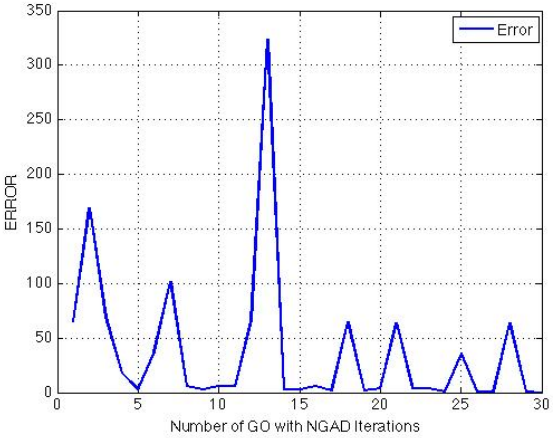
Figure C.17.: *Rate of Convergence Results for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches on Problems 1 and 2 for* $n = 5$



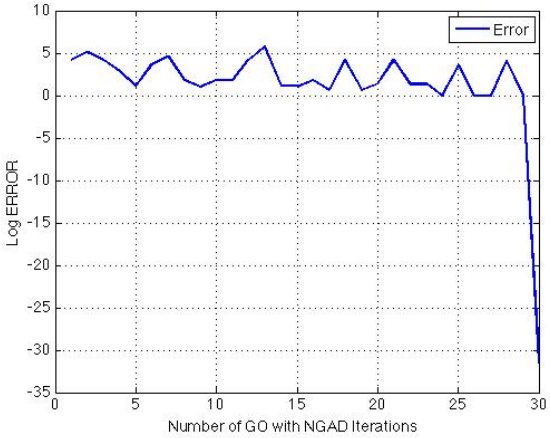
(a) Ackley Function, $n = 5$: Error vs. Iterations



(b) Ackley Function, $n = 5$: Log Error vs. Iterations

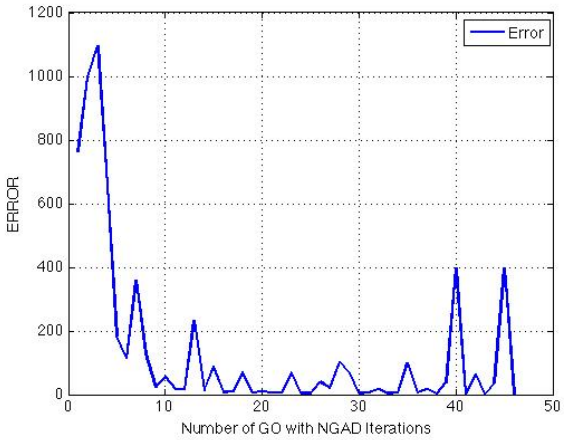


(c) Rastrigin Function, $n = 5$: Error vs. Iterations

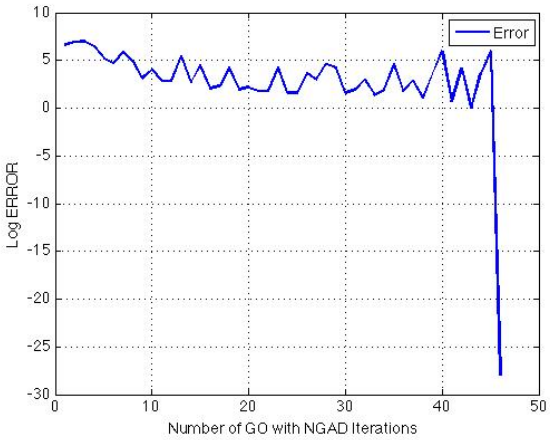


(d) Rastrigin Function, $n = 5$: Log Error vs. Iterations

Figure C.18.: *Rate of Convergence Results for Global Optimisation Approach 2 - Saddle Points and Unbounded Line Searches on Problem 1 for* $n = 10$



(a) Rastrigin Function, $n = 10$: Error vs. Iterations



(b) Rastrigin Function, $n = 10$: Log Error vs. Iterations

Bibliography

[1] Morris Kline. *Mathematics: The Loss of Certainty*, chapter 3: The Mathematization of Science, page 66. Oxford University Press, 1st edition, January 1983.

[2] Edwin K. P. Chong and Stanislaw H. Zak. *An Introduction to Optimization*. Wiley, 4th edition, 2013.

[3] INFORMS Computing Society. The nature of mathematical programming. *Mathematical Programming Glossary*, 2010. Online; `http://glossary.computing.society.informs.org/index.php?page=nature.html` Accessed on September 6, 2013.

[4] Leo Liberti. *Introduction to Global Optimisation*. LIX, Ecole Polytechnique, Palaiseau, France, February 2011.

[5] Chi-Kong Ng, Duan Li, and Lian-Sheng Zhang. Global descent method for global optimisation. *SIAM Journal on Optimization*, pages 3061–3184, 2010.

[6] Hannes Jnsson Graeme Henkelman, Gsli Jhannesson. Methods for finding saddle points and minimum energy paths. *Theoretical Methods in Condensed Phase Chemistry: Progress in Theoretical Chemistry and Physics 5*, pages 269–302, 2002.

[7] E Weinan and Xiang Zhou. The gentlest ascent dynamics. Technical report, Princeton University, Brown University, New York, 2011.

[8] D J Wales. Energy landscapes with application to cluster molecules an glasses. Technical report, Cambridge University, Cambridge, 2003.

[9] E Weinan and Amit Samanta. Atomistic simulation of rare events using gentlest ascent dynamics. Technical report, Princeton University, New Jersey, USA, 2011.

[10] C.T.J. Dodson. *Lecture Notes for Curves and Surfaces Module*. University of Manchester, Manchester, United Kingdom, 2007. [Online (`http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.142.3019`); his PDF teaching notes accessed on September 6, 2013].

[11] Christian Bar. *Elementary Differential Geometry*. Cambridge University Press, 1st edition, 2010.

[12] Alyn Rockwood and Peter Chambers. *Introduction to Curves and Surfaces*. ACM SIGGRAPH, New Orleans, 1996. [Online (`http://www.inf.ed.ac.uk/teaching/courses/cg/Web/intro_to_curves.pdf`); accessed on September 6, 2013].

[13] J.F. Talbert and H.H. Heng. *Additional Mathematics: Pure and Applied*. Pearson Education, 6th edition, 1995.

[14] Henrik Schlichtkrull. *Curves and Surfaces: Lecture Notes for Geometry 1*. University of Copenhagen, Copenhagen, 2011. [Online (`http://www.math.ku.dk/noter/filer/geom1.pdf`); accessed on September 6, 2013].

[15] Yukio Matsumoto. *An Introduction to Morse Theory.* American Mathematical Society, Providence, Rhode Island, 2002.

[16] Vijay Natarajan. *Lecture 3: Introdution to Morse Theory.* Indian Institute of Science, Bangalore, 2010. [Online (`http://drona.csa.iisc.ernet.in/~vijayn/courses/TopoForVis/notes/Notes03_morsetheory.pdf`); accessed on September 6, 2013].

[17] Arthur Cayley. On contour and slope lines. *The Philosophical Magazine 18 (120)*, pages 264–268, 1859. Online; `http://www.maths.ed.ac.uk/~aar/papers/cayleyconslo.pdf` Accessed on September 6, 2013.

[18] James Clerk Maxwell. On hills and dales. *The Philosophical Magazine 40 (269)*, pages 421–427, 1870. Online; `http://www.maths.ed.ac.uk/~aar/surgery/hilldale.pdf` Accessed on September 6, 2013.

[19] Paul Dawkins. *Differential Equations.* Lamar University, Beaumont, Texas, United States, 2007. [Online (`http://tutorial.math.lamar.edu/getfile.aspx?file=B,1,N`); his comprehensive PDF and web-based teaching notes accessed on September 6, 2013].

[20] Erwin Kreyzig. *Advanced Engineering Mathematics.* Wiley, 9th edition, 2006.

[21] Lawrence Shampine. *Numerical Solution of Ordinary Differential Equations.* Chapman and Hall, London, UK, 1994.

[22] Gerald Teschl. *Ordinary Differential Equations and Dynamical Systems.* American Mathematical Society, Vienna, Austria, 2012.

[23] Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. *Differential Equations, Dynamical Systems and an Introduction to Chaos.* Elsevier Academic Press, San Diego, California, USA, 2nd edition, 2004.

[24] Marsha Berger. *Differential Equations and Dynamical Systems.* New York University, New York, New York, USA, 2013. [Online (`http://www.cs.nyu.edu/courses/fall06/G22.2112-001/DiffEq.pdf`); PDF teaching notes accessed on September 6, 2013.].

[25] Renpu Ge. A filled function method for finding a global minimizer of a function of several variables. *Mathematical Programming 46*, pages 191–204, 1990.

[26] A. A. Goldstein and J. F. Price. On descent from minima. *Mathematics of Computation 25*, pages 569–574, 1971.

[27] A. V. Levy and A. Montalvo. Tunneling algorithm for the global minimisation of functions. *SIAM Journal on Scientific Computing 6*, pages 15–29, 1985.

[28] Renpu Ge. The theory of filled function method for finding global minimizers of nonlinearly constrained minimization problems. *Journal Computational Mathematics 5*, pages 1–9, 1987.

[29] Renpu Ge. Finding more and more solutions of a system of nonlinear equations. *Journal of Applied Mathematics and Computation 36*, pages 15–30, 1990.

[30] Renpu Ge and C.B. Huang. A continuous approach to nonlinear integer programming. *Journal of Applied Mathematics and Computation 34*, pages 36–60, 1989.

[31] C. Kanzow. Global optimization techniques for mixed complementarity problems. *Journal of Global Optimisation 16*, pages 1–21, 2000.

[32] C. A. Borghi and M. Fabbri. A combined technique for the global optimization of the inverse electromagnetic problem solution. *IEEE Transactions on Magnetics 33*, pages 1947–1950, 1997.

[33] C. A. Borghi, M. Fabbri, D. Casadei, and G. Serra. Reduction of the torque ripple in permanent magnet actuators by a multi-objective minimization technique. *IEEE Transactions on Magnetics 34*, pages 2869–2872, 1998.

[34] C. A. Borghi and M. Fabbri. Iron shielded mri optimization. *The European Physical Journal - Applied Physics 3*, pages 343–348, 1998.

[35] C. A. Borghi, M. Fabbri, P. Di Barba, and A. Savini. A comparative study of loneys solenoid by different techniques of global optimization. *International Journal of Applied Electromagnetics and Mechanics,10*, pages 417–423, 1999.

[36] E. F. Campana, G. Liuzzi, S. Lucidi, D. Peri, V. Piccialli, and A. Pinto. New global optimization methods for ship design problems. *Optimization and Engineering 10*, pages 533–555, 2009.

[37] Renpu Ge and Y.F. Qin. A class of filled functions for finding global minimizers of a function of several variables. *Journal of Optimization Theory and Applications*, pages 241–252, 1987.

[38] B Jansen, C Roos, and T Terlaky. A new algorithm for the computation of the smallest eigenvalue of a symmetric matrix and its eigenspace. Technical report, Delft University of Technology, Delft, Netherlands, 1995.

[39] Eric Temple Bell. *Men of Mathematics*, page 477. Simon and Schuster, 1st edition, 1986.

[40] Michelle Schatzma. *Numerical analysis: a mathematical introduction.* Clarendon Press, Oxford, UK, 2002.

[41] Richard L. Burden and J. Douglas Faires. *Numerical Analysis.* Brooks/Cole, 7th edition, 2001.

[42] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer-Verlag, Berlin, New York, 2nd edition, 2006.

[43] Marco Locatelli and Fabio Shoen. Fast global ptimization of difficult lennad-jones clusters. *Computational Optimization and Applications 21*, pages 55–70, 2002.

[44] David J. Wales and Jonathan P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *Journal of Physical Chemistry 101*, pages 5111–5116, 1997.

[45] Arnold Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review 39*, pages 407 – 460, 1997.

[46] M. R. Hoare and P. Pal. Advances in physics. *Advances in Physics 20*, page 161, 1971.

[47] M. R. Hoare and P. Pal. Nature (physicical sciences). *Nature (Physicical Sciences) 5*, page 230, 1971.

[48] Victor Vinnikov. We shall know: Hilberts apology. *The Mathematical Intelligencer 21*, pages 42–46, 1999.

[49] Adorio E. P. and Diliman U. P. Multivariate test functions library in c for unconstrained global optimization. Technical report, 2005.

[50] Molga M. and Smutnicki C. Test functions for optimization needs. Technical report, 2005.

[51] Back T. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Technical report, Informatik Centrum, Dortmund, Germany, 1996.

[52] Abdel-RahmanHedar. Global optimization test problems. Technical report, 2013. [Online (`http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm`); accessed on September 6, 2013].

[53] Pohlheim H. Geatbx examples: Examples of objective functions. Technical report, 2005. [Online (`http://www.geatbx.com/download/GEATbx_ObjFunExpl_v37.pdf`); accessed on September 6, 2013].

[54] Laguna M. and Marti R. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. Technical report, 2002. [Online (`http://www.uv.es/rmarti/paper/docs/global1.pdf`); accessed on September 6, 2013].

[55] Hartmut Pohlheim. Genetic and evolutionary algorithm toolbox for use with matlab (geatbx). Technical report, 2002. [Online (`http://www.pg.gda.pl/~mkwies/dyd/geadocu/fcnindex.html`); accessed on September 6, 2013].