University of Bergen
Department of Informatics

On Derivative-Free Optimisation Methods

Author: Pim Heeman
Supervisor: Jan-J. Rückmann

UNIVERSITY OF BERGEN
Faculty of Mathematics and Natural Sciences

May 2023

Abstract

As part of the field of mathematical optimisation, derivative-free optimisation is the study of optimisation methods that are not granted full access to the derivative of the objective function. In this master's thesis, three derivative-free optimisation methods known from the literature that do not use the derivatives have been studied, namely the Nelder–Mead method, the conditional trust-region method and the discrete gradient method. For each of these methods, besides recalling a description and a convergence statement, focus was given on providing motivation and background for increased understanding of the method without requiring specific prior knowledge in derivative-free optimisation. Different types of differentiability as total differentiability or subdifferentiability have been recalled for general usage in the understanding of those methods. As part of the description of the discrete gradient method, Wolfe's method for finding a minimum norm vector in a convex set is recalled, with a modified statement for proven convergence. Each of the methods are accompanied with an implementation for use with the MATLAB programming and numeric computing platform, or a reference to one such existing implementation is given. Numerical experiments were performed to compare the quality of variants of the methods.

Acknowledgements

During my entire master's studies at the University of Bergen, I had the opportunity to learn from Jan, in the courses taught by you and through the meetings we had for a reading course and this master's thesis, with you as supervisor. I greatly enjoyed the discussions we had about optimisation and the mathematics of it, the ways to present a topic, academia in general, with from time to time a conversation about how we perceive Norway, both having moved from abroad. I left our meetings each time with enthusiasm to further dig into a topic, and the regularity of them made me feel welcome in Norway; thank you for everything.

I furthermore want thank my sister, mother and father and my friends, also for their support in moving to Norway in the first place to study optimisation, and what allowed me to write this thesis. Ik heb superveel genoten van mijn tijd in Bergen, en ben heel blij hier naar toe te zijn gegaan, ondanks dat ik niet altijd de beste ben in contact houden. Dank jullie wel voor dit onvergetelijke avontuur!

In the period of working on this thesis, I also followed Norwegian language courses, causing some variation of things to work on, which only has helped in writing this thesis. Tusen takk for ei fin tid!

Pim Heeman
Friday 12 May 2023

Contents

List of Algorithms

List of Figures

List of Implementations

Chapter 1

Introduction

As part of mathematical optimisation, derivative-free optimisation is the study of optimisation where full access to an important tool used in regular optimisation is no longer granted: the access to the derivative of the objective function. In mathematical optimisation, a local minimum of function is sought, possibly under some constrains regarding the feasibility of points. The derivative on such points takes the value zero – except for points on the boundary of the set of feasible points – while we otherwise commonly are able to directly extract information from the derivative on the direction to follow to find a local optimum, making the derivative a powerful tool for finding optima.

However, for some problems, the full derivative of the objective function is not available, and traditional methods that require this full access can no longer be applied to those problems. An example of such a problem can be found in [6], where a description is given of the problem of choosing of 31 design variables that would minimise a measure of vibration of a helicopter rotor blade. A measure was given by a complex simulation rather than by a direct sets of mathematical expression of which the derivative could be taken, and because of the complexity and the running time, it is unworkable to directly approximate the full derivative by other means, as numerical or automatic differentiation. Nevertheless, for optimisation methods that only uses direct function evaluations, a surrogate model was created with a shorter running time, on the cost of having less accuracy, and the function values to work on being more noisy than the underlying model is. Taking the derivatives of this surrogate model would therefore lead to unsatisfactory results.

With this in mind, the goal of derivative-free optimisation methods are to find a local optimum based on function evaluations of the objective, where those evaluations can be both costly and noisy. A measure of the effectiveness of the method are therefore both the quality of the obtained solution, and the number of function evaluations needed for this.

We emphasise that we assume certain mathematical properties on the underlying objective function, as for example differentiability. Derivative-free optimisation is then the mathematical study in finding methods with proven convergence for problems as described above,

where we need underlying assumptions on the function for those theorems. It here differs with blackbox optimisation, where no such assumptions are made, and the so-called heuristics as simulated annealing are studied, for more ad-hoc solution methods. Three recent works providing a greater overview on derivative-free optimisation are [18], providing an overview of publications on derivative-free optimisation, [1], stating several derivative-free and blackbox optimisation algorithms with a short analysis, and [12], describing some derivative-free optimisation algorithms, and especially providing background in the derivative-free trust-region method.

In this work, we review three derivative-free optimisation methods: the Nelder–Mead method in Chapter 3, as one of the earliest methods of its kind, the conditional trust-region method in Chapter 4, where a well-known method known from the regular setting was adjusted to fit in the derivative-free setting, and the discrete gradient method in Chapter 5, as method that does not assume the objective function to be continuously differentiable for its convergence statements. To convey the ideas behind the methods best, we restrict ourself to the unconstrained setting in presentation of those methods, with a note that variants for the constrained case do exist in the literature. We start by recalling some concepts of differentiability in Chapter 2, and conclude this work with some numerical experiments in Chapter 6. Lastly, for a selection of the algorithms encountered, a MATLAB implementation has been made available in Appendix A.

Chapter 2

Differentiability and Optimality

In this chapter, our goal is to come up with a way of describing conditions for a solution of an unconstrained minimisation problem to be optimal based on intrinsic properties of the objective function. We will start by defining a generalisation of differentiability in more dimensions that will help us in defining optimality conditions. Then, we will consider non-differentiable functions for which we want to have defined nevertheless similar properties, to be used in defining optimality conditions too. Lastly, based on this, we state the actual optimality conditions. For solutions that are not optimal, we furthermore provide guidance in finding a solution providing a lower objective values, to be used by more sophisticated optimisation methods, with a direction that at least in the immediate neighbourhood provides a lower function value.

2.1 Total differentiability

Commonly known from the one-dimensional setting, the derivative of a function describes the effect of a fixed change in function argument on the resulting function value, by providing some value for each point in the domain of that function. For a real function $g\colon \mathbb{R} \to \mathbb{R}$, the derivative $g'\colon \mathbb{R} \to \mathbb{R}$ at point $x \in \mathbb{R}$ is defined as

$$g'(x) \coloneqq \lim_{t \to 0, t \in \mathbb{R}} \frac{g(x+t) - g(x)}{t}.$$

However, for more general functions $f\colon \mathbb{R}^n \to \mathbb{R}^m$, the domain is multi-dimensional, and we cannot capture all changes in this domain with a single scalar value, as t did in the previous equation. Thus, we need a generalisation of the derivative for multi-variate real functions f.

We can describe one notion of the derivative by encoding some direction explicitly in the definition of the derivative, effectively reducing the dimension in which a change can be made to a one-dimensional space. This gives rise to the following definition of the directional derivative:

Definition 2.1. Let $S \subseteq \mathbb{R}^n$ and $f\colon S \to \mathbb{R}^m$. The one-sided *directional derivative* of f at $x \in S$ in the direction $d \in \mathbb{R}^n$ is given by[1]

$$f'(x;d) \coloneqq \lim_{t\to 0^+} \frac{f(x+td) - f(x)}{t}. \tag{2.1}$$

One could then define differentiability of multi-variate real functions f as a property that holds if the directional derivative exists in all possible directions, just as a real function is differentiable if the derivative exist. However, as noticed in [23], several useful properties of differentiability in one dimension are then no longer implied for those multi-variate functions; differentiable functions on $\mathbb{R}$ are continuous, but functions for which the directional derivative in all directions exist are not necessarily continuous, since only straight directions are considered:

Example 2.1. Define

$$f(x,y) = \begin{cases} 1, & y = x^2, x \neq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Visually, in a contour plot of this two-dimensional function, the parabola $y = x^2$ would represent points on which f takes value 1 (except on $(0,0)$), with 0 on all other points. Let $d = (d_1, d_2) \in \mathbb{R}^2 \setminus (0,0)$ and let $t_0 > 0$ be such that $t_0 d_2 = (t_0 d_1)^2$. Then, $t_0 d_2 = t_0^2 d_1^2 \Leftrightarrow d_2 = t_0 d_1^2$ and for every $0 < t < t_0$, $td_2 = t \cdot t_0 d_1^2 \neq t^2 d_1^2 = (td_1)^2$. Thus, for every direction d, there exists a $t_0 > 0$ such that for all $0 < t < t_0$, $td_2 \neq (td_1)^2$, and $f(td) = 0$.
The directional derivative of f at $(0,0)$ exist thus in every direction $d \in \mathbb{R}^2$, with the following value:

$$f'(\mathbf{0};d) = \lim_{t\to 0^+} \frac{f(td) - f(\mathbf{0})}{t} = \lim_{t\to 0^+} \frac{0-0}{t} = 0.$$

This function f is clearly not continuous though, as continuity does consider all paths of reaching a point, including paths taking the form of a parabola:

$$\lim_{(x,x^2)\to(0,0)} f\left(x, x^2\right) = 1 \neq 0 = f(0,0).$$

Compositions of directionally differentiable functions are not necessarily directionally differentiable either, and consequently, we take the following definition as our definition of (total) differentiability for multi-variate functions instead, for which those two properties do hold:

Definition 2.2 ([23])**.** Let $S \subseteq \mathbb{R}^n$ and $f\colon S \to \mathbb{R}^m$. Then, f is *(totally) differentiable* at

[1] Some, e.g. [23], reserve the notion of a plain directional derivative to the two-sided one, with the 0^+ in the limit condition replaced by 0. We follow here the notation of [9] to be in line with the definition of a to-be-seen generalisation of the directional derivative.

$x \in S$ if there exists an $m \times n$ matrix G such that

$$\lim_{h \to 0} \frac{f(x+h) - f(x) - Gh}{\|h\|} = 0. \tag{2.2}$$

The (unique) matrix $G \in \mathbb{R}^{m \times n}$ is then called the *total derivative* – or simply *derivative* for short – of f at x, denoted by $Df(x)$ and can be viewed as a function $\mathbb{R}^n \to \mathbb{R}^{m \times n}$.

It follows from this definition that the derivative itself is a linear operator. We write $\mathcal{C}^k(S, S')$, or plainly $\mathcal{C}^k$ if the domain and codomain are clear from the context, as set containing all functions $f\colon S \to S'$ for which all lth derivatives exist and are continuous, with $l = 0, \ldots, k$.

With the earlier mentioned properties for differentiable one-dimensional functions holding with this other definition, total differentiability is indeed a stronger concept than directional differentiability, as the following theorem shows:

Theorem 2.1 ([23, Thm. 5.1])**.** *Let $S \subseteq \mathbb{R}^n$ and $f\colon S \to \mathbb{R}^m$. If f is differentiable at $x \in S$, then all directional derivatives of f at x exist and for all directions $d \in \mathbb{R}^n$,*

$$f'(x; d) = Df(x) \cdot d. \tag{2.3}$$

However, even if a function is continuous and its directional derivatives exist and are zero in all directions, the function is not necessarily differentiable, and thus, the previous theorem cannot be reversed, as we can conclude from the following example:

Example 2.2 (inspired by [42])**.** Define $g(t) = 1/\left(1 + t^2\right)$ and

$$f(x, y) = \begin{cases} x \cdot g\left(\left(y - x^2\right)/\left(x^2 + y^2\right)^2\right), & (x, y) \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

A plot of this function can be found in Figure 2.1.
We study the differentiability of f in the origin, $(0, 0)$, with $f(0, 0) = 0$, and start by considering points $\left(x, x^2\right)$ on a path that takes the place of h in the necessary condition of differentiability, (2.2). For such points, $f\left(x, x^2\right) = x$ holds, and we will compare the rate of reaching $f(0, 0) = 0$ along the path $\left(x, x^2\right)$ with the paths by following both axes x and y in a straight line.
In the case of following the axes, we are interested in the directional derivative of f at $(0, 0)$ in the directions $\mathbf{e}_1$ and $\mathbf{e}_2$, given respectively by

$$\begin{aligned} f'((0, 0); (1, 0)) &= \lim_{t \to 0} \frac{f(t, 0) - f(0, 0)}{t} = \lim_{t \to 0} \frac{1}{t} \cdot t \cdot g\left(-t^2/t^4\right) = \lim_{t \to 0} g\left(-1/t^2\right) \\ &= \lim_{t \to 0} \frac{1}{1 + t^4} = 0 \end{aligned}$$

and

$$f'((0,0);(0,1)) = \lim_{t\to 0} \frac{f(0,t) - f(0,0)}{t} = \lim_{t\to 0} \frac{1}{t} \cdot 0 \cdot g\left(t/t^4\right) = \lim_{t\to 0} \frac{1}{t} \cdot 0 \cdot g\left(1/t^3\right)$$
$$= \lim_{t\to 0} \frac{1}{t} \cdot 0 \cdot \frac{1}{1+t^6} = 0.$$

Thus, for f to be differentiable, it should hold that $Df \equiv 0$, as can be seen by plugging in both results of the directional derivatives in (2.3) to fulfil the linearity of the derivative. However, should f be differentiable, then (2.2) should hold too, and thus at $(0,0)$ for the path $\left(x, x^2\right)$, it should hold that

$$0 = \lim_{x\to 0} \frac{f\left(x,x^2\right) - 0 - 0}{\|(x,x^2)\|} = \lim_{x\to 0} \frac{x}{\sqrt{x^2+x^4}}$$

in particular. However, we have

$$\lim_{x\to 0^{\pm}} \frac{x}{\sqrt{x^2+x^4}} = \pm \lim_{x\to 0^{\pm}} \sqrt{x^2/\left(x^2\left(1+x^2\right)\right)} = \pm \lim_{x\to 0^{\pm}} \sqrt{1/\left(1+x^2\right)} = \pm 1,$$

and thus, f cannot be differentiable, since the two previous formulas are in contradiction.

As special notation, we recall the *gradient* of some function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ as the transpose of the derivative: $\nabla f(x) \coloneqq Df(x)^T \in \mathbb{R}^{n\times m}$. For twice-differentiable – for which the derivative is differentiable – real functions $f\colon \mathbb{R}^n \to \mathbb{R}$, the *Hessian* is then defined as the gradient of the derivative: $\nabla^2 f(x) \coloneqq \nabla Df(x) = D(Df)(x)^T \in \mathbb{R}^{n\times n}$.

2.2 Subdifferentiability

A natural question after having seen differentiable functions is if certain 'useful' properties can be generalised to non-differentiable functions too, to later allow reasoning about the optimality conditions as mentioned at the beginning of this chapter using properties common for both differentiable and non-differentiable functions. A broad class of such functions is the class of convex functions, that are directionally differentiable without necessarily being totally differentiable.

Definition 2.3. Let $f\colon \mathbb{R}^n \to \mathbb{R}$. Then, f is *convex* if for all $z_1 \neq z_2 \in \mathbb{R}^n$ and $\lambda \in (0,1)$,

$$f(\lambda z_1 + (1-\lambda) z_2) \leq \lambda f(z_1) + (1-\lambda) f(z_2).$$

If the above condition holds with a strict inequality, f is more specifically called *strictly convex*.

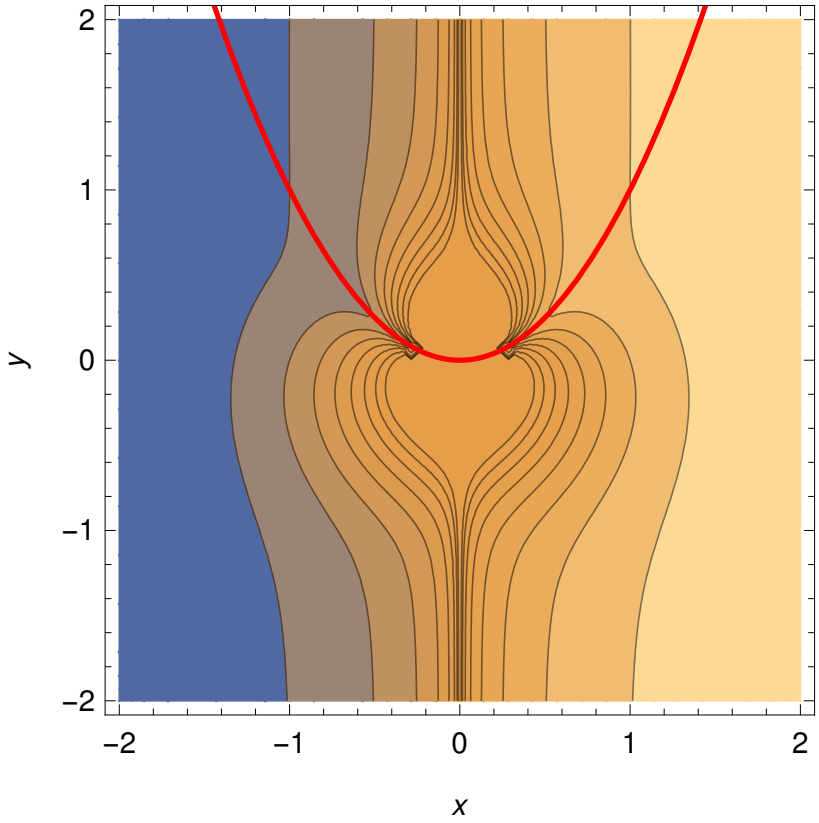


Figure 2.1: Plot of the function $f\colon \mathbb{R}^2 \to \mathbb{R}$ defined by (2.4) on the interval $[-2,2] \times [-2,2]$. On top of this function f are points $\left(x, x^2\right)$ marked, being points with $f\left(x, x^2\right) = x$.

An example of a convex function is a quadratic function, while a non-trivial cubic function is not.

For convex functions in general, we can conveniently derive a different expression for the directional derivative, compared to our previous expression (2.1) involving a limit:

Lemma 2.2 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function and $x \in \mathbb{R}^n$ a point in the domain of f. Then, all directional derivatives of f at x exist and for all directions $d \in \mathbb{R}^n$, it holds that*

$$f'(x;d) = \inf_{t>0} \frac{f(x+td) - f(x)}{t}. \tag{2.5}$$

Now, for differentiable convex functions, rewriting this new expression (2.5) using (2.3) gives

$$f'(x;d) \leq \frac{f(x+td) - f(x)}{t} \Leftrightarrow \qquad \forall t > 0, d \in \mathbb{R}^n \tag{2.6a}$$

$$\nabla f(x)^T \frac{1}{t}(y-x) \leq \frac{1}{t}(f(y) - f(x)) \Leftrightarrow \qquad \forall t > 0, y \in \mathbb{R}^n, d = \frac{1}{t}(y-x) \tag{2.6b}$$

$$f(x) + \nabla f(x)^T (y-x) \leq f(y). \qquad \forall y \in \mathbb{R}^n \tag{2.6c}$$

For non-differentiable functions f in x, the gradient $\nabla f(x)$ for which this inequality is fulfilled[2] does not exists, but instead, we can look for other vectors in the place of $\nabla f(x)$ for making (2.6c) valid:

Definition 2.4. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, the *subdifferential of a convex function* of f at $x \in \mathbb{R}^n$ is given by

$$\partial_{\mathrm{c}} f(x) = \left\{\xi \in \mathbb{R}^n \colon f(y) \geq f(x) + \xi^T (y - x), \forall y \in \mathbb{R}^n\right\}.$$

Elements $\xi \in \partial_{\mathrm{c}} f(x)$ are called *subgradients* of f at x.

With the subdifferential being a generalisation of the gradient in (2.6c), which in (2.6) was equivalent with (2.6a) involving the directional derivative, we can link the subdifferential and the directional derivative together too, such that values of the directional derivative in all direction is enough for knowing the subdifferential, and in reverse, that knowing the subdifferential is enough for knowing the directional derivative in any direction:

Theorem 2.3 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, for all $x \in \mathbb{R}^n$,*

(i) $f'(x; d) = \max\left\{\xi^T d \colon \xi \in \partial_{\mathrm{c}} f(x)\right\}$ *for any* $d \in \mathbb{R}^n$,
(ii) $\partial_{\mathrm{c}} f(x) = \left\{\xi \in \mathbb{R}^n \colon \forall d \in \mathbb{R}^n, f'(x; d) \geq \xi^T d\right\}$.

A next step in generalisation the notion of differentiability is to come up with a expression taking over the function of the directional derivative in case it doesn't exist, for a more generalised version of Theorem 2.3.(ii). Commonly used in proofs of the previous theorem though (see e.g. [3, Thm. 2.28] and [7, Thm. 3.1.8]) is that the subadditivity of $\mathbb{R}^n \to \mathbb{R}, d \mapsto f'(x; d)$ for all $x \in \mathbb{R}^n$: for all $d_1, d_2 \in \mathbb{R}^n$, $f'(x; d_1 + d_2) \leq f'(x; d_1) + f'(x; d_2)$ holds, and our generalisation might need this too.

The first generalisation of the directional derivative to encounter is the Dini directional derivative. Inspired by the definition of the definition of the (total) Dini derivative in [26], it has later been used in a directional way (e.g. in [7]), leading to the following definition:

Definition 2.5. Let $f\colon \mathbb{R}^n \to \mathbb{R}$. The *upper Dini directional derivative* of f at $x \in \mathbb{R}^n$ in the direction $d \in \mathbb{R}^n$ is given by

$$f^+(x; d) \coloneqq \limsup_{t \to 0^+} \frac{f(x + td) - f(x)}{t}$$

and the *lower Dini directional derivative* by

$$f^-(x; d) \coloneqq \liminf_{t \to 0^+} \frac{f(x + td) - f(x)}{t}.$$

[2] In fact, equality occurs in (2.6c) for differentiable functions in general in the statement below; in our quest for finding a generalisation of the gradient, we do not focus on this in our motivation.

Clearly, the upper and lower Dini directional derivative in the direction d at some point x always exist and if the (regular) directional derivative of the function exists, then its value equals that of both the upper and lower Dini directional derivative.

With this generalisation of the directional derivative, the following provides a natural-following generalisation of the subdifferential of a convex function, which as mentioned is inspired by Theorem 2.3.(ii):

Definition 2.6. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. Then, the *upper Dini subdifferential* of f at x is given by

$$\partial_+ f(x) = \left\{\xi \in \mathbb{R}^n \colon \forall d \in \mathbb{R}^n, f^+(x;d) \geq \xi^T d\right\}$$

and the *lower Dini subdifferential* by

$$\partial_- f(x) = \left\{\xi \in \mathbb{R}^n \colon \forall d \in \mathbb{R}^n, f^-(x;d) \geq \xi^T d\right\}.$$

However, both mappings $d \mapsto f^+(x;d)$ and $d \mapsto f^-(x;d)$ are not subadditive, and an equivalent statement as given by Theorem 2.3.(i) by simply replacing the directional derivative and its corresponding subdifferential with the Dini variant – which would state that $f^{\pm}(x;d)$ equals $\max\left\{\xi^T d \colon \xi \in \partial_{\pm} f(x)\right\}$ for any $d \in \mathbb{R}^n$ – does not hold in general, as shown in the following example:

Example 2.3 (choice of function f inspired by [7])**.** Define $f(x) = -|x|$.
This function is directionally differentiable at 0 in any direction $d \in \mathbb{R}$ with the value

$$f'(0;d) = \lim_{t \to 0^+} \frac{f(0+td) - f(0)}{t} = \lim_{t \to 0^+} \frac{-|td|}{t} = -|d|,$$

and thus, the upper and lower Dini directional derivative are given by the same value: $f^+(0;d) = f^-(0;d) = -|d|$. However, this function $d \mapsto -|d|$ is not subadditive: $-|1+-1| = 0 > -2-1+-1-|1|+-|-1|$.
We will now check if nevertheless the equivalent statement of Theorem 2.3.(i) for the Dini directional derivatives can be fulfilled. The upper and lower Dini subdifferential of f at 0 for this are given by

$$\begin{aligned}\partial_+ f(0) = \partial_- f(0) &= \left\{\xi \in \mathbb{R} \colon \forall d \in \mathbb{R}, f^-(0;d) \geq \xi d\right\} \\ &= \{\xi \in \mathbb{R} \colon \forall d \in \mathbb{R}, -|d| \geq \xi d\} \\ &= \{\xi \in \mathbb{R} \colon \forall d \in \mathbb{R}, |d| \leq -\xi d\}.\end{aligned}$$

For $d = 0$, the condition holds trivially for all values $\xi \in \mathbb{R}$. For $d > 0$, $|d| = d$ and the condition becomes $d \leq -\xi d \Leftrightarrow 1 \leq -\xi \Leftrightarrow \xi \leq -1$. However, for $d < 0$, $|d| = -d$ and the condition becomes $-d \leq -\xi d \Leftrightarrow d \geq \xi d \Leftrightarrow \xi \geq 1$, and thus, both Dini

subdifferentials of f at 0 are empty: $\partial_+ f(0) = \partial_- f(0) = \varnothing$. An equivalent version of Theorem 2.3 does thus not hold.

As alternative, we will consider the so-called Clarke directional derivative, which is defined for the following type of functions:

Definition 2.7. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. Then, f is called *locally Lipschitz continuous* at x if there exist $K > 0$ and $\epsilon > 0$ such that

$$|f(y) - f(z)| \leq K \|y - z\| \text{ for any } y, z \in B(x; \epsilon). \tag{2.7}$$

Lemma 2.4 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable at $x \in \mathbb{R}^n$. Then, f is locally Lipschitz continuous at x.*

A function is not locally Lipschitz at some point if it is unbounded steep: $x \mapsto \sqrt{|x|}$ is an example of a function that is not locally Lipschitz at 0, having asymptotes to ∞ and $-\infty$ there.

With this, we can recall the Clarke directional derivative as the following:

Definition 2.8 ([9])**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$. The *Clarke directional derivative* of f at $x \in \mathbb{R}^n$ in the direction $d \in \mathbb{R}^n$ is given by

$$f^\circ(x; d) \coloneqq \limsup_{\substack{y \to x \\ t \to 0^+}} \frac{f(y + td) - f(y)}{t}. \tag{2.8}$$

Lemma 2.5 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. Then, all Clarke directional derivatives of f at x exist for all directions $d \in \mathbb{R}^n$.*

With this, we will see another generalisation of the subdifferential, defined in the same spirit as the upper and lower Dini subdifferential was:

Definition 2.9. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. Then, the *Clarke subdifferential* of f at x is given by

$$\partial_\circ f(x) = \left\{\xi \in \mathbb{R}^n \colon \forall d \in \mathbb{R}^n, f^\circ(x; d) \geq \xi^T d\right\}.$$

Elements $\xi \in \partial_\circ f(x)$ are called *subgradients* of f at x.

As desired, this generalisation of the directional derivative is actually subadditive, and the equivalence of Theorem 2.3.(i) can using this be proved too for the Clarke directional derivative:

Lemma 2.6 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. Then, the Clarke subdifferential $\partial_\circ f(x)$ of f at x is non-empty, convex and compact set.*

Theorem 2.7 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. Then,*

$$f^\circ(x;d) = \max\left\{\xi^T d\colon \xi \in \partial_\circ f(x)\right\} \text{ for any } d \in \mathbb{R}^n. \tag{2.9}$$

However, getting the subadditivity and this equivalence being proved came at the cost of the Clarke directional derivative not necessarily coinciding with the regular directional derivative, if this exists, as shown in the following example, using the same function as in the previous example on the Dini directional derivative:

Example 2.4. Define $f(x) = -|x|$.
As seen in Example 2.3, this function is directionally derivative at 0 with the directional derivative given for any $d \in \mathbb{R}$ by

$$f'(0;d) = -|d|.$$

With help of the triangle inequality, we can almost immediately see that f is everywhere locally Lipschitz continuous, for $K = 1$ in (2.7):

$$|-|y| - (-|z|)| = ||y| - |z|| \leq |y - z|.$$

We will now calculate Clarke directional derivative of $x \mapsto -|x|$ in direction $d \in \mathbb{R}$, by relating it to the Clarke directional derivative of $-f\colon x \mapsto |x|$. Using the definition of the directional derivative, we write

$$(-f)^\circ(0;d) = \limsup_{\substack{y\to 0\\ t\to 0^+}} \frac{f(y+td) - f(y)}{t} = \limsup_{\substack{y\to 0\\ t\to 0^+}} \frac{|y+td| - |y|}{t}.$$

Then, with a lower bound, found with some estimation of the lim sup, of

$$(-f)^\circ(0;d) \geq \lim_{s\to 0^+} \frac{\left|s^2 + sd\right| - \left|s^2\right|}{s} = \lim_{s\to 0^+} \frac{|s|(|s+d| - |s|)}{s} = \lim_{s\to 0^+} |s+d| - |s| = |d|,$$

and an upper bound, found with the triangle inequality, of

$$(-f)^\circ(0;d) \leq \limsup_{\substack{y\to 0\\ t\to 0^+}} \frac{|y| + |td| - |y|}{t} = \limsup_{\substack{y\to 0\\ t\to 0^+}} \frac{|td|}{t} = \limsup_{\substack{y\to 0\\ t\to 0^+}} \frac{|td|}{t} = |d|,$$

we conclude $(-f)^\circ(0;d) = |d|$.
For the Clarke directional derivative, we use the general relation (with $z := y + td$)

$$(-f)^\circ(x;d) = \limsup_{\substack{y\to x\\ t\to 0^+}} \frac{-f(y+td) - (-f(y))}{t} = \limsup_{\substack{z\to x\\ t\to 0^+}} \frac{-f(z) - (-f(z-td))}{t}$$

$$= \limsup_{\substack{z \to x \\ t \to 0^+}} \frac{f(z - td) + f(z)}{t} = f^\circ(x; -d)$$

to conclude $f^\circ(0; d) = |-d| = |d|$ as well.
Thus, the Clarke directional derivative and the regular directional derivative of $x \mapsto |x|$ are different for all non-trivial directions.

For convex functions though, the Clarke directional derivative coincides with the regular directional derivative, and the Clarke subdifferential agrees thus with the regular subdifferential for convex functions:

Theorem 2.8 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, for all $x \in \mathbb{R}^n$*

(i) *$f^\circ(x; d) = f'(x; d)$ for any $d \in \mathbb{R}^n$,*
(ii) *$\partial_\circ f(x) = \partial_c f(x)$.*

With the concept of the Clarke subdifferential derivative providing a true generalisation of the subdifferential for convex functions with the equivalence of Theorem 2.3 – providing a relation between the subdifferential and directional derivative, which we will soon use in defining the optimality conditions – holding as well, we will in what follows drop the Clarke specification when referring to the Clarke subdifferential, with the symbol ∂ instead of $\partial_\circ$.

We conclude this section on subdifferentiability by relating the subdifferential with the gradient, by means of the following three theorems:

Theorem 2.9 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous and differentiable function at $x \in \mathbb{R}^n$. Then,*

$$\nabla f(x) \in \partial f(x).$$

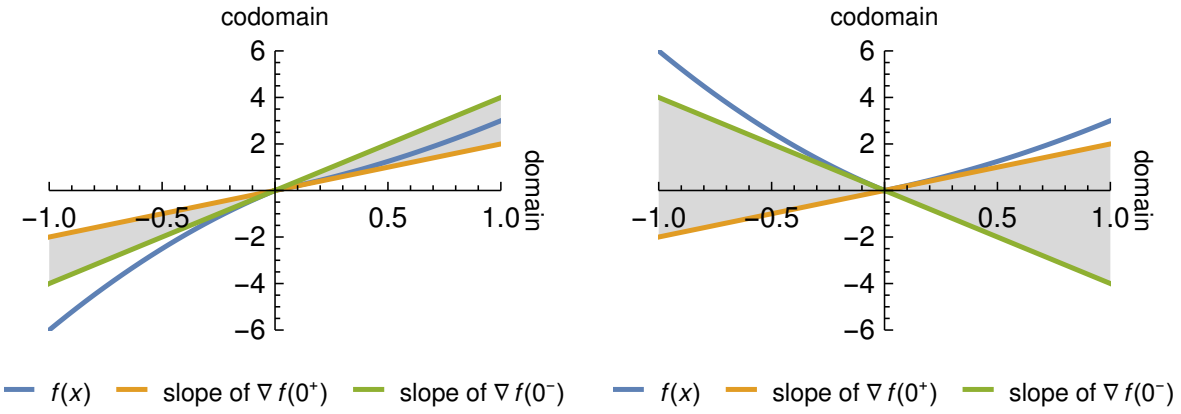
Theorem 2.10 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function at $x \in \mathbb{R}^n$. Then,*

$$\{\nabla f(x)\} = \partial f(x).$$

Theorem 2.11 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$ and define $\Omega_f \subset \mathbb{R}^n$ as the set of points at which f is not differentiable. Then,*

$$\partial f(x) = \operatorname{conv}\{\xi \in \mathbb{R}^n\colon \textit{there exists a sequence } \{x_k\} \subset \mathbb{R}^n \setminus \Omega_f \textit{ such that } x_k \to x \textit{ and } \nabla f(x_k) \to \xi\}. \tag{2.10}$$

With Lipschitz continuous functions being almost everywhere differentiable, and the set of non-differentiable points Ω_f being a null set, the above theorems provide a different way to compute and visualise the subdifferential of some function, by thinking of the subdifferential as the set of vectors that provide a smooth transformation between all the gradient vectors when approaching a non-differentiable point from different directions. A visualisation in the



(a) $f(x) = \begin{cases} (x+1)^2 - 1, & x > 0 \\ -2\left((x+1)^2 - 1\right), & \text{otherwise} \end{cases}$

(b) $f(x) = \begin{cases} (x+1)^2 - 1, & x > 0 \\ 2\left((x+1)^2 - 1\right), & \text{otherwise} \end{cases}$

Figure 2.2: The subdifferential of two different continuous functions $f\colon \mathbb{R} \to \mathbb{R}$ visualised at 0. With f being only non-differentiable at 0, the gradient when approaching 0 from the left and from the right is shown, visualised by an affine line at $f(0)$ whose slope corresponds to the value of the gradient. The subdifferential is visualised similarly, with lines with their slope in the subdifferential being marked light gray.

two-dimensional case is shown in Figure 2.2, with the calculation of the subdifferential for our example function $x \mapsto -|x|$ below:

Example 2.5. Define $f(x) = -|x|$, an everywhere locally Lipschitz continuous function.
This function is only not differentiable at 0, while being continuous differentiable on $(-\infty, 0)$ and $(0, \infty)$, with the derivative being described by the constant functions 1 and -1 on those intervals. Thus, the subdifferential of f at 0 is given by $\partial f(0) = \operatorname{conv}\{1, -1\} = [-1, 1]$.

2.3 Optimality conditions

Having seen different ways to measure the change in function value at some point, we can now state the optimality conditions for unconstrained mathematical optimisation problems – or the conditions on points in $\mathbb{R}^n$ to be considered a solution of a minimisation problem.

One type of such points are the local minima, for which the function value is smaller than the function values at all surrounding points, and the global minima, in which we take the whole domain into account for this comparison:

Definition 2.10. Let $f\colon \mathbb{R}^n \to \mathbb{R}$. A point $x^* \in \mathbb{R}^n$ is then called a *local minimum* of f if

there exists a $\delta > 0$ such that

$$f(x^*) \leq f(x) \text{ for any } x \in B(x^*; \delta).$$

Similarly, if this inequality holds for any $x \in \mathbb{R}^n$, we call x^* more specifically a global minimum.

For recognising such a point, we can use the following theorem, stating necessary conditions for a point to be a local minimum for locally Lipschitz continuous functions, including continuously differentiable functions. Commonly-known proofs (e.g. in [3]) use (2.9), the equality that as generalisation of the regular directional derivative holds for the Clarke directional derivative, but not for the upper and lower Dini directional derivative:

Theorem 2.12 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x^* \in \mathbb{R}^n$. If the point x^* is a local minimum of f, then*

$$\mathbf{0} \in \partial f(x^*) \text{ and } f^\circ(x^*; d) \geq 0 \text{ for any } d \in \mathbb{R}^n. \tag{2.11}$$

For a continuously differentiable function, the first condition in (2.11) could be stated as $\mathbf{0} = \nabla f(x^*)$, using Theorem 2.10. We treat special attention to points fulfilling this first condition, being potential local minimisers, while being able to exclude points not fulfilling this criterion in a less stringent way than checking both conditions would give:

Definition 2.11. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. If $\mathbf{0} \in \partial f(x)$, we call x a *stationary point* or *first-order critical point.*

Coming back to the figure at the end of the previous section, we can recognise a local minimum in Figure 2.2b at $x = 0$, with a vector with zero slope being marked light gray, and thus $\mathbf{0} \in \partial f(0)$. In this case, f only has a local minimum being a global minimum, and seeking for a stationary point leads us to this point. In general, as motivation for studying stationary points, this property holds in general for convex functions:

Theorem 2.13 ([3])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function at $x^* \in \mathbb{R}^n$.*

Then x^* is a global minimum of f if and only if $\mathbf{0} \in \partial f(x)$ or $f^\circ(x^*; d) \geq 0$ for any $d \in \mathbb{R}^n$.

We now restrict ourself to twice continuously differentiable functions, for a different pair of sufficient and necessary conditions on a local minimum. As necessary condition, we get the following theorem, similar to Theorem 2.12:

Theorem 2.14 ([11])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a twice continuously differentiable function. If $x^* \in \mathbb{R}^n$ is a local minimum of f, then*

$$\mathbf{0} = \nabla f(x^*) \text{ and } d^T \nabla^2 f(x^*)\, d \geq 0 \text{ for any } d \in \mathbb{R}^n.$$

The reversed statement, but this time under the same assumptions, is then the following:

Theorem 2.15 ([11]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a twice continuously differentiable function and $x^* \in \mathbb{R}^n$. If*

$$\mathbf{0} = \nabla f(x^*) \text{ and } d^T \nabla^2 f(x^*)\, d > 0 \text{ for any } d \in \mathbb{R}^n \setminus \{0\},$$

then x^* is a (strictly isolated) local minimum of f.

Based on this, we can define a different type of critical point:

Definition 2.12. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. If $\nabla f(x) = \mathbf{0}$ and $d^T \nabla^2 f(x^*)\, d > 0$ for any $d \in \mathbb{R}^n$, we call x a *second-order critical point.*

2.4 Descent direction

In the previous section, we have seen conditions for a point being a local minimum, as goal for optimisation points to find. In the course of optimisation methods, an important concept to find such points is the concept of a descent direction, which provides us a direction from some (iteration) point to (at least locally) a point with a lower function value:

Definition 2.13 ([3]). Let $f\colon \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. The direction $d \in \mathbb{R}^n$ is a *descent direction* for f at x if there exists an $t_0 > 0$ such that for all $t \in (0, t_0]$,

$$f(x + td) < f(x).$$

For (locally) Lipschitz continuous functions, the following theorem is about the existence of a descent direction:

Theorem 2.16 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $x \in \mathbb{R}^n$. The direction $d \in \mathbb{R}^n$ is then a descent direction if*

$$\xi^T d < 0 \text{ for all } \xi \in \partial f(x^*) \text{ or } f^\circ(x^*; d) \geq 0.$$

Along with the first-order necessary conditions for a local minimum from Theorem 2.12, those two theorems tell us that for some Lipschitz continuous function, either the zero vector is in the subdifferential at some point, or some descent direction exists. An explicit direction can be obtained by the following:

Lemma 2.17 ([3]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at x. Then either $\mathbf{0} \in \partial f(x)$ or, for $\xi^* = \operatorname{argmin}_{\xi in \partial f(x)} \|\xi\|$, $-\xi^*/\|\xi^*\|$ is a descent direction.*

For continuously differentiable functions, we recall the concept of the *steepest descent* among descent direction as the descent direction such that following that direction in the

linearised version of the function provides the greatest decrease, e.g. a direction d^* such that $d^* := \text{argmin}_{\|d\|=\|\nabla f(x)\|} f(x) + \nabla f(x)^T d$, where we decided the norm of d^* to be without loss of generality $\|\nabla f(x)\|$, out of all strictly positive values it could have been, as we so far are only interested in the direction and not it's magnitude. To find d^*, using [32], because of Cauchy-Schwarz, we can write

$$\nabla f(x)^T d \leq \|\nabla f(x)\| \|d\| \Leftrightarrow \nabla f(x)^T \frac{d}{\|d\|} \leq \|\nabla f(x)\|,$$

where the upper bound in this inequality is reached for any $d = -\lambda \nabla f(x)$ for $\lambda \in \mathbb{R}$. Setting $\lambda = 1$ then gives us the steepest descent direction: $d^* = -\nabla f(x)$.

Chapter 3

Nelder–Mead Method

In 1965, [24] was published, describing what later became known as the Nelder–Mead method for function minimisation. The method operates by having a simplex formed of $n+1$ linearly independent points in the n-dimensional parameter space of the function, and moving the simplex through the space towards (hopefully) the minimiser, adjusting itself to the landscape. No theoretical properties of the method have been given in the original paper describing the method, and examples for which the method fails to converge to a minimiser in the theoretical sense have been discovered later with only scare convergence results proved; despite this, the usage of the method is widespread, for example in the MATLAB numeric computing environment.

In this chapter, we review the original Nelder–Mead method. We then look at one of the described modified versions that are in the spirit of the original method, but for which convergence to a minimiser has been proved for broad classes of functions, in contrast to the original method.

3.1 Simplices

The concept of a *simplex* – a Latin adjective for 'simple' – was first described in [31]. A simplex $S\,(d+1)$ represented by $d+1$ suitably positioned point was described as being the simplest way to span a d-dimensional linear space, with the simplex itself consisting of all points that were inside all the facets spanned by the points. Specifically, an 1-dimensional simplex is a line segment, a 2-dimensional simplex a triangle, a 3-dimensional simplex a tetrahedron, et cetera.

In an n-dimensional Euclidean space, a simplex is described as the convex hull of $n+1$ affinely independent points. Given $n+1$ points $y^0, y^1, \ldots, y^n$, those points are called affinely independent if the vectors to reach n of those from the remaining point, the base point, are linearly independent, or without loss of generality, those points are affinely independent if the vectors $y^1 - y^0, y^2 - y^0, \ldots, y^n - y^0$ are linearly independent. Thus,

$\left\{y^1 - y^0, y^2 - y^0, \ldots, y^n - y^0\right\}$ forms a basis of $\mathbb{R}^n$ for $y^0, y^1, \ldots, y^n$ affinely independent points.

As measure of a property of a simplex, the *volume* of a simplex Y represented by the $n+1$ points $y^0, y^1, \ldots, y^n$ is, for $L = \begin{bmatrix} y^1 - y^0 & y^2 - y^0 & \cdots & y^n - y^0 \end{bmatrix} \in \mathbb{R}^{n \times n}$ known to be

$$\operatorname{vol}(Y) = \frac{|\det(L)|}{n!}$$

as found in [34], with the *diameter* of the simplex Y being defined as

$$\operatorname{diam}(Y) \coloneqq \max_{y, y' \in Y} \left\| y - y' \right\|.$$

During iterative algorithms where a simplex is being modified, as happens in the soon-described Nelder–Mead method, one would prefer a safeguard from the simplex becoming degenerate – the points being no longer affinely independent. For this to happen, the volume has to become zero and thus, a natural measure to avoid degeneracy could be to bound a number proportional to the volume away from zero. To let this be a scaling-invariant measure, the concept of the *normalised volume* was introduced in [38], with the following definition:

$$\operatorname{von}(Y) \coloneqq \operatorname{vol}\left(\frac{1}{\operatorname{diam}(Y)} Y\right) = \frac{1}{\operatorname{diam}(Y)^n} \operatorname{vol}(Y).$$

As an example of the normalised volume being scaling invariant, the 2-dimensional simplex represented by the vertices $\{(1,1), (1+t,1), (1,1+t)\}$ has the same normalised volume for all positive values t, corresponding to the internal angles staying the same too, whereas the volume itself, given by $t^2/2$, converges to zero for $t \to 0$.

3.2 Original variant of the Nelder–Mead method

The Nelder–Mead method was inspired by the previous work [33], published in 1962, in which optimisation of dynamic processes was described using *regular simplices* as Evolutionary Operation. Roughly speaking, a regular simplex – a simplex solely composed of lower-dimensional regular simplices with a regular triangle, the smallest-dimensional simplex, being an equilateral triangle – was conceptualised in the input domain of the process, with each vertex corresponding to some parameters for which an observation of the performance of the dynamic process could be made. Then, the vertex in this simplex corresponding to the worst observation was reflected over the centroid of the other vertices in the input domain to change a single vertex of the regular simplex, if at least all observations were recent enough; otherwise, a not-recent-enough observation was updated. This way, the sequence of simplices will move to an optimum, also as it moves in the dynamic process, and circle around the optimum with a distance dictated by the step size once arrived.

When considering the static problem of minimising a mathematical function, by allowing the size of the faces of the simplex from the previously described method to change, the optimum could get approached indefinitely close, making such a modified method a potential derivative-free method for function minimisation: the Nelder–Mead method was created and published in [24], only three years after publication of the work that inspired this:

Algorithm 3.1 (Nelder–Mead Method ([16]))**.**

Input *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be the function to be minimised and Y^0 the vertices of a non-degenerate starting simplex. Furthermore, we require the following constants as parameters of the method:*

- $0 < \rho$*: the parameter defining the ratio of length for reflection;*
- $1 < \chi$*: the parameter defining the ratio of length for lengthening the reflection for expansion;*
- $0 < \gamma < 1$*: the parameter defining the ratio of length for shortening the reflection for contraction;*
- $0 < \sigma < 1$*: the parameter defining the ration of length for shrinking.*

Output *The point corresponding to the lowest function value of f in $\mathbb{R}$ found so far at iteration k for $k \to \infty$: y_k^0.*

Initialisation *Set $k = 0$.*

Step 1 (restructuring) *Order the vertices of Y^k based on their function value such that it holds that*

$$f\left(y_k^i\right) \leq f\left(y_k^j\right), \qquad \forall i \leq j \tag{3.1}$$

in accordance with tie-breaking rules (see directly after this algorithm).
Compute the centroid of all-but-the-worst point $y_k^1, \ldots, y_k^{n-1}$: $\bar{x}_k \coloneqq \sum_{i=0}^{n-1} y_k^i / n$.

Step 2 (reflecting) *Compute the point x_k^{r} resulted from a reflection of the worst point y_k^n over the computed centroid $\bar{x}_k$:*

$$x_k^{\mathrm{r}} \coloneqq \bar{x}_k + \rho\left(\bar{x}_k - y_k^n\right)$$

and evaluate it on f as $f\left(x_k^{\mathrm{r}}\right)$.
If the function value of this point suffices $f\left(y_k^0\right) \leq f\left(x_k^{\mathrm{r}}\right) < f\left(y_k^{n-1}\right)$, e.g. is comparable with the function values of the remaining vertices that are not the best or worst, y_k^n is replaced with x_k^{r} in Y^k resulting in Y^{k+1}, k is set to $k+1$ and a new iteration is started by continuing in Step 1.

Step 3 (expanding) *If $f\left(x_k^{\mathrm{r}}\right) < f\left(y_k^0\right)$, the reflected point yields a better function value*

than the previously best point. We check if a continuation in this direction yields an even better function value by calculating

$$x_k^{\mathrm{e}} \coloneqq \bar{x}_k + \chi\rho\left(\bar{x}_k - y_k^n\right)$$

and evaluating it on f as $f\left(x_k^{\mathrm{e}}\right)$.
If $f\left(x_k^{\mathrm{e}}\right) < f\left(x_k^{\mathrm{r}}\right)$, y_k^n is replaced with x_k^{e} in Y^k resulting in Y^{k+1}. Otherwise, y_k^n is replaced with x_k^{r} instead. In any case, k is set to $k+1$ and a new iteration is started by continuing in Step 1.

Step 4 (contracting) *Now, $f\left(y_k^{n-1}\right) \leq f\left(x_k^{\mathrm{r}}\right)$ and a so-called contraction step is taken by evaluating a point between y_k^n and $\bar{x}_k$ or between $\bar{x}_k$ and x_k^{r}, depending on which side of the centroid looks the most promising based on the function value of the reflection compared to that of the current worst point:*

Step 4a (outside contraction) *If $f\left(x_k^{\mathrm{r}}\right) < f\left(y_k^n\right)$, calculate*

$$x_k^{\mathrm{c}} \coloneqq \bar{x}_k + \gamma\rho\left(\bar{x}_k - y_k^n\right) = \bar{x}_k + \gamma\left(x_k^{\mathrm{r}} - \bar{x}_k\right) = (1-\gamma)\,\bar{x}_k + \gamma x_k^{\mathrm{r}}.$$

If $f\left(x_k^{\mathrm{c}}\right) \leq f\left(x_k^{\mathrm{r}}\right)$, y_k^n is replaced with x_k^{c} in Y^k resulting in Y^{k+1}, k is set to $k+1$ and a new iteration is started by continuing in Step 1.

Step 4b (inside contraction) *Otherwise, if $f\left(y_k^n\right) \leq f\left(x_k^{\mathrm{r}}\right)$, calculate*

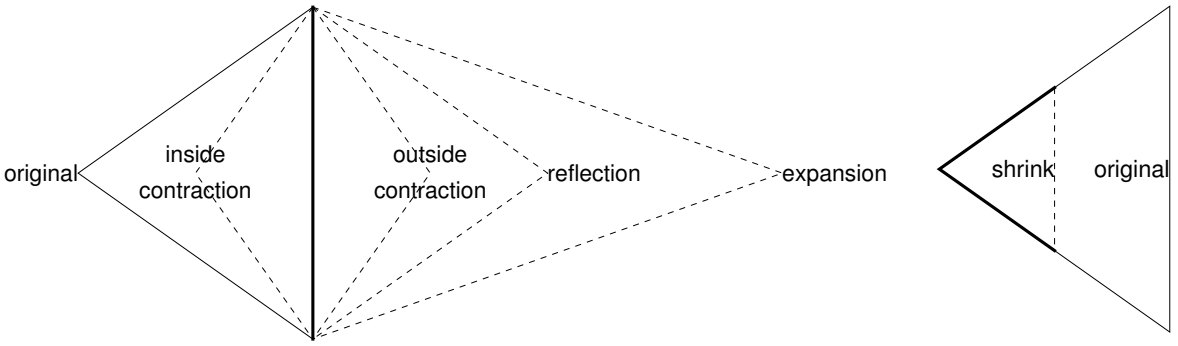
$$x_k^{\mathrm{cc}} \coloneqq \bar{x}_k - \gamma\left(\bar{x}_k - y_k^n\right) = (1-\gamma)\,\bar{x}_k + \gamma y_k^n.$$

If $f\left(x_k^{\mathrm{cc}}\right) < f\left(y_k^n\right)$, y_k^n is replaced with x_k^{cc} in Y^k resulting in Y^{k+1}, k is set to $k+1$ and a new iteration is started by continuing in Step 1.

Step 5 (shrinking) *If no points are replaced so far, a shrink step is taken by replacing every point y_k^i in Y^k with $y_{k+1}^i = y_k^0 + \sigma\left(y_k^i - y_k^0\right)$ for $i = 0, \ldots, n$ resulting in Y^{k+1}, k is set to $k+1$ and a new iteration is started by continuing in Step 1.*

To ensure a well-defined unique execution of the method above given the input data, a rule has been introduced by [16] according to which is decided how to order vertices when multiple orderings fulfil equation (3.1): the so-called tie-breaking rules. For a non-shrinking operation, the incoming vertex is assigned the highest possible index that fulfils the constraints. For a shrinking operation, if the function value of the only vertex kept, y_k^0, is the minimum of what would be the new simplex Y^{k+1}, then this vertex is kept at index 0: $y_{k+1}^0 = y_k^0$.

In the description above, the Nelder–Mead method was stated by following the description in [16] from 1998 instead of the original description found in [24], as the description in the latter is less suitable for mathematical analysis due to not being completely clear on how



(a) The four different operations that can be applied on the worst vertex: an inside contraction, outside contraction, reflection and expansion. The worst vertex is here the one originally on the left.

(b) The shrink operation, with on the left the best vertex.

Figure 3.1: All operations the Nelder–Mead method can apply in an iteration, shown for a 2-dimensional simplex represented by 3 vertices.

to resolve ambiguity. A difference in the description of the Nelder–Mead method is that for the well-defined one, the better of the reflection and expansion point is selected, while in the original form, the expansion point is chosen whenever it provides an improvement over the best vertex, regardless of the relative value of the expansion point over the reflection point.

Studying the operations above in Algorithm 3.1, it can be seen that, except for the time a shrink step is taken, at each iteration k, the worst vertex y_k^n is removed from the simplex Y^k and replaced in the otherwise identical simplex Y^{k+1} to be used in the next iteration $k+1$ by

$$\hat{y}_k = \bar{x}_k + \tau\left(\bar{x}_k - y_k^n\right) \tag{3.2}$$

for different values of τ, namely:

1. $\tau = \rho \in (0, \infty)$ for a reflection; accepted if the resulting function is moderate compared to the others;

2. $\tau = \chi\rho \in (\rho, \infty)$ for an expansion; in case the reflection step yielded a value better than already present in the simplex, the best of the reflection and expansion point is accepted;

3. $\tau = \gamma\rho \in (0, \rho)$ for an outside contraction, if the reflection point only improved the worst value; the outside contractor is accepted if better than the reflector;

4. $\tau = -\gamma \in (-1, 0)$ for an inside contraction, if the reflection point did not provide any improvement; the inside contractor is accepted if better than the reflector.

If none of those steps above are taken, a shrink step, with parameter σ, is taken, as last resort. All those options are visualised in Figure 3.1.

As standard values for those parameters are $\rho = 1$, $\chi = 2$, $\gamma = \frac{1}{2}$ and $\sigma = \frac{1}{2}$ commonly chosen, providing perfect symmetry for a reflection step and making the expansion step twice as large with the contraction and the shrink step half as large. Leading to such a natural image, those values are sometimes assumed to be taken during convergence analysis for the sake of simplicity and their widespread usage.

3.2.1 Counterexamples of convergence to a minimiser

After being published, the Nelder–Mead method has been frequently used, despite little being known about the convergence of the method for different types of functions. Through the years though, several functions were found for which the Nelder–Mead does not converge to a minimiser, making it possible to narrow in any case the class of functions for which the Nelder–Mead method possibly always would converge to a minimiser.

In 1985, an example of a non-convex differentiable 2-dimensional function was published in [41] in sketch form by providing a contour plot for which the Nelder–Mead method was claimed to converge to a non-minimiser by taking only shrink steps. Subsequently, in [20], it was claimed that the point to which convergence took place is a stationary point, though.

In that same publication [20] from 1998, more examples of problems were given for which convergence to a point that is not a minimiser occurs by applying the original Nelder–Mead method. An important difference between the original method that inspired the Nelder–Mead method, published in [33], and the Nelder–Mead method itself came up: in the first, the simplices cannot become degenerate, while they can in the latter. In the case of the simplex becoming degenerate, the faces do no longer span the parameter space $\mathbb{R}^n$, which could cause convergence to a non-minimiser; and indeed, with the function $(x, y) \mapsto x^2 - y\,(y - 2)$ and initial simplex with the vertices $(1, 0)$, $(0, -3)$ and $(0, 3)$, as mentioned in [20], this is the case; see Figure 3.2 for a visualisation of this problem.

The main class of functions studied in [20] though were 2-dimensional functions for which so-called ‘repeated focused inside contraction’ (RFIC) steps were taken: starting from the initial simplex, only inside contraction steps are taken in that case towards one of the points of the initial simplex, which always stays part of the simplex, but might not necessarily be a minimiser of the original function – although it has to be stationary point. Examples of a strictly convex differentiable 2-dimensional function exhibiting this behaviour were given, with a proof showing that such behaviour cannot occur for functions smoother than three-times differentiable functions.

3.2.2 Convergence properties

While we now have seen that convergence is not guaranteed in general for a broad class of functions, we would pose the question for which class of functions convergence actually does happen.

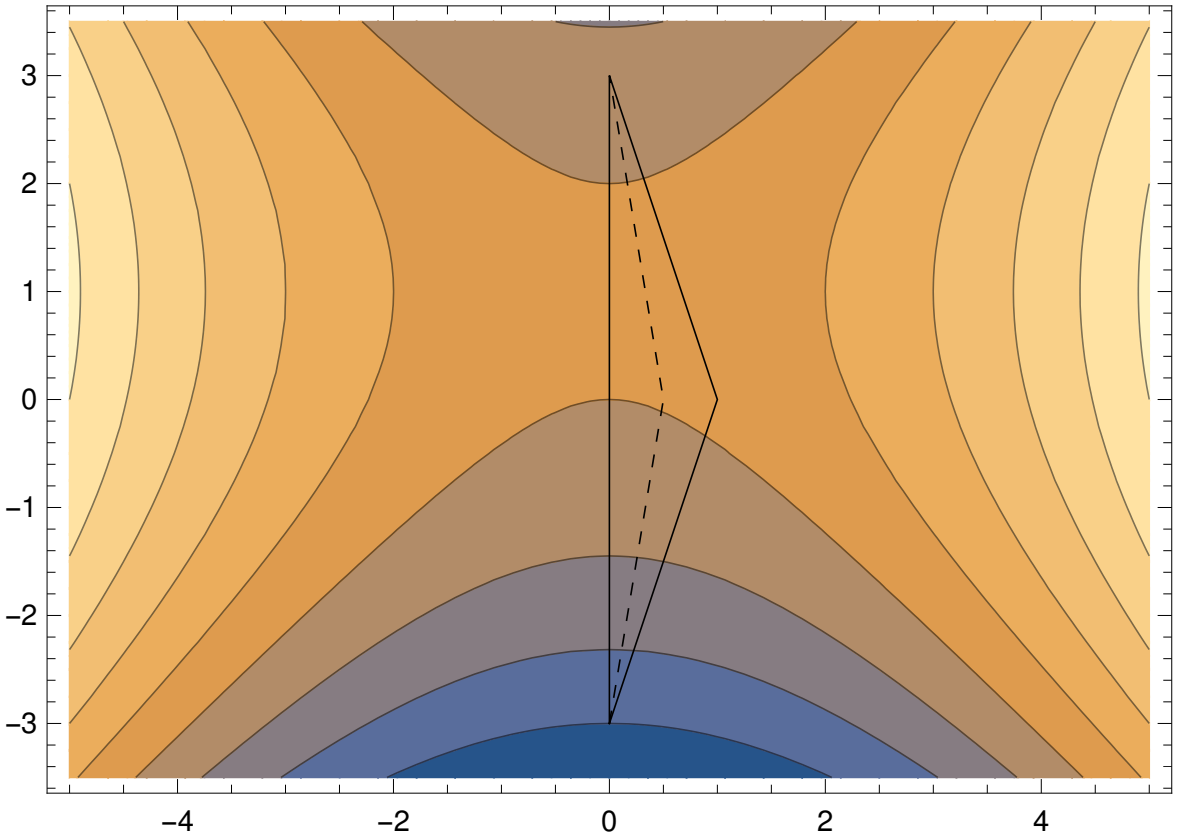


Figure 3.2: One step of the execution of the Nelder–Mead method on the function $(x, y) \mapsto x^2 - y\,(y-2)$ with the initial simplex represented by the vertices $(1, 0)$, $(0, -3)$ and $(0, 3)$. The reflection point for the worst vertex, on the x-axis, yields the same function value as the worst vertex itself, so an inner contraction is tried, with success; this is repeated with the simplex approaching the degenerated simplex represented by the vertices $(0, 0)$, $(0, -3)$ and $(0, 3)$, with clearly none of those vertices corresponding to a minimum or stationary point of the function.

While formalising, [16] shined some light on this, and especially for strictly convex functions, several properties were discovered: in particular for the standard parameters, convergence to a minimiser has been shown for the 1-dimensional functions and the diameter of the simplex converging to zero has been shown for the 2-dimensional functions, without convergence of this simplex – reduced to a point – to a specific point shown. Furthermore, no shrink steps are taken for strictly convex functions:

Lemma 3.1 ([16])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be the function to be minimised and Y^0 the vertices of a non-degenerate starting simplex.*

If f is strictly convex, then no shrink steps are taken in the course of the Nelder–Mead method when applied to f with starting simplex Y^0.

The proof follows from intuitively clear idea that the function value of a strictly convex function for a point corresponding to a convex combination of other points should be strictly

less than the function value for those other points itself. The convex combination for the inside or outside contraction with $\bar{x}_k$ and respectively y_k^n or x_k^{r} yields therefore always a point with a strictly better function value, and the contraction point is accepted; hence, the fail-safe shrink step is not taken.

No other results on the convergence of the original Nelder–Mead method have been discovered to the our best knowledge, until in 2022 [14] was published, in which convergence of the original method is studied for a convergence result under some assumptions on the steps taken. The convergence is studied using notation briefly brought up by [16], by writing the state at each iteration – the vertices – as matrix and the operation on the vertices as matrix too, to make a state change by applying an operation matrix on the state matrix, and making it possible to analyse the behaviour through using linear algebra.

The state matrix, representing the set of vertices that makes up a simplex, is for iteration k written as[1]

$$S^k = \begin{bmatrix} y_k^0 & y_k^1 & \cdots & y_k^n \end{bmatrix}.$$

Since, if no shrink step is taken at iteration k, only a single vertex is changed between the simplex Y^k and Y^{k+1} and this incoming vertex can be described in terms of (3.2), the state matrix for iteration $k+1$ can be described, up to the requirement of fulfilling (3.1) for the order of vertices in accordance to the function value, by

$$S^k T(\tau) = \begin{bmatrix} y_k^0 & y_k^1 & \cdots & \hat{y}_k \end{bmatrix},$$

where the transformation matrix, $T(\tau)$ must fulfil

$$T(\tau) = \begin{bmatrix} I_n & \frac{1+\tau}{n}\mathbf{e} \\ \mathbf{0} & -\tau \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} \tag{3.3}$$

for $\mathbf{e} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$ and for some τ depending on the operation taken in iteration k, as in (3.2). Then, by applying on the product above a permutation matrix $P_j = \begin{bmatrix} e_1 & \ldots & e_{j-1} & e_{n+1} & e_{j+1} & \ldots & e_n & e_j \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$ to make sure the vertices are all correctly ordered according to (3.1) and the tie-breaking rule, we can write

$$S^{k+1} = S^k T(\tau) P_j \tag{3.4}$$

for some τ and j depending on the exact operation taken in iteration k, if no shrink step was taken. Otherwise, should a shrink step be taken, then all-but-one vertices are changed instead, and the new state matrix, up to the ordering, is given by

$$S^k T^{\mathrm{shr}}(\sigma) = \begin{bmatrix} y_k^0 & y_k^0 + \sigma\left(y_k^1 - y_k^0\right) & \cdots & y_k^0 + \sigma\left(y_k^n - y_k^0\right) \end{bmatrix},$$

[1] Note that (3.1) is now assumed to be fulfilled in accordance to the tie-breaking rules.

such that the transformation matrix $T^{\text{shr}}(\sigma)$ is given by

$$T^{\text{shr}}(\sigma) = \begin{bmatrix} 1 & (1-\sigma)\,\mathbf{e}^T \\ \mathbf{0} & \sigma I_n \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} \tag{3.5}$$

and the state matrix for iteration $k+1$ can be written as

$$S^{k+1} = S^k T^{\text{shr}}(\sigma)\, P \tag{3.6}$$

for $P \in \mathbb{R}^{(n+1)\times(n+1)}$ a permutation matrix.

In other words, the state at every iteration $k+1$ can be written as function depending on the state at iteration k and its action, as

$$S^{k+1} = S^k T_k P^k \tag{3.7}$$

for $T_k P^k \in \mathcal{T}$ for $\mathcal{T}$ a finite set of actions, independent of the vertex values; because of this finiteness, the actions can be pre-calculated for a given problem size (consisting of the dimension information), and those actions can be analysed to extract properties from it. We partition this set of actions $\mathcal{T}$ in two disjoint sets, $\mathcal{W}_1$ and $\mathcal{W}_2$, with actions in $\mathcal{W}_1$ corresponding to inside and outside contractions whose incoming vertex is positioned on the first or second position, and shrink steps, and with the remaining actions in $\mathcal{W}_2$.

For convergence properties, in those infinite products, lower (block) triangular matrices would be convenient, with the product of two of such matrices also having this property, and according to the following lemma from [14], a single similarity transformation exists that makes this happen:

Lemma 3.2 ([14])**.** *Given*

$$F = \begin{bmatrix} 1 & -\mathbf{e}^T \\ \mathbf{0} & I_n \end{bmatrix},$$

for all $T_k P^k \in \mathcal{T}$, we can write

$$F^{-1} T_k P^k F = \begin{bmatrix} 1 & \mathbf{0} \\ b_k & C_k \end{bmatrix}.$$

With this lemma, we can state the convergence theorem as found in [14] about the original Nelder–Mead method:

Theorem 3.3 ([14])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ with $n = 1, \ldots, 8$ and assume Y^0 is non-degenerate.*

Let $\|A\|_\vartheta$ be an induced matrix norm such that for all $T_k P^k \in \mathcal{W}_1$, $\|C_k\| < 1$. Then, let constants $0 < q < 1 \leq Q$ be such that $\|C_k\|_\vartheta \leq q < q$ for all $T_k P^k \in \mathcal{W}_1$ and $1 \leq \|C_i\|_\vartheta \leq Q$ for all $T_k P^k \in \mathcal{W}_2$. Lastly, let an $\kappa \in \mathbb{N}$ be such that $q^{1-\kappa} \leq Q \leq q^{-\kappa}$.

Assume that there exists a $\mu \in (0,1)$ such that $t_1(k) \geq \mu k + \kappa t_2(k)$ for $t_j(k)$ counting

how many actions in the iterations up to k were taken from $\mathcal{W}_i$, or assume that only finitely many actions from $\mathcal{W}_i$ are taken in the course of the method. Then, convergence to a point happens for the vertices of the Nelder–Mead method when applying that method on f: there exists an $\hat{y} \in \mathbb{R}^n$ such that for all $i = 0, \ldots, n$,

$$\lim_{k\to\infty} y_k^i = \hat{y}.$$

Despite the strong consequence of Theorem. 3.3, it is not trivial to apply the theorem: one should analyse the steps for a given class of functions to show that steps from $\mathcal{W}_1$ and $\mathcal{W}_2$ are balancing, and thus, the issue of convergence of the original Nelder–Mead method is not settled yet.

3.3 Convergent variants of the Nelder–Mead method

Motivated by the lack of general convergence properties of the original Nelder–Mead method for – relatively to other methods – reasonable functions, several variants of the original Nelder–Mead method have been proposed that try to tackle this by proving convergence for a broader range of functions. Despite those being different, understanding those variants can help understanding the original method too.

3.3.1 Non-expansion Nelder–Mead Method

In the 2012 work [17], a restricted version of the Nelder–Mead method has been studied for two-dimensional problems. In this version, no expansion step is ever taken, and instead of deciding between the expansion and reflection step, the reflection step is always taken, with the primary advantage of forcing the volume of the simplex represented by the vertices to be non-increasing. For this simplified method, convergence to a minimiser can be proved for a specific class of functions:

Theorem 3.4 ([17])**.** *Let $f\colon \mathbb{R}^2 \to \mathbb{R}$ be twice continuously differentiable with bounded level sets and everywhere positive definite Hessian function to be minimised and Y^0 the vertices of a non-degenerate starting simplex.*

Then the algorithm converges to the unique minimiser of f.

According to [17], after experiments for which convergence results could be established, it could be seen that

> expansion steps are almost never taken in the neighbourhood of the optimum. Expansion steps are typically taken early on, forming part of the 'adaptation to the local contours' that constituted the motivation for Nelder and Mead when they originally conceived the algorithm [24]

such that studying this restricted method as a way to understand the original method better could be perceived as less far-fetched than one could initially think.

3.3.2 Fortified-Descent Simplicial Search Method

In 1999, another version was published in [38], with proven convergence to a minimiser for continuously differentiable functions; the fortified-descent simplicial search method, which tries to explicitly overcome the shortcomings of the original Nelder–Mead method where the simplex might become arbitrarily flat with stagnation in the decreasement of the function value before having reached a stationary point.

Highly inspired by the Nelder–Mead method, the new FDSS method has the same concept of steps the Nelder–Mead method has, and thus, a reflection, expansion, inside and outside contraction and shrink step are present in the description. A non-shrink step is taken only if this keeps the interior angles – expressed by the normalised volume – of the simplex (uniformly in the course of the method) away from zero, to avoid the simplex from becoming arbitrarily flat and not spanning the whole space, easily preventing convergence to a minimiser. The shrink step can always be taken – the interior angles stay the same when shrinking – and can for this method too be seen as a last resort step.

Furthermore, as long as no new point is discovered which improves the simplex Y^k that iteration k was entered with, only a shrink step can be taken, leading to the concept of sub-iterations. During a sub-iteration l with simplex Y^{k_l} (with $Y^{k_0} \coloneqq Y^k$), vertices in Y^{k_l} are identified that, if paired up with the vertices of Y^k based on their relative corresponding function value, are good enough, as in belonging to a pair for which the vertex in the old Y^k providing a worse value than in the new Y^{k_l}, and bad, as in not being good enough. If all vertices would be good enough, then we declare the worst vertex nevertheless as bad, to keep the method improving. In every sub-iteration, a non-shrink step (reflection, expansion and inside and outside contraction) can now be tried by keeping the good enough vertices and improving any of the bad vertices to be good enough too. If on first sight no such new vertex can be identified, or this would make the simplex too flat, a shrink step is taken and a new sub-iteration $l + 1$ within the same iteration k is entered; otherwise, a new iteration $k + 1$ is entered.

Lastly, as a new concept, the idea of fortified descent was introduced to avoid the improvement to stagnate prematurely and help the convergence property. This was done by guiding the decrease with a function which has exactly this property.

The version presented in [38] provides multiple degrees of freedom in terms of choosing bad vertices to improve, of how to measure this improvement and generalises furthermore the concept of reflection, expansion and contraction. In the description below as Algorithm 3.2, in line with what was discovered by [38] to yield the best performance when the steps applied as in the original Nelder–Mead method, are those degrees fixed, simplifying the notation furthermore and using the discovered results.

Algorithm 3.2 (Fortified-Descent Simplicial Search Method (FDSS) ([38] with parameters fixed and notation heavily adjusted to fit the current presentation))**.**

Input *Let* $f\colon \mathbb{R}^n \to \mathbb{R}$ *be the function to be minimised and* Y^0 *the vertices of a non-degenerate starting simplex.*

Furthermore, we require the following constants as parameters of the method:

- $0 < \rho$*: the parameter defining the ratio of length for reflection;*
- $1 < \chi$*: the parameter defining the ratio of length for lengthening the reflection for expansion;*
- $0 < \gamma < 1$*: the parameter defining the ratio of length for shortening the reflection for contraction;*
- $0 < \sigma < 1$*: the parameter defining the ration of length for shrinking;*
- $0 < \nu \leq \operatorname{von}\left(Y^0\right)$*: the parameter defining the threshold for the simplex being used being too flat*[a]*;*
- $0 < \theta_{\mathrm{r}} < 1$*: parameter related to the acceptance of a reflection step;*
- α, β*: two continuous functions (*$\phi\colon [0,\infty) \to [0,\infty)$ *and* $\lim_{t\to 0} \phi(t)/t = 0$ *for* $\phi = \alpha, \beta$ *and* $\forall c > 0, \inf_{t\geq c} \alpha(t) > 0$*) forcing the fortified descent.*

Output *The point corresponding to the lowest function value of* f *in* $\mathbb{R}$ *found so far at iteration* k *for* $k \to \infty$*:* y_k^0*.*

Initialisation *Set* $k = 0$*.*

Step 1 (iteration initialisation) *Set* $l = 0$ *and* $Y^{k_l} = Y^k$*.*

Step 2 (restructuring) *Order the vertices of* Y^{k_l} *based on their function value in such way that it holds that*

$$f\left(y_{k_l}^i\right) \leq f\left(y_{k_l}^j\right), \qquad \forall i \leq j$$

and let m_{k_l} *be the greatest index (*$0 \leq m_{k_l} \leq n-1$*) such that* $f\left(y_{k_l}^i\right) \leq f\left(y_k^i\right)$ *for all* $0 \leq i \leq m_{k_l}$*.*

Compute the centroid of the best m_{k_l} *points* $y_{k_l}^0, \ldots, y_{k_l}^{m_{k_l}-1}$ *and it's corresponding 'centroid function value':* $\bar{x}_{k_l} \coloneqq \sum_{i=0}^{m_{k_l}-1} y_{k_l}^i / m_{k_l}$ *and* $\bar{x}_{k_l}^f \coloneqq \sum_{i=0}^{m_{k_l}-1} f\left(y_{k_l}^i\right)/m_{k_l}$*.*

Let Δ_{k_l} *be the diameter of the current simplex:* $\Delta_{k_l} = \operatorname{diam}\left(Y^{k_l}\right)$*.*

Step 3 (reflecting) *Compute the points* $x_{k_l}^{\mathrm{r}_0}, \ldots, x_{k_l}^{\mathrm{r}_{n-m_{k_l}}}$ *resulted from a reflection of the remaining worst points* $y_{k_l}^{m_{k_l}}, \ldots, y_{k_l}^n$ *over the computed centroid* $\bar{x}_{k_l}$

$$x_{k_l}^{\mathrm{r}_i} \coloneqq \bar{x}_{k_l} + \rho\left(\bar{x}_{k_l} - y_{k_l}^{m_{k_l}+i}\right), \qquad \forall i, 0 \leq i \leq n - m_{k_l}$$

and evaluate those on f as $f\left(x_{k_l}^{\mathrm{r}_i}\right)$. Let the simplex represented by the best m_{k_l} vertices and the $n - m_{k_l}$ new vertices be

$$Y_{\mathrm{r}}^{k_l} = \left\{ y_{k_l}^0, \ldots, y_{k_l}^{m_{k_l}-1}, x_{k_l}^{\mathrm{r}_0}, \ldots, x_{k_l}^{\mathrm{r}_{n-m_{k_l}}} \right\}.$$

If $\operatorname{von}\left(Y_{\mathrm{r}}^{k_l}\right) \geq \nu$ *and*

$$\min_{0 \leq i \leq n-m_{k_l}} f\left(x_{k_l}^{\mathrm{r}_i}\right) \leq f\left(y_{k_l}^{m_{k_l}-1}\right) - \alpha\left(\Delta_{k_l}\right) \tag{3.8}$$

$$\min_{0 \leq i \leq n-m_{k_l}} f\left(x_{k_l}^{\mathrm{r}_i}\right) \leq f\left(y_{k_l}^{m_{k_l}-1}\right) - \theta_{\mathrm{r}}\left(f\left(y_{k_l}^n\right) - \bar{x}_{k_l}^f\right) + \beta\left(\Delta_{k_l}\right) \tag{3.9}$$

is fulfilled, continue with Step 4 by trying an expansion in addition to the successful reflection. Otherwise, continue with Step 5 and try a contraction instead for a reflection.

Step 4 (expanding) *Calculate expansions points:*

$$x_{k_l}^{\mathrm{e}_i} \coloneqq \bar{x}_{k_l} + \chi\rho\left(\bar{x}_{k_l} - y_{k_l}^{m_{k_l}+i}\right), \qquad \forall i, 0 \leq i \leq n - m_{k_l}.$$

Let the simplex represented by the best m_{k_l} vertices and the $n - m_{k_l}$ new vertices be

$$Y_e^{k_l} = \left\{ y_{k_l}^0, \ldots, y_{k_l}^{m_{k_l}-1}, x_{k_l}^{\mathrm{e}_0}, \ldots, x_{k_l}^{\mathrm{e}_{n-m_{k_l}}} \right\}.$$

If $\operatorname{von}\left(Y_e^{k_l}\right) \geq \nu$ *and*

$$\min_{0 \leq i \leq n-m_{k_l}} f\left(x_{k_l}^{\mathrm{e}_i}\right) \leq f\left(y_{k_l}^{m_{k_l}-1}\right) - \alpha\left(\Delta_{k_l}\right) \tag{3.10}$$

$$\min_{0 \leq i \leq n-m_{k_l}} f\left(x_{k_l}^{\mathrm{e}_i}\right) \leq f\left(y_{k_l}^{m_{k_l}-1}\right) - \theta_{\mathrm{r}}\left(f\left(y_{k_l}^n\right) - \bar{x}_{k_l}^f\right) + \beta\left(\Delta_{k_l}\right) \tag{3.11}$$

is fulfilled, accept the expansion by setting $Y^{k+1} = Y_e^{k_l}$; otherwise, set $Y^{k+1} = Y_{\mathrm{r}}^{k_l}$. In any case, set k to $k+1$ and continue with Step 1 in a new iteration.

Step 5 (contracting) *Calculate one of the contractions, depending on which seems the most promising:*

Step 5a (inside contraction) *If* $\min_{0 \leq i \leq n-m_{k_l}} f\left(x_{k_l}^{\mathrm{r}_i}\right) < f\left(y_{k_l}^m\right)$, *calculate*

$$x_{k_l}^{\mathrm{c}_i} \coloneqq \bar{x}_{k_l} + \gamma\rho\left(\bar{x}_{k_l} - y_{k_l}^{m_{k_l}+i}\right), \qquad \forall i, 0 \leq i \leq n - m_{k_l}$$

Step 5b (outside contraction) *Otherwise, calculate*

$$x_{k_l}^{\mathrm{c}_i} \coloneqq \bar{x}_{k_l} - \gamma\left(\bar{x}_{k_l} - y_{k_l}^{m_{k_l}+i}\right), \qquad \forall i, 0 \leq i \leq n - m_{k_l}$$

Let the simplex represented by the best m_{k_l} vertices and the $n - m_{k_l}$ new vertices be

$$Y_{\mathrm{c}}^{k_l} = \left\{ y_{k_l}^0, \ldots, y_{k_l}^{m_{k_l}-1}, x_{k_l}^{\mathrm{c}_0}, \ldots, x_{k_l}^{\mathrm{c}_{n-m_{k_l}}} \right\}$$

which is indexed in such a way such that, by letting $y_{k_l}^{\mathrm{c}_i}$ be an element of $Y_{\mathrm{c}}^{k_l}$, it holds that

$$f\left(y_{k_l}^{\mathrm{c}_i}\right) \leq f\left(y_{k_l}^{\mathrm{c}_j}\right), \qquad \forall i \leq j.$$

Then, if $\operatorname{von}\left(Y_{\mathrm{c}}^{k_l}\right) \geq \nu$ *and it holds that*

$$\begin{aligned} f\left(y_{k_l}^{\mathrm{c}_{m_{k_l}+1}}\right) &\leq f\left(y_k^{m_{k_l}+1}\right) \\ \sum_{i=0}^{m_{k_l}+1} f\left(y_{k_l}^{\mathrm{c}_i}\right) &\leq \sum_{i=0}^{m_{k_l}+1} f\left(y_k^i\right) - \alpha\left(\Delta_{k_l}\right), \end{aligned} \tag{3.12}$$

and thus, intuitively speaking, if $Y_{\mathrm{c}}^{k_l}$ yields no worsening of the already best m_{k_l} vertices and one worse vertex of the original simplex used when iteration k was entered, and an improvement actually, then accept the new simplex by setting $Y^{k+1} = Y_{\mathrm{c}}^{k_l}$, setting k to $k+1$ and continuing with Step 1.

Step 6 (shrinking) *If no new simplex is accepted so far, a shrink step is taken by calculating*

$$x_{k_l}^{\mathrm{s}_i} \coloneqq y_{k_l}^0 + \sigma\left(y_{k_l}^i - y_{k_l}^0\right), \qquad \forall i, 0 \leq i \leq n$$

and writing by the new shrunken simplex as

$$Y_{\mathrm{s}}^{k_l} = \left\{ x_{k_l}^{\mathrm{s}_0}, x_{k_l}^{\mathrm{s}_1}, \ldots, x_{k_l}^{\mathrm{s}_n} \right\}.$$

If

$$\min_{0 \leq i \leq n} f\left(x_{k_l}^{\mathrm{r}_i}\right) \leq f\left(y_k^0\right) - \alpha\left(\Delta_{k_l}\right), \tag{3.13}$$

accept the shrinkage by setting $Y^{k+1} = Y_{\mathrm{s}}^{k_l}$, setting k to $k+1$ and continuing with Step 1. Otherwise, set $Y^{k_{l+1}} = Y_{\mathrm{s}}^{k_l}$, setting l to $l+1$ and continue with Step 2.

[a] In the implementation in Implementation A.1, the factor $\frac{1}{n!}$ is removed from the definition of the normalised volume von as it is a quickly-growing constant that can be easily compensated by adjusting ν, since it's upper bound is affected too.

A visualisation of the change of simplices through the parameter space can be found in Figure 3.3 for a 2-dimensional function.

An even more simplified version can be found in [12], which differs from the version stated above roughly by, after a failed initial reflection step with $m_{k_0} = n - 1$, continuing with a new sub-iteration by setting $m_{k_1} = 1$, hereby rotating the whole simplex in what is called a

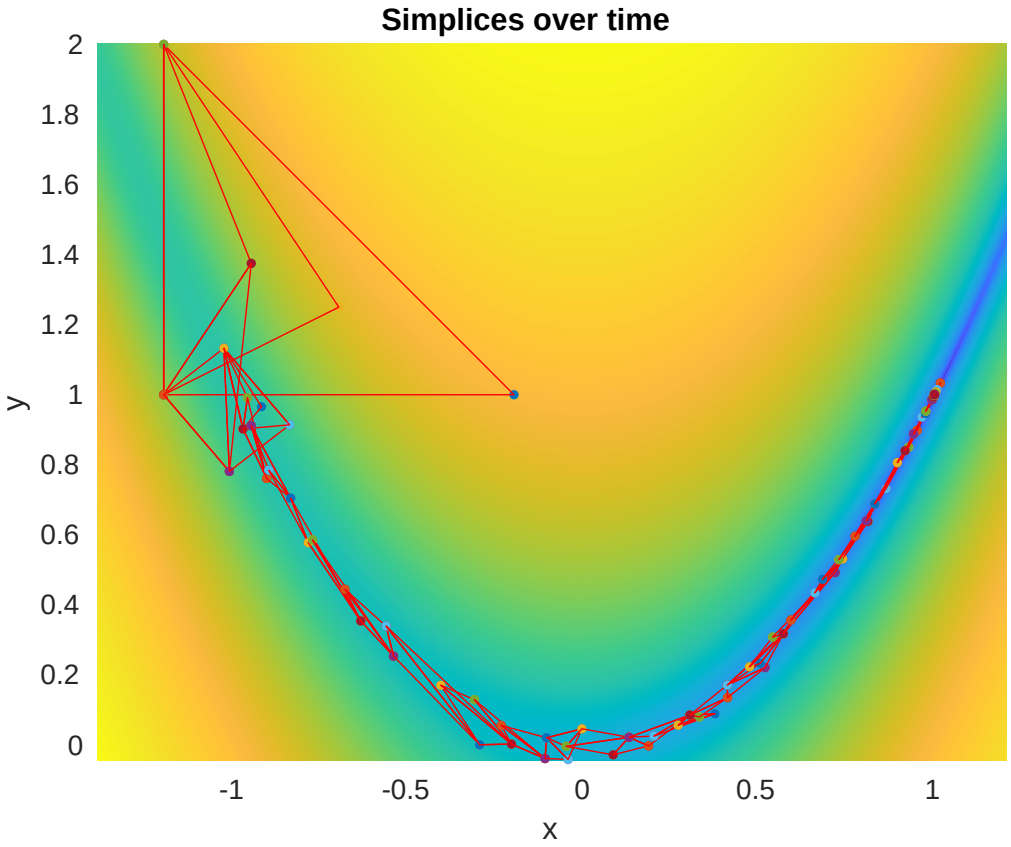


Figure 3.3: Simplices used over the course of the FDSS method using Implementation A.1 of the so-called Rosenbrock function $(x, y) \mapsto 100\left(y - x^2\right)^2 + (1 - x)^2$ as found in [30] with starting point $(-1.2, 1)$, for which the start simplex used here is represented by the vertices $\{(-1.2, 1), (-0.2, 1), (-1.2, 2)\}$. In it's initial stage while converging to the minimum at $(1, 1)$, two inside contraction steps, followed by an outside contraction, an inside contraction and reflection step are identifiable.

safeguard rotation.

A convergence theorem for this method is then the following, stated in terms of the first computed centroid at every iteration:

Theorem 3.5 ([38])**.** *Let $f \in \mathcal{C}^1$ be the function to be minimised and Y^0 the vertices of a non-degenerate starting simplex. Assume that f is uniformly bounded from below on $\mathbb{R}^n$.*

Define $L(Y) := \{x \in \mathbb{R}^n : f(x) \leq \min_{y \in Y} f(y)\}$.

- *If $L\left(Y^0\right)$ is bounded, then either the method quits at some iteration k with a stationary point of f or the method does not quit and at least one accumulation point of $(\bar{x}_{k_0})_{k \in \mathbb{N}}$ is a stationary point of f.*
- *If f is uniformly continuous on $L\left(Y^0\right)$, then either the method quits at some iteration k with a stationary point of f or the method does not quit and every accumulation point of $(\bar{x}_{k_0})_{k \in \mathbb{N}}$ is a stationary point of f.*

In the description of Algorithm 3.2, multiple fortified-descent criteria are present. The first criterion for reflection (3.8) (and the very similar one for expansion (3.8)), the criterion for contraction (3.12) and the criterion for a shrink (3.13) all boil down to, as shown in [38],

$$f\left(y_{k+1}^{i}\right) \leq f\left(y_{k}^{i}\right)$$
$$\sum_{i=0}^{m} f\left(y_{k+1}^{i}\right) \leq \sum_{i=0}^{m} f\left(y_{k}^{i}\right) - \alpha\left(\Delta_{k_l}\right)$$

as acceptance criterion that should be fulfilled to leave iteration k for some m that depends on m_{k_l} for l the last sub-iteration in iteration k, and this is used for proving that the diameters approaches zero (under the conditions stated in Theorem 3.5. The two stronger criteria for reflection (3.8) and (3.9) (and the very similar ones for expansion (3.10) and (3.11)) are used for proving the convergence to the stationary points.

As reported by [38] for four different function minimisations for which original Nelder–Mead method converges, the performance of Algorithms 3.1 and 3.2 are very comparable: both methods take either the same number of function evaluations with the same function value upon termination, of slightly better values. In Chapter 6, more comparison results will be reported.

3.4 Termination criteria

In the previous sections, we have seen the original Nelder–Mead method and different minimisation methods based on the Nelder–Mead method. As minimisation methods in their current description, the algorithms do not terminate, but instead keep trying to get closer to a stationary point, in general to the expense of evaluating the (possible costly) function f more often. In practice, one would like to receive a definitive answer to the minimisation problem as approximation of a local minimum of the original function after some finite amount of time, and thus, a termination criterion is desired to decide whether one can stop looking for a (better) solution.

One type of termination criteria could be based on the execution of the method itself, and could consist of termination upon having executed a certain number of iterations or having evaluated the function f a certain number of times; if evaluation of this function takes some resources, a certain budget might be accounted, and termination of the method occurs when all the resources are taken. Another type of termination criteria could be based on the actual points in the simplex; how they are positioned in the parameter space and what function value corresponds to those could make us decide that this pattern corresponds to being close enough to a minimiser for termination of execution of the method. In two following subsections, we discuss several of those, starting with criteria for the FDSS method where the simplex is guaranteed non-degenerate, and followed by more heuristic-like criteria.

3.4.1 Criteria for Fortified-Descent Simplicial Search Method

For a stopping criterion to be used with the FDSS, we will measure the gradient of the objective function f at the current best point based on the already evaluated points, as done by [38], but shown there without intermediate steps. For this, in line with the convergence statement from Theorem 3.5, we assume f to be differentiable, and compare the calculated gradient it with zero, Then, if the gradient is small enough and comes below a certain threshold, we can declare the method as having been successful in finding a stationary point, and terminate execution of it.

For showing this results here, we recall the definition of the directional derivative from earlier:

$$f'(x;d) \coloneqq \lim_{t\to 0^+} \frac{f(x+td)-f(x)}{t}. \tag{2.1}$$

In coming up with an expression for the gradient at $y_{k_l}^0$, we will then use the already evaluated function values at the $n+1$ points in the simple Y^{k_l}, by considering the directional derivative in the (scaled) direction of $y_{k_l}^i - y_{k_l}^0$, for $i = 1, \ldots, n$:

$$\begin{aligned}
f'\left(y_{k_l}^0; \frac{y_{k_l}^i - y_{k_l}^0}{\left\|y_{k_l}^i - y_{k_l}^0\right\|}\right) &= \lim_{t\to 0^+} \frac{f\left(y_{k_l}^0 + t\frac{y_{k_l}^i - y_{k_l}^0}{\left\|y_{k_l}^i - y_{k_l}^0\right\|}\right) - f\left(y_{k_l}^0\right)}{t} \\
&= \lim_{\left\|y_{k_l}^i - y_{k-l}^0\right\|\to 0^+} \frac{f\left(y_{k_l}^0 + \left\|y_{k_l}^i - y_{k_l}^0\right\| \frac{y_{k_l}^i - y_{k_l}^0}{\left\|y_{k_l}^i - y_{k_l}^0\right\|}\right) - f\left(y_{k_l}^0\right)}{\left\|y_{k_l}^i - y_{k_l}^0\right\|} \\
&= \lim_{\left\|y_{k_l}^i - y_{k_l}^0\right\|\to 0^+} \frac{f\left(y_{k_l}^i\right) - f\left(y_{k_l}^0\right)}{\left\|y_{k_l}^i - y_{k_l}^0\right\|}.
\end{aligned}$$

Under the assumption of the diameter converging to zero in the course of the iterative method for $k \to \infty$, which follows under certain assumptions from Theorem 3.5, by using the relation (2.3) between the directional derivative and the derivative, using also the definition of the gradient – we can write, thus for $k \to \infty$,

$$\begin{aligned}
f'\left(y_{k_l}^0; \frac{y_{k_l}^i - y_{k_l}^0}{\left\|y_{k_l}^i - y_{k_l}^0\right\|}\right) &= \nabla f\left(y_{k_l}^0\right)^T \frac{y_{k_l}^i - y_{k_l}^0}{\left\|y_{k_l}^i - y_{k_l}^0\right\|} \\
\frac{f\left(y_{k_l}^i\right) - f\left(y_{k_l}^0\right)}{\left\|y_{k_l}^i - y_{k_l}^0\right\|} &= \nabla f\left(y_{k_l}^0\right)^T \frac{y_{k_l}^i - y_{k_l}^0}{\left\|y_{k_l}^i - y_{k_l}^0\right\|} \\
f\left(y_{k_l}^i\right) - f\left(y_{k_l}^0\right) &= \nabla f\left(y_{k_l}^0\right)^T \left(y_{k_l}^i - y_{k_l}^0\right).
\end{aligned}$$

By combining those n equations and assuming in addition all $y_{k_l}^0 - y_{k_l}^i$ to be linearly inde-

pendent, and the normalised volume of the simplex thus being non-zero, we can write, for $k \to \infty$ again,

$$\begin{bmatrix} f\left(y_{k_l}^1\right) - f\left(y_{k_l}^0\right) \\ \vdots \\ f\left(y_{k_l}^n\right) - f\left(y_{k_l}^0\right) \end{bmatrix} = \nabla f\left(y_{k_l}^0\right)^T \begin{bmatrix} y_{k_l}^1 - y_{k_l}^0 \\ \vdots \\ y_{k_l}^n - y_{k_l}^0 \end{bmatrix},$$

or, written differently,

$$\nabla f\left(y_{k_l}^0\right) = \begin{bmatrix} \left(y_{k_l}^1 - y_{k_l}^0\right)^T \\ \vdots \\ \left(y_{k_l}^n - y_{k_l}^0\right)^T \end{bmatrix}^{-1} \begin{bmatrix} f\left(y_{k_l}^1\right) - f\left(y_{k_l}^0\right) \\ \vdots \\ f\left(y_{k_l}^n\right) - f\left(y_{k_l}^0\right) \end{bmatrix}. \tag{3.14}$$

as thus also reported by [38]. In line with the assumption of the diameter converging to zero, a stopping criterion might then consist of demanding for $\operatorname{diam}\left(Y^{k_l}\right) \leq \epsilon_1$ the norm of the right-hand side of (3.14) being smaller than ϵ_2, for both $\epsilon_1, \epsilon_2 > 0$ (with possibly $\epsilon_1 = \epsilon_2$).

However, with (3.14) involving the inverse of a square matrix, this stopping criterion can be considered to be computationally heavy, and in this same [38], an alternative expression of

$$\begin{bmatrix} \left(f\left(y_{k_l}^1\right) - f\left(y_{k_l}^0\right)\right) / \left\|y_{k_l}^1 - y_{k_l}^0\right\| \\ \vdots \\ \left(f\left(y_{k_l}^n\right) - f\left(y_{k_l}^0\right)\right) / \left\|y_{k_l}^n - y_{k_l}^0\right\| \end{bmatrix}$$

was suggested, where the two norms of these expressions were reported to differ 'only by a constant factor'.

3.4.2 Other criteria

For the FDSS method, we got lucky by having affinely-independent points around the best point in each (sub)iteration and being able to derive a direct approximation for the gradient of this. We aren't that lucky for the Nelder–Mead method in general though, as the simplex can become arbitrarily flat, as we have seen earlier. Several heuristics have been found helpful to guide the decision of quitting the algorithm.

In [41, Chp. 2.4], a comparison was made between two earlier used stopping criteria and a newly introduced one. We add a different existing criterion to this discussion, found below:

- The criterion mentioned in the original Nelder–Mead publication [24] is inspired by what they call the 'standard error' and is given by

$$\sqrt{\frac{1}{n}\sum_{i=0}^{n}\left(f\left(y_k^i\right) - \bar{y}_k^f\right)^2} < \epsilon_1$$

for $\bar{y}_k^f \coloneqq \sum_{i=0}^{n} f\left(y_k^i\right)/(n+1)^2$ and $\epsilon_1 > 0$ being the threshold parameter.
An careful observation reveals that this stopping criterion is fulfilled for vertices on the same contour line, while being undecided of the distance between the points, allowing significant level changes in the interior of the simplex, making it possible for vertices to be maxima instead too.

- In [25], the different criterion of

$$f\left(y^n\right) - f\left(y^0\right) \leq \epsilon_2 \text{ and } \frac{1}{n}\sum_{i=0}^{n}\left\|y_k^i - y_{k+1}^i\right\| < \epsilon_3$$

was stated, for $\epsilon_2, \epsilon_3 > 0$, which, after having tested different not-mentioned stopping criteria, led to

> the most reliable relation between the convergence parameters and the accuracy attained.

This time, not only the function value is taken into account, but also the amount of changes made in an iteration, where only a small change would better represent a stationary point.

- With the previous criterion in mind and the observation that the main value there depends on the current size of the simplex, given the constants determining the position of the new vertex or new vertices, [41] introduced a new criterion exactly based on this size as

$$\frac{1}{\Delta^k}\max_{1\leq i\leq n}\left\|y_k^i - y_k^0\right\| \leq \epsilon_4$$

for $\Delta^k = \max\left\{1, \left\|y_k^0\right\|\right\}$ and $\epsilon_4 > 0$, with the explanation that

> [this] measures the relative size of ths simplex by considering the length of the longest side incident to $[y_k^0]$, and stops when this length becomes smaller than some preset value.

As motivation for this different criterion was mentioned that the main value in the previous criterion takes greater values for a shrink step than for a contraction step, while shrink steps are claimed to take place more often in the neighbourhood of a local minimiser, with the shrink step being portrayed as last resort here too. Thus, the value of ϵ_5 may need to be set greater than desired, such that the method also terminates prematurely, upon a contraction step.

- Lastly, a different stopping criterion is used in the MATLAB implementation, [37], which could thought of as combinations the previous proposals. Here, the stopping

[2]The exact value for what is assumed to be the mean is not given in [24]. In [41], the normalisation factor for both the mean and the variance is given by $\frac{1}{n}$ instead of $\frac{1}{n+1}$ without further discussion. The use of $\frac{1}{n}$ in the variance, as done in [24] too, gives rise interpretation of the points in the simplex as a (limited) sample of points in some neighbourhood for which the main value in the criterion gives an unbiased estimator of the variance.

criterion consists of

$$\max_{1\leq i\leq n} \left\|y_k^i - y_k^0\right\|_\infty < \epsilon_5 \text{ and } \max_{1\leq i\leq n} \left|f\left(y_k^i\right) - f\left(y_k^0\right)\right| < \epsilon_6$$

for $\epsilon_5, \epsilon_6 > 0$. Thus, in this case, the simplex needs to be small enough, while the points corresponding function values are close to the best too, indeed providing the view of a locally flat surrounding if the criterion holds.

Chapter 4

Trust-Region Methods

An established class of minimisation methods can be described as trust-region method. In the course of this iterative-based method, steps through the parameter space are made by locally minimising an approximation to the objective function – chosen in such a way to allow this minimisation to take place in an easier way than minimisation of the original function would have – with the ultimate goal of converging to a local minimiser of the objective function. By the nature of a local approximation, the quality of this approximation degenerates in general the further one is away from the local area we based the approximation on — and that contains the iteration point – and thus, a trust region is defined in every step in which one hopes to trust the model enough to make a decent descent step.

In the field of derivative-based optimisation, a satisfying local approximation can be obtained by creating a Taylor polynomial of sufficient low degree based on the original function around the iteration point. However, to construct a non-trivial Taylor polynomial, derivative information of this function needs to be accessed, which is not possible in the derivative-free setting. In this chapter, a local approximation will instead be derived by evaluating the objective function on a set of finitely many points and creating an approximation based on those points, granted that those points fulfil some geometric requirements, leading to the so-called *conditional trust-region method*.

We begin this chapter by assuming the existence of an approximation function with the required general properties and by describing a minimisation method based on this class of approximation functions. Afterwards, we describe how to construct such an approximation function for use in the minimisation method and how to minimise those functions locally. For simplification in notation, we assume the vector norm to be the ℓ^2 norm, e.g. $\|\cdot\| = \|\cdot\|_2$.

4.1 Fully linear and fully quadratic models

We start by formalising the idea of a model function as approximation to the objective function in a trust region – which we in the rest of this chapter will model in each iteration

as a closed ball around a base point – by introducing a specific class of functions. In each iteration, we should be able to access a model function from that class in a finite and uniformly bounded (with respect to the trust region) number of steps, and the error between this model function and the objective function should, apart from a constant, depend only on the size of the trust region, where a smaller trust region should provide a smaller error.

For the definition of this general class, we assume that our objective function f is sufficiently differentiable (with Lipschitz continuous gradient) in an open set containing the points that can be evaluated in the method. For the starting iteration point x_0, the set $L(x_0)$ is defined as set that includes all potential further iteration points, which will be extended outwards by the maximum trust-region radius to include also all points that can be evaluated in the course of the method. Based on this assumption, depending on the exact classification of differentiability of the objective function, [12] defines a class of functions that have the desired properties mentioned in the previous paragraph.

For once differentiable functions, we consider more exactly functions fulfilling the following assumption:

Assumption 4.1 ([12])**.** *For a set of interest S and maximum trust-region radius $\Delta_{\max}$, assume $f\colon \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable with Lipschitz continuous gradient in an open set containing $\bigcup_{x\in S} B(x;\Delta_{\max}) \subset \mathbb{R}^n$, being an extended version of S.*

The actual class of functions fulfilling the properties mentioned in the first paragraph is then defined as follows:

Definition 4.1 (based on [12] with different presentation)**.** Let f satisfy Assumption 4.1 for a set of interest S and maximum trust-region radius $\Delta_{\max}$.

We call a specific set of functions $\mathcal{M} \subset \mathcal{C}^1(\mathbb{R}^n, \mathbb{R})$ a *fully linear class of models* with parameters $(\kappa_f, \kappa_g, \nu_1)$ if the following two conditions hold:

1. For all centre points $x \in S$ and trust-region radii $\Delta \in (0, \Delta_{\max}]$, there exists a function $m_{x,\Delta} \in \mathcal{M}$, with Lipschitz continuous gradient and the corresponding Lipschitz constant bounded by ν_1 such that

   - the error between the gradient of the model and that of the objective satisfies

   $$\|\nabla f(x+s) - \nabla m_{x,\Delta}(x+s)\| \leq \kappa_g \Delta \qquad \forall s \in B(\mathbf{0};\Delta), \tag{4.1}$$

   and

   - the error between the model and the objective satisfies

   $$|f(x+s) - m_{x,\Delta}(x+s)| \leq \kappa_f \Delta^2 \qquad \forall s \in B(\mathbf{0};\Delta). \tag{4.2}$$

   We call a model function $m \in \mathcal{M}$ fully linear on $B(x;\Delta)$ with parameters $(\kappa_f, \kappa_g, \nu_1)$ if the conditions above on the errors hold for a specific $x \in S$ and $\Delta \in (0, \Delta_{\max}]$.

2. For all $m \in \mathcal{M}$, $x \in S$ and $\Delta \in (0, \Delta_{\max}]$, we are able to, in a finite and uniformly bounded (with respect to x and Δ) number of steps,

   - either certify that the model function m is fully linear on $B(x; \Delta)$ with parameters $(\kappa_f, \kappa_g, \nu_1)$, in which case we call m certifiable fully linear (CFL) on $B(x; \Delta)$ with parameters $(\kappa_f, \kappa_g, \nu_1)$,
   - or[1] a find another function $\tilde{m} \in \mathcal{M}$ instead that is fully linear on $B(x; \Delta)$ with parameters $(\kappa_f, \kappa_g, \nu_1)$.

Thus, given a fully linear class $\mathcal{M}$ with some parameters, for every point in our set of interest S, we are able to obtain a function $m \in \mathcal{M}$ whose error bounds for the value and the gradient depend respectively quadraticly and linearly on the radius, Δ, of the region of interest, with no restrictions for points outside this radius.

We note that Definition 4.1 does not require all functions $m \in \mathcal{M}$ to be fully linear for any base point or trust-region radius; on the contrary, it can easily be seen that given a fully linear class $\mathcal{M}$, any set of functions $\tilde{\mathcal{M}}$ such that $\mathcal{M} \subset \tilde{\mathcal{M}}$ is a fully linear class, and $\mathcal{C}^1$ itself is also a fully linear class too – as long as there exists at least a fully linear class at all. As we will later see when covering minimisation algorithms using fully linear and fully quadratic classes though, we require the functions in the classes to be (approximately) minimisable in preferably some efficient way – something that clearly can't be accomplished by choosing $\mathcal{C}^1$ as class. The challenge is thus not per se to find a class in general, but to find a class with several other nice properties, as being easily (approximately) minimisable.

Similarly to the definition of a fully linear class, a fully quadratic class of models with parameters $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$ can be defined for later usage in second-order minimisation methods. We assume now the following:

Assumption 4.2 ([12])**.** *For a set of interest S and maximum trust-region radius $\Delta_{\max}$, assume $f \colon \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable with Lipschitz continuous Hessian in an open set containing $\bigcup_{x \in S} B(x; \Delta_{\max}) \subset \mathbb{R}^n$ being an extended variation of S.*

Then, this stronger class of functions is defined as follows:

Definition 4.2 (based on [12] with different presentation)**.** Let f satisfy Assumption 4.2 for a set of interest S and maximum trust-region radius $\Delta_{\max}$.

We call a specific set of functions $\mathcal{M} \subset \mathcal{C}^2(\mathbb{R}^n, \mathbb{R})$ a *fully quadratic class of models* with parameters $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$ if the following two conditions hold:

1. For all centre points $x \in S$ and trust-region radii $\Delta \in (0, \Delta_{\max}]$, there exists a function $m_{x,\Delta} \in \mathcal{M}$, with Lipschitz continuous Hessian and the corresponding Lipschitz constant bounded by ν_2 such that

[1] Note that we did not require all functions that are fully linear to be certified as such (in fact, we don't even require the result of the certification attempt to be consistent upon re-certification), and thus, this either–or construction could equally well be written as or–or construction.

- the error between the Hessian of the model and that of the objective satisfies

$$\left\|\nabla^2 f(x+s) - \nabla^2 m_{x,\Delta}(x+s)\right\| \leq \kappa_h \Delta \qquad \forall s \in B(\mathbf{0};\Delta), \tag{4.3}$$

- the error between the gradient of the model and that of the objective satisfies

$$\|\nabla f(x+s) - \nabla m_{x,\Delta}(x+s)\| \leq \kappa_g \Delta^2 \qquad \forall s \in B(\mathbf{0};\Delta), \tag{4.4}$$

  and

- the error between the model and the objective satisfies

$$|f(x+s) - m_{x,\Delta}(x+s)| \leq \kappa_f \Delta^3 \qquad \forall s \in B(\mathbf{0};\Delta). \tag{4.5}$$

We call a model function $m \in \mathcal{M}$ fully quadratic on $B(x;\Delta)$ with parameters $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$ if the conditions above on the errors hold for a specific $x \in S$ and $\Delta \in (0, \Delta_{\max}]$.

2. For all $m \in \mathcal{M}$, $x \in S$ and $\Delta \in (0, \Delta_{\max}]$, we are able to, in a finite and uniformly bounded (with respect to x and Δ) number of steps,

   - either certify that the model function m is fully quadratic on $B(x;\Delta)$ with parameters $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$, in which case we call m certifiable fully quadratic (CFL) on $B(x;\Delta)$ with parameters $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$,
   - or[2] a find another function $\tilde{m} \in \mathcal{M}$ instead that is fully quadratic on $B(x;\Delta)$ with parameters $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$.

Compared to the definition of a fully linear class, the following changes are made for the definition of a fully quadratic class:

- The parameters tuple has been changed from $(\kappa_f, \kappa_g, \nu_1)$ to $(\kappa_f, \kappa_g, \kappa_h, \nu_2)$;
- The functions in $\mathcal{M}$ needs to be of the class $\mathcal{C}^2$ instead of $\mathcal{C}^1$ and model function $m_{x,\Delta}$ now needs to have a Lipschitz continuous Hessian instead of gradient, with the corresponding constant bounded by ν_2 instead of ν_1;
- An additional condition (4.3) on the error has been added, with a condition on the Hessian. Condition (4.1) has been changed into condition (4.4) by replacing Δ with Δ^2 on the right-hand side, and similarly, condition (4.2) has been changed into (4.5) by replacing Δ^2 with Δ^3 on the right-hand side.

We now state two lemmata, saying that a function that is fully linear (or fully quadratic) for a tuple of parameters on a specific ball, is also fully linear (or fully quadratic respectively) for the same tuple of parameters on a ball with a bigger radius and the same centre point.

[2] See Footnote 1, with 'fully linear' replaced by 'fully quadratic'.

We will later use this when describing ways of coming up with a fully linear or fully quadratic model.

Lemma 4.1 ([12])**.** *Consider a function f satisfying Assumption 4.1 for a set of interest S and maximum trust-region radius $\Delta_{\max}$ and consider a function m being fully linear on $B\left(x;\bar{\Delta}\right)$ with parameters $(\kappa_f,\kappa_g,\nu_1)$ with $x\in S$ and $\bar{\Delta}\leq\Delta_{\max}$.*

Assume, without loss of generality, that κ_g is at least the sum of ν_1 and the Lipschitz constant of the gradient of f, and that $\kappa_f\geq\frac{1}{2}\kappa_g$.

Then m is fully linear on $B(x;\Delta)$ with the same parameters $(\kappa_f,\kappa_g,\nu_1)$ for all $\Delta\in\left[\bar{\Delta},\Delta_{\max}\right]$.

Lemma 4.2 ([12])**.** *Consider a function f satisfying Assumption 4.2 for a set of interest S and maximum trust-region radius $\Delta_{\max}$ and consider a function m being fully quadratic on $B\left(x;\bar{\Delta}\right)$ with parameters $(\kappa_f,\kappa_g,\kappa_h,\nu_2)$ with $x\in S$ and $\bar{\Delta}\leq\Delta_{\max}$.*

Assume, without loss of generality, that κ_h is at least the sum of ν_2 and the Lipschitz constant of the Hessian of f, that $\kappa_g\geq\frac{1}{2}\kappa_h$ and that $\kappa_f\geq\frac{1}{3}\kappa_g$.

Then m is fully quadratic on $B(x;\Delta)$ with the same parameters $(\kappa_f,\kappa_g,\kappa_h,\nu_2)$ for all $\Delta\in\left[\bar{\Delta},\Delta_{\max}\right]$.

Here, 'without loss of generality' refers to the fact that we can always increase the values of the parameters of a fully linear or fully quadratic class, and end up with a comparable fully linear or fully quadratic class respectively for the same function and set of interest.

Assuming that fully linear and fully quadratic models can be constructed, we will in the next section review minimisation methods based on those classes. Subsequently, we will see examples of actual fully linear and fully quadratic models.

4.2 Trust-region minimisation methods

Now that we have seen the class of fully linear and fully quadratic functions as approximation to the objective function, we will describe in this section minimisation methods that iteratively minimise those approximations. It turns out that sufficient freedom is present in the method for not having the need to find a local exact minimiser of the approximation function, but only one which provide sufficient decrease compared to the some other defined point, of which we will first describe some notions.

Two different notions of 'points providing sufficient decrease' are described by Cauchy points and eigenpoints. To allow reasoning about the resulting reduction for general types of model functions, in line with [12], we require for the rest of this section the objective function to satisfy the following assumption:

Assumption 4.3. *Assume m_k is twice continuously differentiable with bounded Hessian, and thus assume there exists a $\kappa_{\text{umh}}>1$ such that for all points $x\in\mathbb{R}^n$ and iterations k, it holds*

that

$$\left\|\nabla^2 m_k(x)\right\| \leq \kappa_{\text{umh}} - 1.$$

The requirement of the model function being twice continuously differentiable alone is already stronger than what is required for model functions that are fully linear, as these only need to be once continuously differentiable. We can nevertheless justify this assumption with the observation that for the commonly-studied classes of model functions based on polynomial interpolation (to be seen in Section 4.3), the less restricted fully linear class (that is not a fully quadratic class) fulfils this assumption. For functions not fulfilling this assumption though, more specialised analysis needs to be performed. As we thus will assume twice continuously differentiability of the model function for the rest of this section, we define with $x_k \in \mathbb{R}^n$ the iteration point as short-hand notation

$$g_k \coloneqq \nabla m_k(x_k) \text{ and } H_k \coloneqq \nabla^2 m_k(x_k)$$

for the gradient and Hessian at the base point of iteration k respectively and

$$\beta_k \coloneqq 1 + \max_{s \in B(\mathbf{0};\Delta_k)} \left\|\nabla^2 m_k(x_k + s)\right\|$$

to obtain a value $\beta_k \in (0, \kappa_{\text{umh}}]$, with κ_{umh} as defined in Assumption 4.3.

With this, we can consider some specific points in the ball to base the sufficiently decrease on:

Cauchy Point

A *Cauchy point* of a function m_k in a neighbourhood $B(x_k;\Delta_k)$ for $\Delta_k > 0$ is defined as a point x_k^{C} that minimises m_k along the steepest direction from x_k in $B(x_k;\Delta_k)$, assuming x_k is not a stationary point of m_k, and thus $g_k \neq 0$. The steepest descent direction is, as seen earlier in Section 2.4, given by $-g_k$, and by scaling along this direction for the minimum with scaling factor

$$t_k^{\text{C}} \coloneqq \text{argmin}_{t \geq 0 \text{ and } x_k - t g_k \in B(x_k;\Delta_k)} m_k(x_k - t g_k), \tag{4.6}$$

the Cauchy point itself is defined as $x_k^{\text{C}} = x_k + s_k^{\text{C}}$, for $s_k^{\text{C}} \coloneqq -t_k^{\text{C}} g_k$ the Cauchy step. Now, should the function m_k be quadratic and of the form

$$m(x_k + s) = m(x_k) + g_k^T s + \frac{1}{2} s^T H_k s, \tag{4.7}$$

it follows, by [11], that an explicit formula for the Cauchy point can be given:

$$t_k^{\text{C}} = \begin{cases} \Delta_k / \|g_k\|, & g_k^T H_k g_k \leq 0 \\ \min\left\{ \|g_k\|^2 / \left(g_k^T H_k g_k\right), \Delta_k / \|g_k\| \right\}, & \text{otherwise.} \end{cases}$$

This expressions minimises along the steepest direction if the function is strictly convex and the minimiser would lay in the trust region, and otherwise places it on the boundary. If the function is not quadratic though, we cannot expect to get an explicit expression for the Cauchy point in general, and we can approximate the Cauchy point by backtracking. By defining

$$x_k(j) \coloneqq x_k - \kappa_{\text{bck}}^j \frac{\Delta_k}{\|g_k\|} g_k \tag{4.8}$$

and looking for the smallest non-negative integer $j = j_{\text{C}}$ such that

$$m_k(x_k(j)) \leq m_k(x_k) + \kappa_{\text{ubs}} g_k^T (x_k(j) - x_k) \tag{4.9}$$

for $\kappa_{\text{bck}}^j \in (0,1)$ and $\kappa_{\text{ubs}} \in \left(0, \frac{1}{2}\right)$ as application of Armijo line-search, the approximate Cauchy point can be defined as $x_k^{\text{AC}} \coloneqq x_k(j_{\text{C}})$.
The actual decreases of the model functions for the Cauchy point and the approximate Cauchy point respectively are described by the following lemmata:

Lemma 4.3 ([11])**.** *Let m_k be a model function that can be written in the form of (4.7). Then, it holds that*

$$m_k(x_k) - m\left(x_k^{\text{C}}\right) \geq \frac{1}{2} \|g_k\| \min\left\{\frac{\|g_k\|}{\beta_k}, \Delta_k\right\}. \tag{4.10}$$

Lemma 4.4 ([11])**.** *Let m_k be a model function for which Assumption 4.3 is satisfied. Then, there exists a constant $\kappa_{\text{dcp}} \in (0,1]$ independent of k such that*

$$m_k(x_k) - m_k\left(x_k^{\text{AC}}\right) \geq \kappa_{\text{dcp}} \|g_k\| \min\left\{\frac{\|g_k\|}{\beta_k}, \Delta_k\right\}. \tag{4.11}$$

Clearly, the decrease of the Cauchy point (4.10) can be described in terms of the decrease of the approximate Cauchy point (4.11), and we will base our notion of sufficiently decrease on the decrease of the approximate point.

Eigenpoint

In [11], for quadratic functions m_k, the notion of the *eigenpoint* of m_k in a neighbourhood $B(x_k; \Delta_k)$ for $\Delta_k > 0$ was defined, assuming H_k has a strictly negative eigenvalue $\tau_k < 0$.
For the eigenpoint, we seek a direction $u_k \in \mathbb{R}^n$ such that for some $\kappa_{\text{snc}} \in (0,1]$, it holds that

$$u_k^T g_k \leq 0, \|u_k\| = \Delta_k \text{ and } u_k^T H_k u_k \leq \kappa_{\text{snc}} \tau_k \Delta_k^2. \tag{4.12}$$

Choosing u_k to be a corresponding eigenvector to τ_k of H_k gives $H_k u_k = \tau_k u_k$, such that the last inequality of (4.12) reduces to $\tau_k \|u_k\|^2 \leq \kappa_{\text{snc}} \tau_k \Delta_k^2$. The other two equalities (4.12) then define the direction and scaling, for a eigenvector u_k that satisfies the requirements above.

Given a vector u_k that satisfies (4.12), we define, similar to how we defined the step size of the Cauchy point in (4.6), the step size from the current iteration point x_k to the eigenpoint as

$$t_k^{\mathrm{E}} \coloneqq \operatorname{argmin}_{t\in(0,1]} m_k\left(x_k + tu_k\right),$$

to let $x_k^{\mathrm{E}} = x_k + t_k^{\mathrm{E}} u_k$ be the eigenpoint.
Just as in case of the Cauchy point, if m_k takes the special form (4.7), we can provide an explicit expression for the eigenpoint, as it holds that $t_k^{\mathrm{E}} = 1$ in this case, according to [11]. Otherwise, for quadratic functions in general, similar to the approach used for calculating the approximate Cauchy point, a line search can be used for calculating the approximate eigenpoint x_k^{AE}, with as equivalences for (4.8) and (4.9) respectively

$$x_k\left(j\right) \coloneqq x_k + \kappa_{\mathrm{bck}}^j u_k$$

and

$$m_k\left(x_k\left(j\right)\right) \leq m_k\left(x_k\right) + \kappa_{\mathrm{ubc}}\tau_k\kappa_{\mathrm{bck}}^j \left\|u_k\right\|^2.$$

The actual decreases of the model functions for the eigenpoint and the approximate eigenpoint respectively are described by the following lemmata:

Lemma 4.5 ([11])**.** *Let m_k be a model function that can be written in the form of (4.7). Then, for τ_k a negative eigenvalue of H_k, it holds that*

$$m_k\left(x_k\right) - m_k\left(x_k^{\mathrm{E}}\right) \geq \frac{1}{2}\kappa_{\mathrm{snc}} \left|\tau_k\right| \Delta_k^2. \tag{4.13}$$

Lemma 4.6 ([11])**.** *Let m_k be a model function for which Assumption 4.3 is satisfied. Then, there exists a constant $\kappa_{\mathrm{sod}} > 0$ independent of k such that for τ_k a negative eigenvalue of H_k, it holds that*

$$m_k\left(x_k\right) - m_k\left(x_k^{\mathrm{AE}}\right) \geq \kappa_{\mathrm{sod}} \left|\tau_k\right| \min\left\{\tau_k^2, \Delta_k^2\right\}. \tag{4.14}$$

Clearly this time too, the decrease of the eigenpoint (4.13) can be described in terms of the decrease of the approximate eigenpoint (4.14), and we will base our notion of sufficiently decrease on the decrease of the approximate point.

Requiring at least a constant factor of the decrease made by a (approximate) Cauchy or eigenpoint can then be formalised as follows, respectively:

Assumption 4.4. *Assume that a constant $\kappa_{\mathrm{fcd}} \in (0,1)$ exists such that for all steps k, it holds that*

$$m_k\left(x_k\right) - m_k\left(x_{k+1}\right) \geq \kappa_{\mathrm{fcd}}\left(m_k\left(x_k\right) - m_k\left(x_k^{\mathrm{AC}}\right)\right).$$

Assumption 4.5. *Assume that a constant $\kappa_{\mathrm{fod}} \in (0,1)$ exists such that for all steps k, it*

holds that

$$m_k(x_k) - m_k(x_{k+1}) \geq \kappa_{\text{fod}} \left(m_k(x_k) - \min \left\{ m_k\left(x_k^{\text{AC}}\right), m_k\left(x_k^{\text{AE}}\right) \right\} \right).$$

Now, assuming access to a fully linear class $\mathcal{M}$ on which improvement steps can be made according to Assumption 4.4, we can apply the following first-order trust-region minimisation method:

Algorithm 4.1 (Derivative-Free Trust-Region Method (first order) ([12]))**.**

Input *Let $f \in \mathcal{C}^1$ be the function to be minimises using the fully linear class of models $\mathcal{M}$ for parameters $(\kappa_f, \kappa_g, \nu_1)$ and $x_0 \in \mathbb{R}^n$ a starting point.*
Furthermore, we require the following constants as parameters of the method:

- $0 \leq \eta_0 \leq \eta_1 < 1$ *(and $\eta_1 > 0$): defining a threshold for interpreting how well the expected decrease corresponds to the obtained decrease;*
- $0 < \gamma < 1 < \gamma_{\text{inc}}$*: controlling the size of the trust-region radius for the next iteration step;*
- $\epsilon_{\text{c}} > 0$*: threshold for the current model having enough descent as measured by $\left\|g_k^{\text{icb}}\right\|$, to decide whether or not to check the fully linearity of the model;*
- $\mu > \beta > 0$*: parameters linking the trust-region radius and the model gradient together;*
- $\omega \in (0,1)$*: parameter controlling the decrease of the model in the accuracy sub-steps.*

Initialisation *Set $k = 0$.*

Step 1: accuracy step *If $\left\|g_k^{\text{icb}}\right\| > \epsilon_{\text{c}}$, skip the certification, set $m_k = m_k^{\text{icb}}$ and $\Delta_k = \Delta_k^{\text{icb}}$ and continue with Step 2.*
Otherwise, attempt to certify the fully linearity of m^{icb} on $B\left(x_k; \Delta_k^{\text{icb}}\right)$ for parameters $(\kappa_f, \kappa_g, \nu_1)$. If m^{icb} is CFL for those parameters on $B\left(x_k; \Delta_k^{\text{icb}}\right)$ and $\Delta_k^{\text{icb}} \geq \mu \left\|g_k^{\text{icb}}\right\|$, set $m_k = m_k^{\text{icb}}$ and $\Delta_k = \Delta_k^{\text{icb}}$ and continue with Step 2.
Otherwise, improve the model:

Step 1a *Set $i = 0$ and $m_k^{(0)} = m_k^{\text{icb}}$.*

Step 1b *Increment i by one and use improve $m_k^{(i-1)}$ to create a CFL-for-parameters-$(\kappa_f, \kappa_g, \nu_1)$ model $m_k^{(i)}$ on $B\left(x_k; \omega^{i-1}\Delta_k^{\text{icb}}\right)$.*

Step 1c *If $\omega^{i-1}\Delta_k^{\text{icb}} > \mu \left\|g_k^{(i)}\right\|$, continue with Step 1b.*
Otherwise, set $m_k = m_k^{(i)}$ and $\Delta_k = \min\left\{\max\left\{\omega^{i-1}\Delta_k^{\text{icb}}, \beta\left\|g_k^{(i)}\right\|\right\}, \Delta_k^{\text{icb}}\right\}$ and continue with Step 2.

Step 2: sub-problem solution *Compute a step s_k such that a fraction of the reduction possible reduction of the Cauchy step is reduced, e.g. that Assumption 4.4 is fulfilled.*

Step 3: acceptance test *Evaluate $f(x_k + s_k)$ to define the ratio*

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}$$

to measure the successness of the step.

If $\rho_k \geq \eta_1$, declare the iteration successful*, accept the step by setting $x_{k+1} = x_k + s_k$ and increase the trust-region radius by setting $\Delta_{k+1}^{\text{icb}} \in [\Delta_k, \min\{\gamma_{\text{inc}}\Delta_k, \Delta_{\max}\}]$, increase k by one and continue with Step 1.*

Otherwise, if $\eta_1 > \rho_k \geq \eta_0$ and m_k is CFL for parameters $(\kappa_f, \kappa_g, \nu_1)$ on $B\left(x_k; \Delta_k^{\text{icb}}\right)$, declare the iteration acceptable*, accept the step by setting $x_{k+1} = x_k + s_k$ and decrease the trust-region radius by setting $\Delta_{k+1}^{\text{icb}} = \gamma\Delta_k$, increase k by one and continue with Step 1.*

Otherwise, if $\rho_k < \eta_1$ and m_k is not CFL for parameters $(\kappa_f, \kappa_g, \nu_1)$ on $B\left(x_k; \Delta_k^{\text{icb}}\right)$, declare the iteration model improving*, improve the model on $\mathcal{M}$ with parameters $(\kappa_f, \kappa_g, \nu_1)$ on $B\left(x_k; \Delta_k^{\text{icb}}\right)$, possibly with the value of $f(x_k + s_k)$ and continue with Step 1 with the same base point $x_{k+1} = x_k$ and trust-region radius $\Delta_{k+1} = \Delta_k$ and k thereafter set to $k + 1$.*

Otherwise, if $\rho_k < \eta_0$ and m_k and m_k is CFL for parameters $(\kappa_f, \kappa_g, \nu_1)$ on $B\left(x_k; \Delta_k^{\text{icb}}\right)$, declare the iteration unsuccessful*, continue with Step 1 with the same base point $x_{k+1} = x_k$ but a decreased trust-region radius $\Delta_{k+1} = \gamma\Delta_k$ and k thereafter set to $k + 1$.*

At first glance, it might look confusing that the 'improvement' algorithm for the fully linear class is called in this algorithm at two different locations, with in Step 1 a loop that is executed until the model is certifiable fully linear, and in Step 3 just a single call to the improvement algorithm. And indeed, a similar algorithm as presented in [1] only contains the first step, that is unconditionally called and after which the method does not need to be CFL. The way the algorithm is presented above tries to make more efficient use of the evaluated points. The improving in the accuracy step makes sure not both the magnitude of the model's gradient, g_k^{icb}, and the trust-region radius can become uncontrollable small, such that the fraction of the Cauchy decreases becomes so too, and the algorithm could converge to a point that is not a first-order critical point of f. By improving the model gradually in the last step, with only one step at the time, we can skip the more expensive improvement during the accuracy step, as long as the model's gradient is steep enough and the decrease agrees enough with that of the objective function, even if the model is not CFL.

For a convergence results of this algorithm, we assume that a minimiser exists that we can reach:

Assumption 4.6. *Assume that f is uniformly bounded from below on $L(x_0)$.*

Then, the convergence result for the first-order method is as follows:

Theorem 4.7 ([12])**.** *Suppose Assumption 4.6 holds and let $(x_k)_{k\in\mathbb{Z}}$ be a sequence of base points resulting from applying Algorithm 4.1 on f with a fully linear class of models.*[3] *Then:*

$$\lim_{k\to+\infty} \nabla f(x_k) = 0.$$

For second-order convergence, a method is stated in [12, Alg. 10.3] that is similar to Algorithm 4.1, where the quality of the decrease of the sub-problem is this time compared to that of the eigenpoint, e.g. to make Assumption 4.5 fulfilled. Furthermore, as another major difference, while the gradient of the model was a measure for the first-order method for having reached a stationary point, a measure is in this case

$$\sigma(x) = \max\{\|g_k\|, |\lambda_{\min}(H_k)|\},$$

with the second element denoting the magnitude of the most negative eigenvalue of H_k, which corresponds to conditions in the definition of a second-order critical point. For this second-order method, can state the stronger convergence result as follows:

Theorem 4.8 ([12])**.** *Suppose Assumption 4.6 holds and let $(x_k)_{k\in\mathbb{Z}}$ be a sequence of base points resulting from applying second-order trust-region method in [12, Alg. 10.3] on f with a fully quadratic class of models.*[4] *Then:*

$$\lim_{k\to+\infty} \sigma(x_k) = 0.$$

4.3 Examples of fully linear and fully quadratic models

In this section, we will consider several types of fully linear and fully quadratic models. An approach commonly found in the literature is to create a model function whose value at sufficient sample points around the base point equals the value of the objective function on those points, leading to a moderate error for points outside the sample sets that are sufficiently surrounded by other sample points. Thus, we will impose geometric requirements on the sample set for such a set to serve as basis of a fully linear or fully quadratic model.

In the first sub-section, we will review the basics of interpolation, being the creation of the mentioned model function equalling the objective function at specific points. In the later sub-sections, we will consider interpolation for polynomial model functions, being of linear or quadratic form with different amounts of interpolation points, with the corresponding geometric requirements.

[3] Note that, in addition to Assumption 4.6, Assumption 4.1 is assumed to be satisfied by f as part of the definition of a fully linear class of models

[4] See Footnote 3, with 'fully linear' replaced by 'fully quadratic'.

4.3.1 Interpolation conditions

Given a set of sample points Y of size $p_1 \coloneqq p+1$, we want to construct an *interpolation* model $m\colon \mathbb{R}^n \to \mathbb{R}$ of the objective function f by seeking a function such that $m\left(y^i\right) = f\left(y^i\right)$ holds for all $y^i \in Y$: what is left, is polishing (*poliō*) of the model in-between (*inter*) the sample points.

We consider interpolation functions m that are written as the sum of $q_1 \coloneqq q+1$ different functions ϕ_j of a certain class (e.g. polynomials), weighted by fixed coefficients $\alpha_j \in \mathbb{R}$, as $m(x) = \sum_{j=0}^{q} \alpha_j \phi_j(x)$.

Now, by enforcing the interpolation condition $m\left(y^i\right) = f\left(y^i\right)$ for all $y^i \in Y$, we get

$$\sum_{j=0}^{q} a_j \phi_j\left(y^i\right) = f\left(y^i\right), \qquad i = 0, \ldots, p$$

or, in matrix form,

$$M(\phi, Y)\,\alpha = f(Y) \tag{4.15}$$

with

$$M(\phi, Y) = \begin{bmatrix} \phi_0\left(y^0\right) & \phi_1\left(y^0\right) & \cdots & \phi_q\left(y^0\right) \\ \phi_0\left(y^1\right) & \phi_1\left(y^1\right) & \cdots & \phi_q\left(y^1\right) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0\left(y^p\right) & \phi_1\left(y^p\right) & \cdots & \phi_q\left(y^p\right) \end{bmatrix}, \alpha = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_q \end{bmatrix}, \text{ and } f(Y) = \begin{bmatrix} f\left(y^0\right) \\ f\left(y^1\right) \\ \vdots \\ f\left(y^p\right) \end{bmatrix}.$$

Now that the notation has been introduced, we will consider what happens for different classes, and different number of points compared to the number of functions. We will consider what happens when we require m to be a polynomial function in $\mathbb{R}^n$ of a degree less than or equal to $d \in \mathbb{N}$, e.g. $m \in \mathcal{P}_n^d$, with a finite basis $\phi = \{\phi_1, \phi_2, \ldots, \phi_q\}$ of $\mathcal{P}_n^d$ as set of functions to write m construct m with. Depending on the relation between q_1 and p_1, the interpolation condition (4.15) might be overdetermined ($q_1 < p_1$) or underdetermined ($q_1 > p_1$). We will start with the ideal case where the system is not under- or overdetermined though, for $q_1 = p_1$.

4.3.2 Polynomial interpolation

In this sub-section, the case of the dimension of $\mathcal{P}_n^d$, and thus the number of basis functions of $\mathcal{P}_n^d$, q_1, being equal to the number of interpolation points, p_1, will be considered: we call this case simply polynomial interpolation. In that case, for any basis ϕ of $\mathcal{P}_n^d$, the matrix $M(\phi, Y)$ is a square $p_1 \times p_1$ matrix, and the matrix system (4.15) has a unique solution in α if and only if $M(\phi, Y)$ is non-singular. By considering any other basis ψ of $\mathcal{P}_n^d$, the two baseis can be related to each other by $\psi(x) = P^T \phi(x)$ for an invertible matrix P and any $x \in \mathbb{R}^n$, and we get $\psi\left(y^i\right)^T = \phi\left(y^i\right)^T P$ for all $i = 0, \ldots, p$ and thus $M(\psi, Y) = M(\phi, Y)\,P$. The matrix system has therefore a unique solution in α if and only if $M(\psi, Y)$ is non-singular.

It follows that whether or not we are able to exactly interpolate a function through a set of sample points is independent of the specific basis chosen to determine the interpolation coefficients, and a definition for a set being suitable to be used as set of interpolation points can be stated in the following way:

Definition 4.3 ([12])**.** The set $Y = \left\{y^0, y^1, \ldots, y^p\right\}$ is *poised for polynomial interpolation* in $\mathbb{R}^n$ if the corresponding matrix $M(\phi, Y)$ is non-singular for some basis ϕ in $\mathcal{P}_n^d$.

It turns out that not only the poisedness of an interpolation set is independent of the basis chosen, but even the interpolation polynomial itself is:

Theorem 4.9 (verbatim from [12])**.** *Given a function $f\colon \mathbb{R}^n \to \mathbb{R}$ and a poised set $Y \in \mathbb{R}^n$, the interpolation polynomial $m(x)$ through f at Y exists and is unique.*

We will now discuss some baseis of $\mathcal{P}_n^d$ that can be used for interpolation, with the number of elements in such basis is given by $\binom{d+n}{n}$. An intuition for this expression for the dimension can be found by enumerating all possibilities of dividing the degree d over the available linearly independent monomials in $\mathcal{P}_n^d$ up to degree 1, say $1, x_1, x_2, \ldots, x_n$, to be used to built up a basis element. Using the argument of stars and bars, we can write d stars and have to add n bars to divide the stars over the monomials, making us seek the placement of the n bars over $d+n$ options; there are $\binom{d+n}{n}$ possibilities for such arrangement. For the commonly encountered case of linear interpolation, with $d = 1$, the dimension of the space of model functions is then given by $p_1 = n + 1$, while for quadratic interpolation, with $d = 2$, by $p_1 = (n+1)(n+2)/2$. With an expression for the number of basis elements in $\mathcal{P}_n^d$, we consider two actual baseis of that space:

Natural basis $\bar{\phi}$

The natural basis is a basis that also appears in a (natural) multi-variate Taylor expansion for approximating functions using polynomials. Applying a Taylor expansion up to and including order d on a function $g\colon \mathbb{R}^n \to \mathbb{R}$ (with x as input) around y to z results in, according to [35, Thm. A.7],

$$g(y+z) = g(y) + \sum_{r=1}^{d} \frac{U_r(z)}{r!} + E_d$$

with

$$U_r(y) = \left[\left(z_1 \frac{\partial}{\partial x_1} + \cdots + z_n \frac{\partial}{\partial x_n}\right)^r g\right](y)$$

and E_d being a rest term, whose exact value is of no interest for defining the natural basis. We can then write $f(y+z) - E_d$ as polynomial in z. Extracting the different terms and ignoring the coefficients, we end up with $\binom{d+n}{n}$ linearly independent monomials in z, which all together form the so-called natural basis of $\mathcal{P}_n^d$:

Definition 4.4 ([12])**.** Let $\alpha^i = \left(\alpha_1^i, \alpha_2^i, \ldots, \alpha_n^i\right)$ be a tuple of numbers and define $\alpha = \left\{\alpha^0, \alpha^1, \ldots, \alpha^p\right)$ as a set of p_1 distinct tuples α^i such that $\sum_{j=1}^n \alpha_j^i \leq d$.
We can define the ith *natural basis* element of $\mathcal{P}_n^d$ as

$$\bar{\phi}_i(x) = \frac{1}{\prod_{j=1}^n a_j^i!} \prod_{j=1}^n x_j^{a_j^i}.$$

The terms in this basis together look like the following:

$$\bar{\phi} = \left\{1, x_1, \ldots, x_n, x_1^2/2, x_1 x_2, \ldots, x_{n-1}^{d-1} x_n/(d-1)!, x_n^d/d!\right\}.$$

Lagrange polynomials ℓ

While the natural basis was chosen independent of the interpolation points, a basis consisting of Lagrange polynomials depends actually on those points, by choosing the coefficients of the polynomials in such a way that there is a unique polynomial for each point having value 1 at that point, and 0 at all the others:

Definition 4.5 ([12])**.** Given a set of interpolation points $Y = \left\{y^0, y^1, \ldots, y^p\right\}$, a set of polynomial functions $\{\ell_0, \ell_1, \ldots, \ell_p\} \subset \mathcal{P}_n^d$ (for p_1 the dimension of $\mathcal{P}_n^d$) is called a set of *Lagrange polynomials* if

$$\ell_j\left(y^i\right) = \delta_{ij}.$$

It is shown in [12] that the set of Lagrange polynomials for a poised set of interpolation polynomials forms a basis of $\mathcal{P}_n^d$. Thus, the unique interpolation polynomial can be clearly written as

$$m(x) = \sum_{i=0}^{p} f\left(y^i\right) \ell_i(x) \tag{4.16}$$

as for each y^i as argument to m, all terms except for $f\left(y^i\right)$ vanish, and it thus holds that $m\left(y^i\right) = f\left(y^i\right)$.

At the beginning of this sub-section, we have seen the condition for an interpolation set Y to be poised for polynomial interpolation to be that the matrix $M(\phi, Y)$ is non-singular for some basis ϕ. We are now interested in whether we can define something that expresses the well-poisedness of an interpolation set, where a set that is better poised than another set might yield a lower error between the model and the original function on those points that are not in the interpolation set, in some bounded area.

Since it is enough for an interpolation set Y to be poised that the matrix $M(\phi, Y)$ is non-singular for some basis ϕ, a natural question is whether a measure of the singularity of $M(\phi, Y)$ is a measure of well-poisedness too. A traditional measure of non-singularity of a matrix A is the condition number $\kappa(A) = \|A\| \left\|A^{-1}\right\|$, which, according to [15], up to a factor of the norm of A, is reverse-proportional to the distance to any singular matrix. According

to [12] though, for any interpolation set Y and scalar $\bar{\kappa} > 0$, there is a basis ϕ of $\mathcal{P}_n^d$ such that $k\left(M\left(\phi, Y\right)\right) = \bar{\kappa}$, and the condition number depends on the scaling of Y, and thus, in general, the condition number for a basis and scaling in general doesn't provide insight in the well-poisedness. In the special case of the natural basis $\bar{\phi}$ and scaled-down version of Y though, the condition number does, as we are now going to see.

We define a specific scaled version of $Y = \left\{y^0, y^1, \ldots, y^p\right\}$ as $\hat{Y}$, being the set centred relative for y^0 at the origin with radius 1. We write, for $\Delta = \Delta\left(Y\right) \coloneqq \max_{1 \leq j \leq p} \left\|y^i - y^0\right\|$,

$$\hat{Y} = \left\{\hat{y}^0 \coloneqq 0, \hat{y}^1 \coloneqq \left(y^1 - y^0\right)/\Delta, \ldots, \hat{y}^p \coloneqq \left(y^p - y^0\right)/\Delta\right\}. \tag{4.17}$$

We can then write in the linear and quadratic case $\hat{M} = M\left(\bar{\phi}, \hat{Y}\right)$ respectively as $M\left(\bar{\phi}, \hat{Y}\right) = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{1} & \hat{L} \end{bmatrix}$ and $M\left(\bar{\phi}, \hat{Y}\right) = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{1} & \hat{Q} \end{bmatrix}$, with, for $d = 1$,

$$\hat{L} = \begin{bmatrix} \hat{y}_1^1 & \hat{y}_2^1 & \cdots & \hat{y}_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_1^n & \hat{y}_2^n & \cdots & \hat{y}_n^n \end{bmatrix}, \tag{4.18}$$

and, for $d = 2$ with $p = p_1 - 1 = (n+1)(n+2)/2 - 1$,

$$\hat{Q} = \begin{bmatrix} \hat{y}_1^1 & \hat{y}_2^1 & \cdots & \hat{y}_n^1 & \frac{1}{2}\left(\hat{y}_1^1\right)^2 & \hat{y}_1^1\hat{y}_2^1 & \cdots & \frac{1}{2}\left(\hat{y}_n^1\right)^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ldots & \vdots \\ \hat{y}_1^p & \hat{y}_2^p & \cdots & \hat{y}_n^p & \frac{1}{2}\left(\hat{y}_1^p\right)^2 & \hat{y}_1^p\hat{y}_2^p & \cdots & \frac{1}{2}\left(\hat{y}_n^p\right)^2 \end{bmatrix}. \tag{4.19}$$

Note that both $\hat{L}$ and $\hat{Q}$ are, in case of interpolation, square matrices that are non-singular if and only if $\hat{Y}$ is poised (in the linear or quadratic interpolation sense respectively), which is clearly poised if and only if Y is poised in the same sense.

Now, for a quadratic interpolation model, we will state a theorem describing the error between the model function and the objective function in the ball around y^0 with radius Δ, which was defined as the maximum distance between y^0 and the other points. The model function is in that case twice continuously differentiable with non-trivial Hessian, and we require the objective function to have related properties to be able to capture the contours of the objective function well:

Assumption 4.7 (notation inspired by [12])**.** *Assume $Y = \left\{y^0, y^1, \ldots, y^n\right\} \subset \mathbb{R}^n$ is a poised set of sample points in the quadratic interpolation sense in the ball $B = B\left(y^0; \Delta\left(Y\right)\right)$ of radius $\Delta = \Delta\left(Y\right)$.*

Assume f is twice continuously differentiable in an open domain Ω containing B and $\nabla^2 f$ is Lipschitz continuous in Ω.

The actual theorem is then as follows:

Theorem 4.10 ([12]). *Assume function f, interpolation set Y and radius $\Delta = \Delta(Y)$ fulfil Assumption 4.7 with the Lipschitz constant of ∇f^2 being denoted by γ_f and that m is the quadratic interpolation polynomial of f on Y. Then, a bound on the error between the function f and the model m and their derivatives in $B = B\left(y^0; \Delta(Y)\right)$ is given by*

$$\left\|\nabla^2 f(x+s) - \nabla^2 m(x+s)\right\| \leq \frac{3}{2}\sqrt{2p}\gamma_f \left\|\hat{Q}^{-1}\right\| \Delta,$$
$$\left\|\nabla f(x+s) - \nabla m(x+s)\right\| \leq \frac{3}{2}\left(1+\sqrt{2}\right)\sqrt{p}\gamma_f \left\|\hat{Q}^{-1}\right\| \Delta^2$$

and

$$\left|f(x+s) - m(x+s)\right| \leq \left(\frac{1}{4}\left(6+9\sqrt{2}\right)\sqrt{p}\gamma_f \left\|\hat{Q}^{-1}\right\| + \frac{1}{6}\gamma_f\right)\Delta^3$$

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

With a bound for the quadratic case being stated here, a similar error bound for the linear case can be found in Sub-section 4.3.4, where Theorem 4.16 provides as special case bounds for the error for interpolation based on a set of $p_1 = n+1$ points that are poised in interpolation sense.

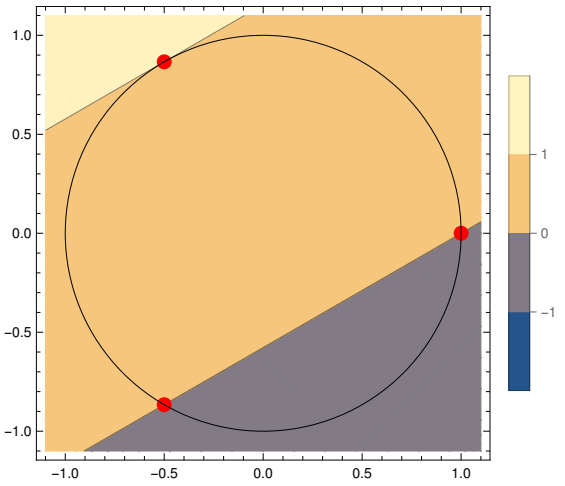
Tempting as it may seem, those two theorems on the error are not enough to state that interpolation based on a linear or quadratic polynomial yields respectively a fully linear of fully quadratic model for some constants, as exactly at the places in the inequalities describing the error where otherwise in the definition of a fully linear of quadratic model the constant would appear, a dependency on a set Y that can change is present, making that factor not constant under different interpolation sets. Instead, we seek an upper bound of $\left\|\hat{M}^{-1}\right\|$, which in itself is already an upper bound for $\left\|\hat{L}^{-1}\right\|$ and $\left\|\hat{Q}^{-1}\right\|$, possibly by restricting ourselves to interpolation sets that would fulfil such a bound. As we desire to easily enforce such bounds (by, for example, changing some points in the interpolation set) in a finite and uniformly bounded (with respect to the trust region) number of steps for a fully linear or fully quadratic class, we need to seek a reachable upper bound, and one such is given by the idea of Λ-poisedness:

Definition 4.6 ([12]). Given a positive constant Λ and a set of interest $B \in \mathbb{R}^n$, a poised set $Y = \left\{y^0, y^1, \ldots, y^p\right\}$ of interpolation points is said to be Λ-*poised (in the interpolation sense)* if

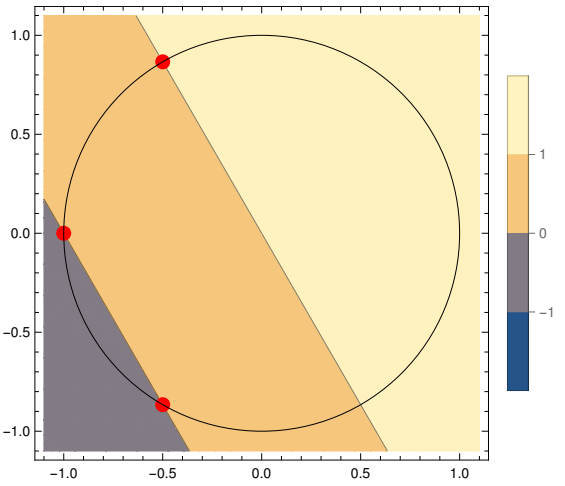
$$\Lambda \geq \max_{0 \leq i \leq p} \max_{x \in B} \left|\ell_i(x)\right|$$

for the (unique) basis of Lagrange polynomials $\{\ell_1, \ell_2, \ldots, \ell_p\}$ corresponding to Y.

We could imagine Λ-poisedness as a measure of how well Y spans B. If all interpolation points in Y are present in B, then for each interpolation point, there is a Lagrange polynomial which evaluated at that point yields the value 1, while yielding the value 0 at all the other points.



(a) Contour plot of one of the Lagrange polynomials of an interpolation set consisting of three points on $B(\mathbf{0}; 1)$ with equal distance. This set is clearly Λ-poised for $\Lambda = 1$ on $B(\mathbf{0}; 1)$: the values of the Lagrange polynomials never exceed 1 in the set.



(b) Contour plot of the Lagrange polynomials corresponding to the top point of the same interpolation set as in the previous, but with $(1, 0)$ swapped over the vertical axis, to $(-1, 0)$. This set is Λ-poised for $\Lambda = 1 + \frac{1}{2\sqrt{3}} + \frac{\sqrt{3}}{2} \approx 2.1547$ on $B(\mathbf{0}; 1)$.

Figure 4.1: Example of two interpolation sets poised for linear interpolation in $\mathbb{R}^2$, with one being a slightly modified version of another, with different Λ-poisedness as consequence. It can easily be seen that all interpolation sets in $\mathbb{R}^2$ that are Λ-poised with $\Lambda = 1$ are of the form of the first figure, where for every interpolation point the two other interpolation points must lay parallel on the tangent of the other point, while that point must lay on the boundary.

The function values at other points are determined by the moderate Lagrange functions, being a polynomial of some lower degree, with thus a restricted degrees of freedom. Now, the greater Λ, the greater the deviation from values between 0 and 1 for a point in B, which because of the restricted freedom likely means that some part of B is less covered by the interpolation set, Y. Intuitively, it thus might be desirable to work with an interpolation set that is Λ-poised for a small value of Λ, and indeed, as we will see, the value of Λ is directly correlated to the maximum error of an interpolation polynomial in a restricted area, under certain assumptions.

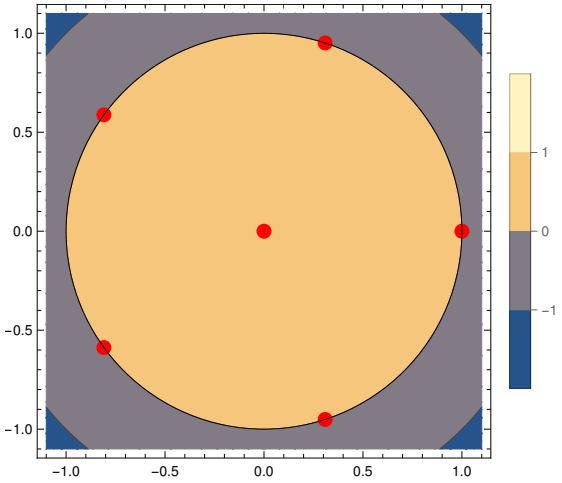
Before reviewing some examples, we present several properties of Λ-poisedness, that easily follow from the definition:

Lemma 4.11 (verbatim from [12]). *Let $B, Y \subset \mathbb{R}^n$.*

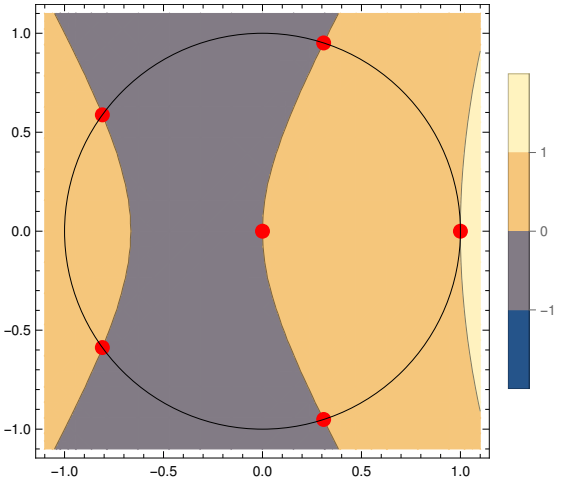
(i) *If B contains a point in Y and Y is Λ-poised in B, then $\Lambda \geq 1$.*

(ii) *If Y is Λ-poised in a given set B, then it is Λ-poised (with the same constant) in any subset of B.*

(iii) *If Y is Λ-poised in B for a given constant Λ, then it is $\bar{\Lambda}$-poised in B for any $\bar{\Lambda} > \Lambda$.*

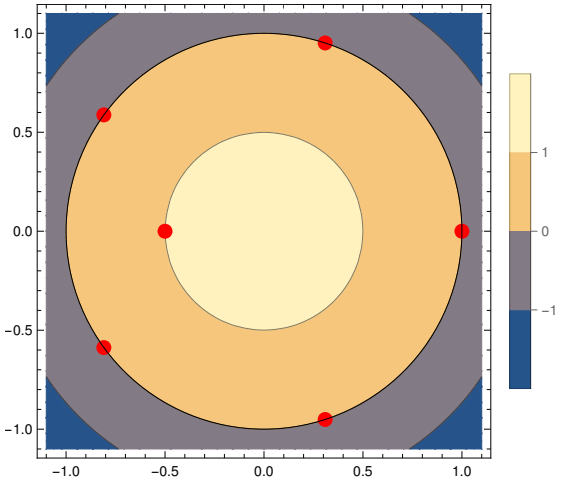


(a I) Contour plot of the Lagrange polynomial corresponding to $(0, 0)$.

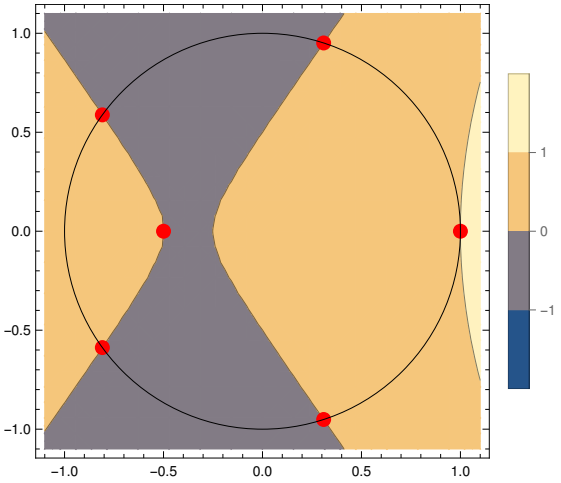


(a II) Contour plot of the Lagrange polynomial corresponding to $(1, 0)$.

(a) The original interpolation set on the original ball. The interpolation set is clearly Λ-poised for $\Lambda = 1$: none of the Lagrange polynomials take a value greater than 1 on this ball.



(b I) Contour plot of the Lagrange polynomial corresponding to $\left(-\frac{1}{2}, 0\right)$.



(b II) Contour plot of the Lagrange polynomial corresponding to $(1, 0)$.

(b) The interpolation set with $(0, 0)$ replaced by $\left(-\frac{1}{2}, 0\right)$ on the original ball. This interpolation set Λ-poised for $\Lambda = \frac{4}{3} > 1$: for the Lagrange polynomial corresponding to the new internal point, the only degree left by the other points is contracting, and thus, the maximum value is attained in the centre, whose value must be greater than 1 to allow any non-centre point to take 1 as value.

Figure 4.2: Example of two interpolation sets poised for quadratic interpolation in $\mathbb{R}^2$, with one being a slightly modified version of another, with different Λ-poisedness as consequence. The base set of interest B in this example is the ball with radius 1 centred around the origin, $B(\mathbf{0}; 1)$, with the interpolation set Y consisting of the origin and five points on the ball with equal distance between them, with $(1, 0)$ included.

In Figures 4.1 and 4.2, examples of the poisedness of an interpolation set $Y \subset \mathbb{R}^2$ fully inside the set of interest $B = B(\mathbf{0}; 1)$ are shown for both linear and quadratic interpolation, for which, as shown earlier, respectively $2 + 1 = 3$ and $(2 + 1)(2 + 2)/2 = 6$ interpolation points are required. As a consequence of the previous lemma, the smallest possible value of Λ for those examples, while keeping Y fully contained in B, for the Λ-poised sets is $\Lambda = 1$. It can easily be seen that extending the set of interest B while keeping the same interpolation sets will increase the value of Λ; points for which the Lagrange functions yield higher functions are now included in B, increasing the maximum of the function values of the Lagrange functions in B, and thus Λ.

A relation between $\left\|\hat{M}^{-1}\right\|$ and Λ-poisedness is then established by means of the following theorem:

Theorem 4.12 (verbatim from [12])**.** *If $\hat{M}$ is non-singular and $\left\|\hat{M}^{-1}\right\| \leq \Lambda$, then the set $\hat{Y}$ is $\sqrt{p_1}\Lambda$-poised in $B(\mathbf{0}; 1)$.*

Conversely, if the set $\hat{Y}$ is Λ-poised in $B(\mathbf{0}; 1)$, then $\left\|\hat{M}^{-1}\right\| \leq \theta\sqrt{p_1}\Lambda$ for θ a constant depending on n and d, but independent of $\hat{Y}$ and Λ.

Thus, if we only consider interpolation sets whose scaled and shifted variant is Λ-poised sets, then the factors on the right-hand sides of the inequalities found in Theorems 4.16 and 4.10 are bounded by a constant depending on Λ, and we can use linear and quadratic polynomial interpolation to establish fully linear and fully quadratic models. Luckily, according to some lemmata in [12], Y being Λ-poised in B is equivalent to $\hat{Y}$ being Λ-poised in $\frac{1}{\Delta}B \subset B(\mathbf{0}; 1)$, and thus we don't need per se to transform Y into the scaled and shifted one to consider the Λ-poisedness of it, and can modify Y directly.

What is left to show for using polynomial interpolation as fully linear or fully quadratic class is the existence of an improvement algorithm that creates a fully linear or fully quadratic class respectively in a finite and uniformly bounded (with respect to the trust region) number of steps. We will review such an algorithm in Section 4.4.

4.3.3 Polynomial regression

In the previous sub-section, we have seen how a fully linear and fully quadratic models could be created. This was done by using linear and quadratic polynomials respectively in a q_1-dimensional space and interpolating them on p_1 points with the original objective function, with $p_1 = q_1$. In this sub-section, we consider the case where $p_1 > q_1$, e.g. the case where we have more points than the dimension of the polynomial space, and the interpolation conditions cannot be fulfilled for every point, as we have to few degrees of freedom for doing so.

In this case, (4.15) can in general not be attained for some coefficient vector $\alpha \in \mathbb{R}^{q_1}$, and

we seek instead a solution of the problem

$$M(\phi, Y)\,\alpha \overset{\text{l.s.}}{=} f(Y) \tag{4.20}$$

for a basis ϕ of $\mathcal{P}_n^d$, meaning that we want a solution fulfilling

$$\min_{\alpha \in \mathbb{R}^{q_1}} \|M(\phi, Y)\,\alpha - f(Y)\|. \tag{4.21}$$

The condition (4.21) is called a least-squares regression condition, as we are minimising the difference between the left-hand side and right-hand side of the equation in the ℓ^2 norm.

As this least-squares regression condition has a unique solution if $M(\phi, Y)$ has full column rank, and $M(\phi, Y)$ having full column rank is independent of the choice of basis ϕ, according to [12], a notion for a sample set being poised for polynomial least-squares regression can be defined in the following way:

Definition 4.7 (verbatim from [12])**.** The set $Y = \{y^0, y^1, \ldots, y^p\}$ is *poised for polynomial least-squares regression* in $\mathbb{R}^n$ if the corresponding matrix $M(\phi, Y)$ has full column rank for some basis ϕ in $\mathcal{P}_n^d$.

Note the similarity with the interpolation case: in fact, in the case of $p_1 = q_1$, the matrix $M(\phi, Y)$ having full column rank is equivalent to this matrix being invertible, and being poised in the interpolation sense can be seen as a special case of being poised in the regression sense. As in the interpolation sense, the least-squares regression polynomial induced by a poised set exists and is unique:

Theorem 4.13 (verbatim from [12])**.** *Given a function $f\colon \mathbb{R}^n \to \mathbb{R}$ and a poised set $Y \in \mathbb{R}^n$, the least-squares regression polynomial $m(x)$ exists and is unique.*

In the case of polynomial interpolation, we were able to construct an interpolation polynomial of degree d by creating Lagrange polynomials of the same degree d whose value at the corresponding interpolation point is 1 and 0 at all other interpolation points. Clearly, as the number of points is now bigger than the degree, we cannot construct such exact Lagrange polynomials anymore, and can only approximate them:

Definition 4.8 (taken from [12])**.** Given a set of sample points $Y = \{y^0, y^1, \ldots, y^p\}$, with $p > q$, a set of $p_1 = p + 1$ polynomial functions $\{\ell_0, \ell_1, \ldots, \ell_p\} \subset \mathcal{P}_n^d$ (for p_1 the dimension of $\mathcal{P}_n^d$) is called a set of *regression Lagrange polynomials* if

$$\ell_j(y^i) \overset{\text{l.s.}}{=} \delta_{ij}.$$

This regression condition here is equivalent to stating that for each regression Lagrange polynomial corresponding to point y^j, a vector α_ϕ^j is desired as solution of $M(\phi, Y)\,\alpha_\phi^j \overset{\text{l.s.}}{=} \mathbf{e}_{j+1}$ for each $j = 0, \ldots, p$. According to [12], the least-squares regression polynomials exist and

are uniquely defined for a poised set, and the unique least-squares regression polynomial m can be written as

$$m(x) = \sum_{i=0}^{p} f\left(y^i\right) \ell_i(x),$$

just as in the case of polynomial interpolation.

Just as in the case of polynomial interpolation as well, we are interested in the error between the model function and the objective function. A known error bound for all x in the convex hull formed by Y, rewritten by [12], is

$$|f(x) - m(x)| \leq \frac{1}{(d+1)!} \nu_d p_1 \Lambda_\ell \Lambda^{d+1} \tag{4.22}$$

with Λ being the diameter of the smallest ball $B(Y)$ containing Y, ν_d an upper bound on the $d+1$th derivative of f and

$$\lambda_\ell = \max_{0 \leq i \leq p} \max_{x \in B(Y)} |\ell_i(x)|.$$

By bounding the right-hand side of (4.22), the error could be bounded, just as ways to bound the error in the case of polynomial interpolation were provided using Λ-poisedness. However, naively bounding each factor on this right-hand side would require the number of sample points, p_1, to be bounded from above, ruling the possibility of taking as many points as possible into account out, which would have been desirable for getting as much information as possible.

To overcome this problem, the concept of *strong* Λ-poisedness (in the regression sense) will be introduced. Motivated by a derivation in [12], we say that a set Y is strongly Λ-poised if it can be partitioned into $l = \lfloor p_1/q_1 \rfloor$ subsets of q_1 points that are each Λ-poised (in the interpolation sense) and possibly another subset of less than q_1 points. Formally, the definition of strong Λ-poisedness (in the sense of linear regression) is the following:

Definition 4.9 ([12])**.** Given a positive constant Λ and a set of interest $B \in \mathbb{R}^n$, a poised set $Y = \left\{y^0, y^1, \ldots, y^p\right\}$ of sample points is said to be *strongly Λ-poised (in the regression sense)* if[5]

$$\frac{q_1}{\sqrt{p_1}} \Lambda \geq \max_{x \in B} \|\ell(x)\| \tag{4.23}$$

for the (unique) set of Lagrange regression polynomials $\{\ell_0, \ell_1, \ldots, \ell_p\}$ corresponding to Y, with $\ell = \begin{bmatrix} \ell_0 & \ell_1 & \cdots & \ell_p \end{bmatrix}^T$.

We will in practice in the improvement algorithms mainly work with the motivation of this definition, rather than trying to directly modify (4.23).

[5]Note that this time the l^2 norm is used, as opposed to the l^∞ norm that is used in the definition of (regular, non-strong) Λ-poisedness.

With this, we will state a theorem about the error between the model and the objective, similarly to what was done in the previous section. In the overdetermined case this time, we will consider though only the situation of an overdetermined quadratic model. The case of an overdetermined linear model that is not an (overdetermined) quadratic model can then be treated as underdetermined quadratic model, which will briefly be brought up in the next sub-section.

Just as in the interpolation case, we connect the concept of poisedness with the matrix $\hat{M}$. By seeking the reduced singular value decomposition as $\hat{M} = \hat{U}\hat{\Sigma}\hat{V}^T$, for $\hat{U}$ a matrix with orthonormal columns and $\hat{V}$ an orthonormal matrix, we can write error bounds in terms of $\left\|\hat{\Sigma}^{-1}\right\|$. We first write our general assumptions for the theorem:

Assumption 4.8 ([12])**.** *Assume $Y = \left\{y^0, y^1, \ldots, y^p\right\} \subset \mathbb{R}^n$, with $p_1 = p + 1 > (n+1)(n+2)/2$, is a poised set of sample points (in the quadratic regression sense) in the ball $B = B\left(y^0; \Delta(Y)\right)$ of radius $\Delta = \Delta(Y)$.*

Assume f is twice continuously differentiable in an open domain Ω containing B and $\nabla^2 f$ is Lipschitz continuous in Ω.

Then, the error bounds are given by the following theorem:

Theorem 4.14. *Assume function f, sample set Y and radius $\Delta = \Delta(Y)$ fulfil Assumption 4.8 with the Lipschitz constant of ∇f^2 being denoted by γ_f and that m is the quadratic least-squares regression polynomial of f on Y. Then, a bound on the error between the function f and the model m and their derivatives in $B = B\left(y^0; \Delta(Y)\right)$ is given by*

$$\begin{aligned}\left\|\nabla^2 f(x+s) - \nabla^2 m(x+s)\right\| &\leq \left(\gamma_f + \sqrt{2}\bar{p}^{\frac{1}{2}}/2 \cdot \gamma_f \left\|\hat{\Sigma}^{-1}\right\|\right)\Delta, \\ \left\|\nabla f(x+s) - \nabla m(x+s)\right\| &\leq \left(\gamma_f + \left(n^{\frac{1}{2}} + \sqrt{2}\bar{p}^{\frac{1}{2}}\right)/2 \cdot \gamma_f \left\|\hat{\Sigma}^{-1}\right\|\right)\Delta^2\end{aligned}$$

and

$$\left|f(x+s) - m(x+s)\right| \leq \left(\frac{1}{2}\gamma_f + \left(\frac{1}{2} + \frac{1}{2}n^{\frac{1}{2}} + \frac{1}{4}\sqrt{2}\bar{p}^{\frac{1}{2}}\right)\gamma_f \left\|\hat{\Sigma}^{-1}\right\|\right)\Delta^3$$

for all $s \in B(\mathbf{0}; \Delta)$, with $\bar{p} = n(n+1)/2$.

Lastly, the following theorem provides the direct connection between strong Λ-poisedness and $\hat{\Sigma}^{-1}$, with the equivalent in the previous sub-section being Theorem 4.12:

Theorem 4.15 (verbatim from [12])**.** *If $\hat{\Sigma}$ is non-singular and $\left\|\hat{\Sigma}^{-1}\right\| \leq \sqrt{q_1/p_1}\Lambda$, then the set $\hat{Y}$ is strongly Λ-poised in $B(\mathbf{0}; 1)$.*

Conversely, if the set $\hat{Y}$ is strongly Λ-poised in $B(\mathbf{0}; 1)$, then $\left\|\hat{\Sigma}^{-1}\right\| \leq \theta\frac{q_1}{\sqrt{p_1}}\Lambda$ for θ a constant depending on n and d, but independent of $\hat{Y}$ and Λ.

4.3.4 Underdetermined models

In the previous two sub-sections, a quadratic interpolation model and regression model have been described, for the case of $p_1 = q_1$ and $p_1 > q_1$ respectively. In this sub-section, we briefly introduce the last case, of $p_1 < q_1$, when the model is underdetermined. In this case, less interpolation points are available than the dimension of the polynomial space, and, in general, multiple model functions would be able to fulfil the interpolation conditions. The obvious advantage for using underdetermined models is that less points need to be sampled for constructing a model than otherwise: the number of points for a full quadratic model is of order $\mathcal{O}\left(n^2\right)$, which is a lot if perhaps an underdetermined model using $\mathcal{O}\left(n\right)$ points could suffice.

We start by stating a general theorem about the error of a regression model with at least $n+1$ sample points – which for exactly this amount of points corresponds with the dimension of $\mathcal{P}_n^1$, the space of linear polynomials in $\mathbb{R}$ – to obtain error bounds that could help defining a fully linear model. First, as usual, we state our assumptions:

Assumption 4.9 (notation inspired by [12])**.** *Assume $Y = \left\{y^0, y^1, \ldots, y^n\right\} \subset \mathbb{R}^n$, with $p_1 = p+1 \geq n+1$, is a poised set of sample points (in the linear regression sense) in the ball $B = B\left(y^0; \Delta\left(Y\right)\right)$ of radius $\Delta = \Delta\left(Y\right)$.*

Assume f and m are continuously differentiable in an open domain Ω containing B and ∇f and ∇m are Lipschitz continuous in Ω, and that m interpolates f on Y, e.g. $m\left(y^i\right) = f\left(y^i\right)$ for all $y^i \in Y$.

The actual theorem[6] in this regression case is then as follows:

Theorem 4.16 ([39, Thm. 4.1] and [12, Thm. 5.4] combined)**.** *Assume functions f, m, sample set Y and radius $\Delta = \Delta\left(Y\right)$ fulfil Assumption 4.9 with the Lipschitz constant of ∇f and ∇m respectively being denoted by γ_f and γ_m. Then, a bound on the error between the function f and the model m and their derivatives in $B = B\left(y^0; \Delta\left(Y\right)\right)$ is given by*

$$\left\|\nabla f\left(x+s\right) - \nabla m\left(x+s\right)\right\| \leq \frac{5}{2}\sqrt{p}\left\|\hat{L}^{\dagger}\right\|\left(\gamma_f + \gamma_m\right)\Delta$$

and

$$\left\|f\left(x+s\right) - m\left(x+s\right)\right\| \leq \sqrt{p}\left(\gamma_f + \gamma_m\right)\left(\frac{5}{2}\left\|\hat{L}^{\dagger}\right\| + \frac{1}{2}\right)\Delta^2$$

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

Note that, compared to the expressions in Theorem 4.7, $\hat{L}^{-1}$ can in the general setting not be written, as $\hat{L}$ is no longer necessary a square matrix, and the Moore–Penrose pseudoinverse

[6]Note that, although is theorem is placed in the sub-section about underdetermined models, it strictly speaking can be applied to any interpolation model with at least $n+1$ points, and thus for interpolation using different basis too, as for quadratic polynomial interpolation (in which case one might prefer Theorem 4.10 though, since the bounds stated there are stronger.

$\hat{L}^\dagger$ comes into play, which, since $\hat{L}$ is of full column rank because of Y being poised in the linear regression sense, boils down to the left-inverse: $\hat{L}^\dagger = \left(\hat{L}^T\hat{L}\right)^{-1}\hat{L}^T$. Thus, in the linear interpolation sense when $\hat{L}$ is square, $\left\|\hat{L}\right\|$ is present in the upper bounds of this theorem and the model's derivative is constant, we can as mentioned use the theory from Sub-section 4.3.2 to show that a fully linear model can be created based on linear interpolation.

The case of a real underdetermined model, is still left though, as we haven't seen how to bound in that case the right hand side of the inequalities found in Theorem 4.16. If we assume that the model $m(x)$ is a quadratic polynomial, then two different approaches can be found in the literature: the case where the quadratic model polynomial is uniquely defined by minimising the Frobenius norm of the upper or lower triangular part of the Hessian of $m(x)$, as found in [12, Chp. 5], and by minimising the minimum Frobenius norm of the *difference* of (the upper or lower triangular part of) the Hessian, as described for the so-called NEWUOA method by Powell in [28]. As a straightforward implementation of the theory results in this case in a considerable less efficient implementation than possible, those two methods are not part of the discussion here, and the reader is encouraged to reach out to those two sources for a discussion that includes the creation of an efficient algorithm to maintain the model. We will conclude this sub-section by noting that a fully linear class can be created based on underdetermined models (that, as reported, work quite well using $2n+1 = \mathcal{O}(n)$ points, which leads to an improvement over $\mathcal{O}\left(n^2\right)$ for a fully determined model), with an efficient model-improving algorithm.

4.4 Λ-poisedness improvement algorithms

In this section, we will consider algorithms that can generate in a finite and uniformly bounded (with respect to the trust region) number of steps poised sample sets in the ball $B(x;\Delta)$ around $x \in \mathbb{R}^n$ with diameter $\Delta > 0$ that are Λ-poised (in the interpolation sense) or strongly Λ-poised (in the regression sense) for a constant $\Lambda > 1$. Then, together with the results about fully linearity and fully quadratically of such models, by using those algorithms as improvement algorithms, we can conclude that linear and quadratic polynomial interpolation and regression models form actually a class of fully linear and quadratic models respectively. We will consider in this section two methods of managing the interpolation set to fulfil the well-poisedness requirement: one by explicitly controlling the value of the Lagrange polynomials in the set of interest, and one more indirect method, by controlling $\left\|\hat{M}^{-1}\right\|$. We start off with the first one, the one directly controlling Lagrange polynomials, and then shortly afterwards the other. For both those methods, we first consider both methods for the interpolation case only though, and come back to the regression case at the end of this sub-section.

Now, given an set of $p_{\text{ini}}+1$ points whose function value on f has already been evaluated, and thus are more attractive to be considered during the model creation process over other

points whose function value is unknown, we want to select p_1 interpolation points that are poised in the interpolation sense and want to get a set of Lagrange polynomials for them. This can be accomplished by the following algorithm:

Algorithm 4.2 (Calculating Lagrange polynomials for sample set ([12]))**.**

Input *Let $\{\phi_0, \phi_1, \ldots, \phi_p\}$ be a basis of $\mathcal{P}_n^d$ (for example, the natural basis $\bar{\phi}$ or an earlier approximation of the Lagrange polynomials) and let $Y_{\text{ini}} = \left\{y_{\text{ini}}^0, y_{\text{ini}}^1, \ldots, y_{\text{ini}}^{p_{\text{ini}}}\right\}$ be a set of points for which the objective function has already been evaluated. Furthermore, let $B \in \mathbb{R}^n$ be the set of interest.*

Output *A poised (in the interpolation sense) set $Y = \left\{y^0, y^1, \ldots, y^p\right\}$ with corresponding Lagrange polynomials $\{\ell_0, \ell_1, \ldots, \ell_p\}$.*

Initialisation *Set $i = 0$, as iteration counter, and $\ell_i = \phi_i$ for $i = 0, \ldots, p$, as approximation of the basis of Lagrange polynomials.*

Step 1: point selection *If $i \leq p_{\text{ini}} + 1$, find $j_i = \operatorname{argmax}_{i \leq j \leq p_{\text{ini}}} \left|\ell_i\left(y_{\text{ini}}^j\right)\right|$. If $\left|\ell_i\left(y_{\text{ini}}^{j_i}\right)\right| > 0$, add y^{j_i} to Y and swap the points at positions i and j_i in Y_{ini}. Otherwise, if no point has been added in the course of the current step, add $y^i = \operatorname{argmax}_{x \in B} |\ell_i(x)|$ to Y.*

Step 2: Lagrange approximation update *Update*

$$\ell_i(x) \leftarrow \ell_i(x) / \ell_i\left(y^i\right)$$

and

$$\ell_j(x) \leftarrow l_j(x) - \ell_j\left(y^i\right) \ell_i(x), \qquad j = 0, \ldots, p, j \neq i.$$

Step 3: iteration counter update *If $i < p$, set i to $i + 1$ and continue with Step 1. Otherwise, we are done.*

This algorithm tries including points of Y_{ini} in Y in order of encountering them. Thus, by changing the order in which points of Y_{ini} are encountered, the set Y can be changed too, which makes it possible to control Y by applying some heuristics. If the algorithm runs out of points that can be included given the earlier included points, new points are selected that were not in the set of points given as input to the algorithm.

Now, given a set of Lagrange polynomials for a poised interpolation set, we would like to control the Λ-poisedness of the set. This can be done by means of the following algorithm:

Algorithm 4.3 (Improving poisedness via Lagrange polynomials ([12]))**.**

Input *Let $\Lambda > 1$, $Y = \left\{y^0, y^1, \ldots, y^p\right\}$ be a poised interpolation set with $|Y| = p_1$ with corresponding Lagrange polynomials $\{\ell_0, \ell_1, \ldots, \ell_p\}$. Furthermore, let $B \in \mathbb{R}^n$ be*

the set of interest.

Output *A Λ-poised interpolation set Y with corresponding Lagrange polynomials.*

Step 1: estimating poisedness *Estimate*

$$\Lambda_{\text{cur}} = \max_{0 \leq i \leq p} \max_{x \in B} |\ell_i(x)|$$

in such a way that either $\Lambda_{\text{cur}} > \Lambda$ or $\Lambda_{\text{cur}} \leq \Lambda$ can be guaranteed.

Step 2: improving poisedness *If $\Lambda_{\text{cur}} > \Lambda$, let i be such that $\max_{x \in B} |\ell_i(x)| > \Lambda$. Then, replace y^i in Y with y^i_* such that $\left|\ell_i\left(y^i_*\right)\right| > \Lambda$, update the Lagrange polynomials (e.g. following Step 2 in Algorithm 4.2) and continue with Step 1. Otherwise, we are done.*

This algorithm updates the interpolation set iteratively by replacing a point if the corresponding Lagrange polynomial takes too large values with a point on which it takes such value, until the magnitude of the Lagrange polynomials is below the threshold set by Λ.

A priori, it is not clear if we can expect this algorithm to finish in a finite and uniformly bounded (with respect to the trust region) number of steps, or if it finishes at all. Luckily, and perhaps surprisingly, the following theorem tells us it does:

Theorem 4.17 (verbatim from [12])**.** *For any given Λ, a closed ball B and a fixed polynomial basis ϕ, Algorithm 4.3 terminates with a Λ-poised set after at most $N = N(\Lambda, \phi)$ iterations, where N is a constant which depends on Λ and ϕ.*

However, another issue arises with this algorithm when it comes to using it as improvement algorithm, which is that in general no points are guaranteed to be in the output that were originally in the input – including the base point, y^0. Thus, if one was to built an interpolation or regression model around a certain point y^0 as centre of the trust region, y^0 has to be present in the sample set, for which there is no guarantee after running this algorithm. At least one more point should be then changed to take place of y^0, but it is not immediately clear which point, to not destroy the achieved Λ-poisedness.

An observant reader might notice one more issue with this algorithm as improving algorithm, which is that, even if y^0 from the previous paragraph is kept in the new sample set Y_{new}, the new smallest ball around y^0 in which all points of Y_{new} fit might be strictly smaller than the original one: $\Delta(Y_{\text{new}}) < \Delta(Y)$. Then, even though Y_{new} is Λ-poised on $B\left(y^0; \Delta(Y)\right)$, we can so far only guarantee that it is fully linear or fully quadratic for base point y^0 and radius $\Delta(Y_{\text{new}})$. Luckily, again, Lemmata 4.1 and 4.2 solve this issue, as a model fully linear or quadratic on some ball is without loss of generality also fully linear or quadratic on a greater ball.

As final remark, as shortly will be mentioned in Section 4.5, estimating if Λ-poisedness has already been reached could be a time-consuming task, making Step 1 of the algorithm, and the whole algorithm, quite costly.

All in all, Algorithm 4.3 might not be the best option for improving the poisedness, and we consider another algorithm proposed in [10], based on (pivoted) LU-factorisation. In [12], by the same authors, a more general version with a degree of freedom left as method is proposed, which will be the main version discussed here, and is as follows:

Algorithm 4.4 (Improving poisedness via LU factorisation ([10] and [12]))**.**

Input *Let* $Y_{\text{ini}} = \left\{y_{\text{ini}}^0 = 0, y_{\text{ini}}^1, \ldots, y_{\text{ini}}^{p_{\text{ini}}}\right\}$ *be a set of points for which the objective function has been evaluated and* $\xi > 0$ *be a threshold for the pivot values. Furthermore, let* $B \in \mathbb{R}^n$ *be the set of interest.*

Output *A poised interpolation set* Y *with* $\left\|M\left(\bar{\phi}, Y\right)\right\|^{-1} \leq \frac{\sqrt{p_1}\epsilon_{\text{growth}}}{\xi}$ *with* ϵ_{growth} *the growth factor.*

Initialisation *Set* $i = 0$ *as iteration counter and* $u_i = \bar{\phi}_i$ *for* $i = 0, \ldots, p$*, with* p *the dimension of* $\mathcal{P}_n^d$*.*

Step 1: origin addition *Add* $y^1 = y_{\text{ini}}^1 = 0$ *to* Y *and continue with Step 3.*

Step 2: point selection *Choose* $j_i \in \{i, \ldots, p_{\text{ini}}\}$ *such that* $\left|u_i\left(y^{j_i}\right)\right| \geq \xi$*. If such a point is found, add* y^{j_i} *to* Y *and swap the points at positions* i *and* j_i *in* Y_{ini}*.*
Otherwise, if no point has been added in the course of the current step, add $y^i = \text{argmax}_{x \in B}\left|\ell_i(x)\right|$ *to* Y*. If* $\left|u_i\left(y^i\right)\right| < \xi$ *though, we abort the current algorithm, and the output is not valid.*

Step 3: Gaussian elimination *Update for* $j = i+1, \ldots, p$*:*

$$u_j(x) \leftarrow u_j(x) - \frac{u_j\left(y^i\right)}{u_i\left(y^i\right)} u_i(x).$$

Step 4: iteration counter update *If* $i < p$*, set* i *to* $i+1$ *and continue with Step 2. Otherwise, we are done.*

Should this algorithm be run with $\hat{Y}$ as input, then, upon success, the output is a poised, scaled set whose non-scaled version as well is Λ-poised with $p_1 \frac{\epsilon_{\text{growth}}}{\xi}$, thanks to Theorem 4.12. According to [10], should in Step 2 j_i be chosen to maximise $\left|u_i\left(y^{j_i}\right)\right|$ as partial pivoting in Gaussian elimination, the growth factor '[is] expected to be of reasonable size for most practical instances', such that we can conclude that in practice, this algorithm provides for a fixed constants Λ sets Y that are Λ-poised (after the mentioned scaling and unscaling procedure).

The only thing left, is to provide conditions for which Algorithm 4.4 succeeds, e.g. is able to create a set such that the pivot values are at least of size ξ and does not abort in Step 2. The lower the threshold value, the higher the likelihood for the threshold to be reached, and thus the more points to be kept, but also the less well-poised is the resulting set. On the other hand, we cannot expect the algorithm to succeed when we set ξ to any arbitrary large

value; we are thus looking for an upper bound for ξ to make the algorithm still succeed.

The following lemma is of great help for this:

Lemma 4.18 ([12]). *There exists a number $\sigma_\infty > 0$ such that for all vectors v with $\|v\|_\infty = 1$, there exists a $y \in B(\mathbf{0}; 1)$ such that $\left|v^T \phi(y)\right| \geq \sigma_\infty$.*

More specifically, according to [12] too, should $v^T \bar{\phi}(x)$ be a linear polynomial, then the corresponding value for σ_∞ fulfils $\sigma_\infty \leq 1$, while it fulfils $\sigma_\infty \leq \frac{1}{4}$ for $v^T \bar{\phi}(x)$ being a quadratic polynomial. We can apply this lemma to the algorithm, as we set $u_i = \bar{\phi}_i$ at the beginning for $i = 0, \ldots, p$, and in each iteration i, each polynomial u_i is only modified by adding or subtracting polynomials u_j for $j = 0, \ldots, i-1$ from it, such that the coefficient of $u_i(x)$ corresponding to $\bar{\phi}(x)_i$ is 1, and we can write $u_i(x) = v_i^T \bar{\phi}(x)$ with $\|v_i\|_\infty \geq 1$. Then, according to the lemma, $\left\| \frac{1}{\|v_i\|} v_i^T \bar{\phi}(x) \right\| \geq \sigma_\infty \Leftrightarrow \left\| v_i^T \bar{\phi}(x) \right\| \geq \|v_i\| \sigma_\infty \geq \sigma_\infty$, and by setting ξ as threshold parameter in Algorithm 4.4 such that $\xi \leq \sigma_\infty$, we can guarantee the algorithm to succeed.

Now that we have discussed how to create sets that are Λ-poised, we are interested in creating strongly Λ-poised sets. For this, we use the motivation of the definition of strongly Λ-poisedness of a set Y, which we recall as being able to split the set Y into $l = \lfloor p/q \rfloor$ subset that are Λ-poised (and another subset of less than q_1 points). This leads to the following algorithm:

Algorithm 4.5 (Improving strongly poisedness ([12])).

Input *Let Y_{ini} be a set of points for which the objective function has already been evaluated, and let either $\Lambda > 1$, or $\xi > 0$. Furthermore, let $B \in \mathbb{R}^n$ be the set of interest.*

Output *Either a strongly Λ-poised set Y, possible with $\Lambda = p_1 \frac{\epsilon_{\text{growth}}}{\xi}$ with $\epsilon_{\text{growth}} > 0$ a growth factor expected to be of moderate size for most practical instances when applying partial pivoting in Gaussian elimination in Algorithm 4.4.*

Step 1 *If $|Y_{\text{ini}}| < q_i$, continue with Step 4.*

Step 2 *Apply either Algorithms 4.2 and 4.3, or Algorithm 4.4 to generate a poised subset Y^i.*

Step 3 *Remove all points in Y^i from Y_{ini}, set i to $i+1$ and continue with Step 1.*

Step 4 *Set $Y = \bigcup_i Y^i$, possible with the remaining points of Y_{ini} added.*

We note that the number of points in the output set is highly depending on the poisedness of the input set: should the points be badly poised, then each subset might contain many newly generated points. One could therefore decide to break the algorithm prematurely, if one considers the number of points in the output set as sufficient.

4.5 Minimisation sub-problem

Part of the core idea of a trust-region method is that instead of directly minimising the objective function, a function with an easier structure, the approximation, is locally minimised after which the result of this (hopefully) helps us in an iterative process to minimise the objective function itself. In the previous sections, we have seen the main minimisation methods directly working on the objective function itself, and examples of the approximation; in this section, we consider how to minimise those locally.

As seen in Section 4.2, it is not needed to seek the local minimum for this – a baseline for the quality of the minimiser in terms of decrease is for the first-order trust-region set by the Cauchy point in Assumption 4.4 and for the second-order trust-region in addition set by the eigenpoint in Assumption 4.5. More specifically, the Cauchy point or the best of the Cauchy point and eigenpoint itself can be taken as approximate minimiser while still fulfilling the assumptions for the convergence statements, with especially the Cauchy point being relatively easily computable. It has been claimed in though [11, Chp. 6] that such an approach leads to slow convergence, and thus, we consider in this section more sophisticated ways of computing a minimiser for which baseline of the Cauchy and eigenpoint is nevertheless reached, and especially a minimiser of a quadratic polynomial from the form (4.7) for the models considered in Section 4.3.

Finding extreme values of linear and quadratic polynomials also plays a role in the poisedness-improving methods, but this time as maximisation problem, which is of equivalent difficulty. However, other requirements on the minimum increase are imposed this time. The algorithm improving the Λ-poisedness directly using Lagrange polynomials, Algorithm 4.3, requires access to a point whose function value exceed Λ, if any, while the algorithm improving the poisedness using LU-factorisation, Algorithm 4.4, needs to find points resulting in a function value greater than constant $\xi > 0$. Luckily, by bounding $\xi \leq \sigma_\infty$, an explicit formula for such points can be stated – which can be found in [12, proof of Thm. 6.7] – to fulfil the requirements at the minimum, and we will not consider explicitly points that need to fulfil this criterion in the rest of this section.

Now, as stated, we consider a quadratic polynomial

$$q(x) = \frac{1}{2}x^T H x + c^T x$$

that we want to minimise, for $\|x\| \leq \Delta$ (with $\Delta > 0$ being a constant), where the constant term of the quadratic polynomial is without loss of generality set to 0. An exact algorithm for solving such a problem, which is of help for all problem earlier described in this section, is described in [11, Chp. 7.3]. As stated by them though, the cost of this algorithm is not insignificant, and we consider methods finding (in general) approximate minimisers.

As one of such alternatives, we consider a method minimising $q(x)$ would the problem have been unconstrained and would $q(x)$ have been a strictly convex function (thus, would

H have been positive definite). This method is described as follows:

Algorithm 4.6 (Fletcher–Reeves Conjugate Gradient Method ([11]))**.**

Input *Let $x_0 \in \mathbb{R}^n$ be a starting point and $q(x) = \frac{1}{2}x^T Hx + c^T x$ a strictly convex quadratic function (that is, with H being positive definite) to be minimised.*

Initialisation *Set $g_0 = Hx_0 + c$, $p_0 = -g_0$ and $l = 0$ as iteration counter.*

Output *The minimiser $x_* = x_l$ of $q(x)$.*

Step 1 *If $g_l = 0$, convergence has occurred and we stop.*

Step 2 *Set*

$$\begin{aligned} \alpha_l &= \|g_l\|^2 / p_l^T H p_l; \\ x_{l+1} &= x_l + \alpha_l p_l; \\ g_{l+1} &= g_l + \alpha_l H p_l; \\ \beta_l &= \|g_{l+1}\|^2 / \|g_l\|^2; \\ p_{l+1} &= -g_{l+1} + \beta_l p_l, \end{aligned}$$

set l to $l+1$ and continue with Step 1.

As explained in [11], each next x_{l+1} is the minimiser of a line-search applied along the direction p (for H being positive definite), with each direction p_l being H-conjugate with each predecessors, such that $p_l^T H p_{l'} = 0$ for each $l' < l$. The conjugate directions are now chosen in such a way that the they form a nested set of subspaces of increasing dimension such that H continues to be positive definite in those subspace, such that after $l = n$ iterations, $\mathbb{R}^n$ is spanned and the resulting point is (indeed) a (the) minimiser in $\mathbb{R}^n$ of the convex function $q(x)$.[7]

However, while the conjugate gradient method helps with finding the minimiser in case of H being positive definite, we need to consider the case of H being not too, and keep in mind the constraint $\|x\| \leq \Delta$. In the first case, it is claimed that $q(x)$ for x on the direction p_l with base point x_l is unbounded from below, and thus, the point providing the lowest value for q along that line for $\|x\| \leq \Delta$ is on the boundary. It has been proved that once this boundary condition is invalidated, we will not return back, and thus, we can then define the following algorithm as truncated variant of the conjugate gradient method as follows:

[7] In practice, using floating-point arithmetic, convergence has not necessarily occurred after n steps. One could modify Algorithm 4.6 to terminate once the gradient is small enough to try to tackle this problem. Furthermore, H might be preconditioned, as shown in [11], to speed-up the approach of the minimiser of $q(x)$, with possibly positive effects on the convergence properties in floating-point arithmetic.

Algorithm 4.7 (Truncated Fletcher–Reeves Conjugate Gradient Method ([11]))**.**

Input *Let $x_0 \in \mathbb{R}^n$ be a starting point and $q(s) = \frac{1}{2}s^T Hs + c^T s$ a quadratic function to approximately be minimised for $\|s\| \leq \Delta$.*

Initialisation *Set $s_0 = 0$, set $g_0 = Hs_0 + c = c$, $p_0 = -g_0$ and $l = 0$ as iteration counter.*

Output *The approximate minimiser $s_* = s_l$ of $q(s)$.*

Step 0: exit step [a] *Compute σ_l such that $\|s_l + \sigma_l p_l\| = \Delta$ (by taking a square root), and $q(s_l + \sigma_l p_l)$ has the smallest value among the possible (two) options (by simple comparing the values), set $s_{l+1} = s_l + \sigma_l p_l$, set l to $l+1$ and stop.*

Step 1 *If $g_l = 0$, convergence has occurred and we stop.*

Step 2 *Set $\kappa_l = p_l^T H p^l$.*

If $\kappa_l \leq 0$ [and, thus, H has not the properties of a positive definite matrix], continue with Step 0.

Step 3 *Set $\alpha_l = \|g_l\|^2 / \kappa_l$.*

If $\|s_l + \alpha_l p_l\| \geq \Delta$ [and we are going to leave the trust-region], continue with Step 0.

Step 4 *Set*

$$\begin{aligned} s_{l+1} &= s_l + \alpha_l p_l; \\ g_{l+1} &= g_l + \alpha_l H p_l; \\ \beta_l &= \|g_{l+1}\|^2 / \|g_l\|^2; \\ p_{l+1} &= -g_{l+1} + \beta_l p_l, \end{aligned}$$

set l to $l+1$ and continue with Step 1.

[a] Note that the algorithm starts with Step 1.

One could easily see that the s_1 (or s_0, if g_0 happened to be 0) is a Cauchy point of the bounded quadratic model to minimise, and thus, the output point of this algorithm is a Cauchy point:

Lemma 4.19. *Let s_k be generated by applying Algorithm 4.7 on $m_k(x)$. Then, Assumption 4.4 is fulfilled.*

As the truncated conjugate gradient method rather abruptly stops execution once the minimum of one manifold lies on the boundary (and does not consider the other dimensions), a natural question is whether the method can be improved, as otherwise, the resulting point is only slightly better than a Cauchy point. In [11], a more complicated method is explained tries to tackle this problem under the name of the Lanczos approach. In [11][Chp. 7.5.5], a connection to the computation with the eigenpoint is also made.

4.6 Summary

In Section 4.1, the framework of fully linear and fully quadratic classes have been shown, taking over the role of Taylor models in the regular trust-region method. Based on this, the derivative-free version has then been stated in Section 4.2 in two variants: for first-order convergence or second-order convergence, with stronger convergence properties for the latter, but requiring also mode from the model.

In Section 4.3 we have seen polynomial linear or quadratic interpolation models, that are possibly overdetermined or underdetermined, with different theorems providing bounds on the error between the model and the objective, and their gradient or Hessian, if applicable. By enforcing the sample set of points to base the interpolation one to be well-poised, the error bounds depend apart from a constant only on the trust-region size, and those models form a fully linear or fully quadratic model.

What is left for being able to create a fully linear or fully quadratic class based on those interpolation models, is the ability to obtain a model in a bounded number of steps, which in this case translates to the ability to obtain a well-poised set of sample points. With the preference to keep as many points as possible to restrict the number of evaluations, algorithms for this are presented in Section 4.4.

Chapter 5

Discrete Gradient Method

After having seen the trust-region method, being explicitly designed for continuously differentiable functions, and the Nelder–Mead method, with a simple intuition, the method in this chapter has been designed to work for (a subset of) non-differentiable functions.

The discrete gradient method, as first described in [2], has been inspired by the concept of the subdifferential, which provides for a non-stationary point a descent direction. By getting with accurately sufficient accuracy such points too by calculating the so-called discrete gradient, this method is also able to find descent directions and find stationary points.

In this chapter, we review the discrete gradient method and its convergence properties. To study this, different properties of functions are described, to classify the functions convergence takes place for. Furthermore, we study the sub-problem of this method, for which we want to find the point in a polytope nearest to the origin, and we provide an improved version of an algorithm known from the literature.

5.1 Idea and background

The goal of the iterative discrete gradient method for non-differentiable functions is to approximate with different precision the subdifferential at the current iteration point well enough to either obtain a descent direction to create the next iteration point, or to conclude that a local minimum is reached at the current iteration point, for the current precision. Depending on how satisfied we are with the current precision, we either increase the precision, or we return from the method. One of the properties the objective functions is assumed to have is the property of Lipschitz continuity. Functions with this property are almost everywhere differentiable and Theorem 2.11 provides for such functions a way to compute the subdifferential at some iteration point as convex hull of the limit of the gradient of the points converging to this iteration point.

In this setting, we will approximate those gradients with the so-called discrete gradient. In the definition of the discrete gradient, we write G for the set of all vertices of a hypercube,

as

$$G \coloneqq \left\{ \begin{bmatrix} e_0 & \cdots & e_{n-1} \end{bmatrix}^T \in \mathbb{R}^n : \forall j = 0, \ldots, n-1, |e_j| = 1 \right\}.$$

The definition of the discrete gradient is then as follows:

Definition 5.1 ([2])**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. A *discrete gradient* of f at x is defined as the vector $\Gamma^i(x, d, e, \lambda, \alpha) = \begin{bmatrix} \Gamma_0^i & \cdots & \Gamma_{n-1}^i \end{bmatrix}^T$ for $d \in S^{n-1}$, $i = \operatorname{argmax}_j \|d_j\|$, $e \in G$, $\lambda > 0$ and $\alpha \in (0, 1]$ with its components being defined by

$$\Gamma_j^i \coloneqq \begin{cases} \left(\lambda \alpha^j e_j\right)^{-1} \left(f(x_j) - f(x_{j-1})\right), & j \neq i \\ (\lambda d_i)^{-1} \left(f(x + \lambda d) - f(x) - \lambda \sum_{j=0, j\neq i}^{n-1} \Gamma_j^i d_j\right), & \text{otherwise} \end{cases} \tag{5.1}$$

for the points $x_0 \coloneqq x + \lambda d$ and $x_j = x_0 + \lambda \begin{bmatrix} \alpha e_1 & \alpha^2 e_2 & \cdots & \alpha^j e_j & 0 & \ldots & 0 \end{bmatrix}^T \in \mathbb{R}^n$.

In line with our motivation of defining the discrete gradient for non-differentiable functions, the definition of Γ_i^i has been chosen by [2] in such a way that its norm is bounded by a constant determined by the objective function, and that its inner product with d makes the following expression to hold:

$$f(x + \lambda d) - f(x) = \lambda \Gamma^i(x, d, e, \lambda, \alpha)^T d. \tag{5.2}$$

We can compare the role of the discrete gradient in the similar formula from the Mean-Value Theorem, for an expression for the gradient at a point between x and $x + \lambda d$, for some $\kappa \in [0, 1]$:

$$f(x + \lambda d) - f(x) = \lambda \nabla f(x + \kappa \lambda d)^T d. \tag{5.3}$$

Example 5.1. Let $f : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto x^2 + y^2$, $x = (0, 0)$, $\lambda = 1/2$ and $d = \left(1/2, \sqrt{3}/2\right)$. With these parameters, we compute a discrete gradient of f at x, and compare the value with the regular gradient of f at some point, to compare (5.2) and (5.3) directly.
As additional parameters as needed for the computation of the discrete gradient, we take in this example $i = 2$, $e = (1, 1)$ and $\alpha = 0.5$. The points to evaluate f on, besides x itself, are given by the following points in the surrounding of x:

$$\begin{aligned} x_0 &= x + \lambda d = \left(1/4, \sqrt{3}/4\right) \\ x_1 &= x_0 + \lambda(\alpha e_1, 0) = \left(1/2, \sqrt{3}/4\right) \\ x_2 &= x_0 + \lambda\left(\alpha e_1, \alpha^2 e_2\right) = \left(1/2, 1/8 + \sqrt{3}/4\right) \end{aligned}$$

Filling in (5.1) for this results in a vector $\Gamma^i(x, d, e, \lambda, \alpha)$ such that $\Gamma^i(x, d, e, \lambda, \alpha)^T d =$

0.875. The gradient of f is given by $\nabla f(x,y) = \begin{bmatrix} 2x & 4y \end{bmatrix}^T$, such that, for the point $x + \kappa\lambda d$ with $\kappa = 1/4$, $\lambda \nabla f(x + \kappa\lambda d)^T d = 0.875$ holds as well.

With this, the idea is to let λ in the definition of the discrete gradient represent the precision we query the discrete gradient for, with the precision being given by a sequence $(\lambda_k)_{k\in\mathbb{Z}}$ of positive numbers converging to zero. This way, by calculating discrete gradient vectors at x in randomly-chosen directions for those values λ_k, we strive for getting a better approximation of the subdifferential $\partial f(x)$ at x. However, to get a meaningful approximation following this motivation, we need to impose more restrictions on the behaviour of the function; the function has to be semi-smooth:

Definition 5.2 ([22])**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. Then, f is called *semi-smooth* at x if

- f is locally Lipschitz continuous at x, and
- for all directions $d \in \mathbb{R}^n$, for all sequences $(\lambda_k)_{k\in\mathbb{Z}}$ of positive numbers with $\lambda_k \to 0^+$ and all sequences $(\theta_k)_{k\in\mathbb{Z}}$ and $(x_k)_{k\in\mathbb{Z}}$ of vectors with $\theta_k/\lambda_k \to 0$ and $g_k \in \partial f(x + \lambda_k d + \theta_k)$, the sequence $\left(g_k^T d\right)_{k\in\mathbb{Z}}$ has exactly one accumulation point.

Lemma 5.1. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a semi-smooth function at $x \in \mathbb{R}^n$. Then, for each $d \in \mathbb{R}^n$, $f'(x;d)$ exists and for a sequence $(g_k)_{k\in\mathbb{Z}}$ fulfilling the properties from Definition 5.2,*

$$f'(x;d) = \lim_{k\to\infty} g_k^T d. \tag{5.4}$$

Thus, for a non-differentiable function that is semi-smooth at some point x, the subgradient vectors the closer to x should nevertheless behave in a regular way. More specific, continuously differentiable functions are always semi-smooth. This way, for the discrete gradient method, our goal is to expect the approximations of the subdifferential to have a meaningful interpretation. An example of two functions having otherwise two different properties on this matter can be found in Figure 5.1.

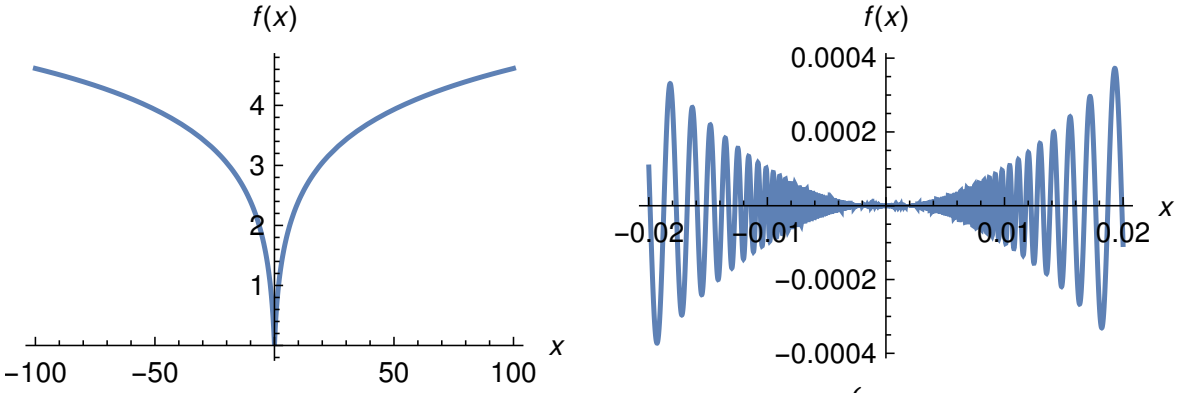
We can relate (5.4) with a formula known from the derivative setting, as we have by Theorem 2.1 for f a differentiable function at $x \in \mathbb{R}^n$, for all $d \in \mathbb{R}^n$,

$$f'(x;d) = \nabla f(x)^T d.$$

Lastly, as property to have for the functions to apply the discrete gradient method on, we recall a (natural) generalisation of the subdifferential, based on (2.9):

Definition 5.3 ([13])**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$. Then is f called *quasidifferentiable* at $x \in \mathbb{R}^n$ if it is directionally differentiable and there exists a pair of convex and compact sets $\underline{\partial} f(x), \overline{\partial} f(x) \subseteq \mathbb{R}^n$ such that it holds that

$$f'(x;d) = \max\left\{\xi^T d\colon \xi \in \underline{\partial} f(x)\right\} + \min\left\{\xi^T d\colon \xi \in \overline{\partial} f(x)\right\}$$



(a) For $f(x) = \log 1 + |x|$.

(b) For $f(x) = \begin{cases} x^2 \sin 1/x, & x \neq 0 \\ 0, & \text{otherwise.} \end{cases}$

Figure 5.1: Two different functions, found in [21], to illustrate the independence of differentiability and semi-smoothness with the extreme cases; the one on the left is non-differentiable, but semi-smooth, while the one on the right is differentiable, but not semi-smooth.

for every $d \in \mathbb{R}^n$.

The pair $\left(\underline{\partial} f(x), \overline{\partial} f(x)\right)$ is then called a *quasidifferential*, with $\underline{\partial} f(x)$ the *subdifferential* and $\overline{\partial} f(x)$ the *superdifferential*.

As generalisation, to be used for the discrete gradient method on non-differentiable functions, it has been shown in [13] that continuously differentiable functions are quasidifferentiable and quasidifferentiable functions form a linear space that is also closed under the point-wise min and max functions.

All in all, we consider functions for which the following assumption holds:

Assumption 5.1. *Assume that $f\colon \mathbb{R}^n \to \mathbb{R}$ is a semi-smooth, quasidifferential function whose subdifferential and superdifferential of a quasidifferentiable function for any $x \in \mathbb{R}$ are polytopes.*

Assume furthermore, that for the expression[1] *for $\lambda > 0$ and $d \in \mathbb{R}^n$*

$$f(x + \lambda d) - f(x) = \lambda f'(x; d) + o(\lambda, d),$$

it holds that $\lambda^{-1} o(\lambda, d)$ converges uniformly to 0 for $\lambda \to 0$.

With the needed classification of functions – which according to [4] includes differentiable, non-smooth convex and non-smooth DC functions – we can state the theorem relating the discrete gradients with the subdifferential of a Lipschitz function by looking at the convex set of some discrete gradients, given in the set

$$D_0(x, \lambda, \alpha) = \operatorname{cl}\operatorname{conv}\left\{\Gamma^i(x, d, e, \lambda, \alpha) \in \mathbb{R}^n \colon g \in S^{n-1}, e \in G\right\}. \tag{5.5}$$

[1] From the definition of the directional derivative, (2.1).

Concluding this section, we present the main theorem, which can be considered the equivalent of Theorem 2.11, but easier applicable:

Theorem 5.2 ([2])**.** *Consider a function $f\colon \mathbb{R}^n \to \mathbb{R}$ for which Assumption 5.1 holds. Then, for any $\epsilon > 0$, there exist $\lambda_0 > 0$ and $\alpha_0 > 0$ such that*

$$D_0(x, \lambda, \alpha) \subset \partial f(x) \oplus B(\mathbf{0}; \epsilon) \text{ for any } \lambda \in (0, \lambda_0], \alpha \in (0, \alpha_0].$$

Furthermore, we are able to extract a descent direction from the set $D_0(x, \lambda, \alpha)$, equivalent to Lemma 2.17:

Lemma 5.3 ([2])**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$. Then, there exist $\lambda_0 > 0$ and $\alpha_0 > 0$ such that either $\mathbf{0} \in \partial f(x)$ or $-\xi^*/\|\xi^*\|$ is a descent direction, where $\xi^* = \operatorname{argmin}_{\xi \in \partial f(x)} \|\xi\|$.*

5.2 Method description and convergence properties

In the previous section, a sketch of the discrete gradient method has been given, with different classifications of functions on which those ideas could be applied. With this motivation and the last two results, a more precise description of the discrete gradient method is given by the following:

Algorithm 5.1 (Discrete Gradient Method ([2]))**.**

Input *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be the function to be minimised and $x_0 \in \mathbb{R}^n$ a starting point. Furthermore, we require the following constants as parameters of the method:*

- *$\alpha \in (0, 1]$: controlling the factor of the step size in calculating the discrete gradient;*
- *$c \in (0, 1)$ and $c_2 \in (0, c]$: controlling the required decrease for a serious step to take place and to search in a discrete descent direction;*
- *$(d_k)_{k\in\mathbb{Z}}$ with $d_k > 0$ and $\lim_{k\to\infty} d_k = 0$ and $(\lambda_k)_{k\in\mathbb{Z}}$ with $\lambda_k > 0$ and $\lim_{k\to\infty} \lambda_k = 0$: controlling the threshold for an inner iteration to be finished and the step size respectively.*

Output *A sequence of iteration point $(x_k)_{k\in\mathbb{N}}$.*

Initialisation *Set $k = 0$.*

Step 1 (inner iteration initialisation) *Set $l = 0$, $x_{k_l} = x_k$ and choose $d'_k \in S^{n-1}$ and $e'_k \in G$. Choose $i = \operatorname{argmax}_{j=1,\dots,n} |g_j|$ and compute the discrete gradient $v_{k_l} = \Gamma(x_{k_l}, d'_k, e'_k, \lambda_k, \alpha)$ at x_{k_l}.*

Step 2 (inner iteration termination) *Compute*

$$\bar{v}_{k_l} = \operatorname{argmin}_{v \in \bar{V}(x_{k_l})} \|v\| .$$

If $\|\bar{v}_{k_l}\| \leq \delta_k$*, set* $x_{k+1} = x_{k_l}$*, set* k *to* $k+1$ *and continue with Step 1 in a new outer iteration.*

Step 3 (inner iteration classification) *Compute*

$$d_{k_l} = -\frac{\bar{v}_{k_l}}{\|\bar{v}_{k_l}\|}.$$

If

$$f(x_{k_l}) - f(x_{k_l} + \lambda_k d_{k_l}) \geq c\lambda_k \|\bar{v}_{k_l}\| , \tag{5.6}$$

the decrease is sufficient for a serious step to be taken as next step, and we continue with Step 4. Otherwise, a null step is taken, and we continue with Step 5.

Step 4 (serious step) *Apply a line search starting at* x_{k_l} *in the direction of* d_{k_l} *with parameter* $t_{k_l} > 0$ *such that for* $x_{k_{l+1}} := x_{k_l} + t_{k_l} d_{k_l}$*, it holds*[a] *that*

$$f(x_{k_l}) - f\left(x_{k_{l+1}}\right) \geq c_2 t_{k_l} \|\overline{v}_{k_l}\| . \tag{5.7}$$

Choose $d'_{k_l} \in S^{n-1}$ *and* $e'_{k_l} \in G$ *and compute the discrete gradient for the next iteration as*

$$v_{k_{l+1}} = \Gamma\left(x_{k_l}, d'_{k_l}, e'_{k_l}, \lambda_k, \alpha\right),$$

which will be the only element of the set of computed discrete gradients for the new inner iteration point so far: $\bar{V}\left(x_{k_{l+1}}\right) = \left\{v_{k_{l+1}}\right\}$.
Set l *to* $l+1$ *and continue with Step 2.*

Step 5 (null step) *Keep the current iteration point:* $x_{k_{l+1}} = x_{k_l}$ *and compute the discrete gradient for the next iteration in the new direction as*

$$v_{k_{l+1}} = \Gamma\left(x_{k_l}, d_{k_l}, e, \lambda_k, \alpha\right),$$

which will be added to the set of discrete gradients for the current inner iteration point:

$$\bar{V}\left(x_{k_{l+1}}\right) = \operatorname{conv}\left(\bar{V}\left(x_{k_l}\right) \cup \left\{v_{k_{l+1}}\right\}\right).$$

Set l *to* $l+1$ *and continue with Step 2.*

[a] This line search allows faster convergence, by allowing a wider exploration of a descent direction. For theoretical convergence, $t_{k_l} = \lambda_k$ suffices though.

A visualisation of the change of iteration points through the parameter space can be found in Figure 5.2 for a 2-dimensional function.

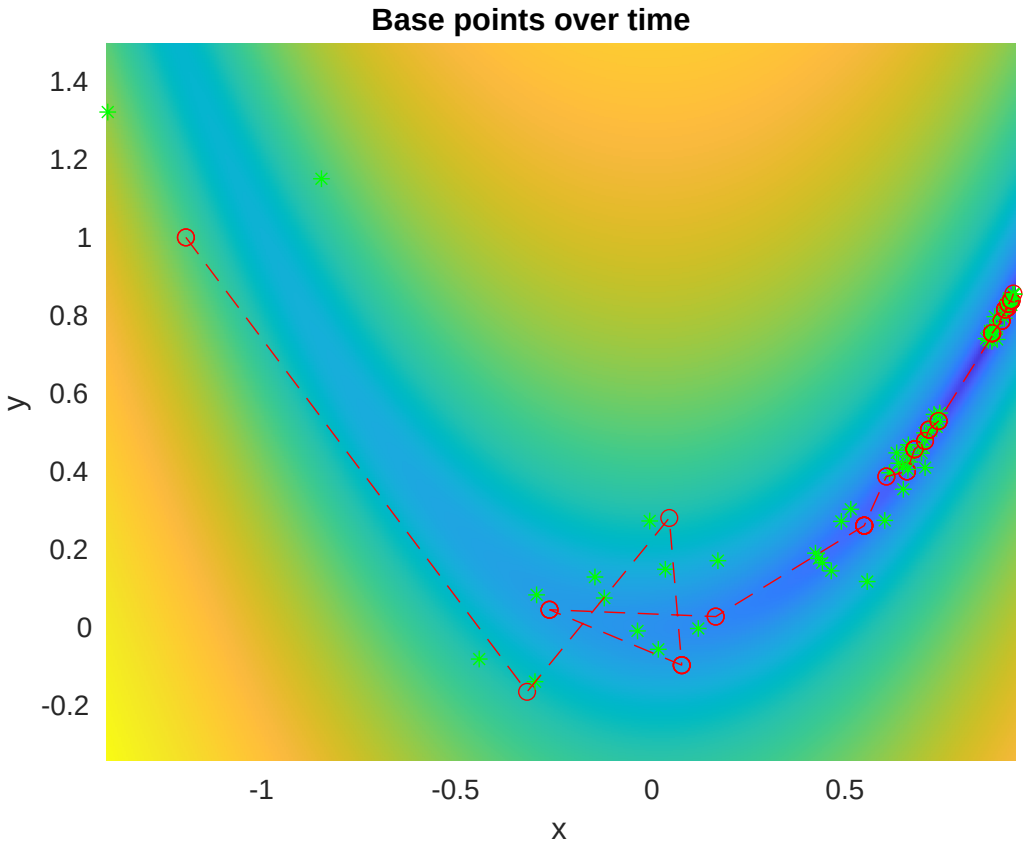


Figure 5.2: Base points and discrete gradient iteration points evaluated over the course of the FDSS method using Implementation A.3 of the so-called Rosenbrock function $(x, y) \mapsto 100\left(y - x^2\right)^2 + (1 - x)^2$ as found in [30] with starting point $(-1.2, 1)$. The red points mark the iteration points, with the green points those on which the discrete gradient is evaluated.

Two different approaches for the line search in Step 3 are reported. In [2] and [4], it has been suggested to seek the biggest value $m_{k_l} \in \mathbb{N}$ such that for $t_{k_l} = m_{k_l}\lambda_k$, (5.7) holds, with an algorithm incrementing m_{k_l} and checking this condition for each increment. Thus, in the direction of d_{k_l} from point x_{k_l}, given a point of sufficient decrease in function value, a line search is executed for a point in the same direction that is farther away, but might provide a slightly less decrease than the original point. The goal here is thus to rather seek the step with the longest step size that still fulfils the decrease condition, at the cost of potentially many function evaluations as the step size is each time increased with the same amount.

An alternative was used in the implementation that accompanies [3, Chp. 17]. Here, we seek the smallest value $p_{k_l} \in \mathbb{N}$ such that for $t_{k_l} = \bar{v}_{k_l}/2^{p_{k_l}}$, (5.7) holds, or $t_{k_l} < \lambda_{k_l}$, in which case t_{k_l} is set to λ_{k_l}. Compared to the first approach, instead evaluating increasingly farther away points, points becoming closer to iteration point are evaluated starting from a point at some distance, $\bar{v}_{k_l}$, until we are closer than the already-found descent point since (5.6) holds. With this, this approach has the advantage of the number of steps being more regular.

A convergence theorem for the discrete gradient method is then the following:

Theorem 5.4 ([2])**.** *Consider a function $f\colon \mathbb{R}^n \to \mathbb{R}$ for which Assumptions 5.1 and 4.6 hold. Let $(x_k)_{k\in\mathbb{N}}$ be a sequence of iteration points resulting from applying Algorithm 5.1 on f.*

Then, for every accumulation point $\hat{x}$ of that sequence $(x_k)_{k\in\mathbb{N}}$, it holds that $\mathbf{0} \in \partial f(\hat{x})$.

As termination criterion for practical usage, it has been suggested in[4] to terminate at iteration k once both λ_k and δ_k are smaller than some threshold $\eta > 0$. A rationale for this can be found by considering Theorem 5.2. The smaller the value of λ_k, the smaller the value ϵ – measuring the uncertainty in our approximation of the subdifferential at some point – can take in this theorem. The smaller δ_k, the smaller the norm of a minimum-norm vector in this subdifferential approximation, and with 0 indicating a stationary point, the greater the chance of having found a stationary point.

5.3 Nearest point in a polytope

An important part of the discrete gradient method – found in Step 2 in the version shown in Algorithm 5.1 – is the retrieval of a point inside a polytope that is nearest to the origin, for which we will review in this sub-section an algorithm first described in [40]. Here, an algorithm specifically designed for the problem of finding this minimum-norm vector of a convex set was presented, as opposed to solving this with a description of the problem as input for some generic (quadratic programming) solver.

The idea behind the algorithm is that while finding a point nearest to the origin in a convex hull is difficult, finding such point in an affine hull can be done more easily, and that checking the optimality of such point as solution of the original problem can also easily be checked. The algorithm below, as geometrical variant found in [40], is moving points in and out in the set in some structured until the correct points for the affine hull are chosen:

Algorithm 5.2 (Wolfe's method for finding that point in a polytope nearest to the origin (geometrical) ([40]))**.**

Input *Let $P \subset \mathbb{R}^n$ be a set of finite points.*

Output *The point in* $\operatorname{conv} P$ *nearest to the origin.*

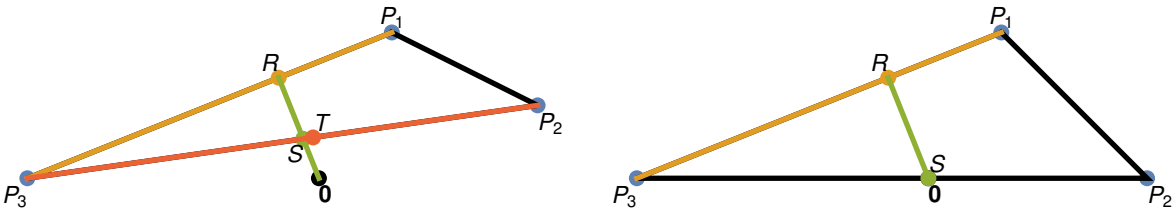
Initialisation *Let*[a] *$Q \subset P$ and $X \in \operatorname{conv} Q$.*

Step 1 *If $X = 0$ or the hyperplane $\left\{x \in \mathbb{R}^n \colon X^T x = X^T X\right\}$ separates P from the origin, return with X.*

Otherwise, a point in P on the side of the origin is added to Q.

Step 2 *Let Y the point in* $\operatorname{aff} Q$ *nearest to the origin.*

If $Y \in \operatorname{ri}\operatorname{conv} Q$, replace X by Y and continue with Step 1.



(a) $P = \{P_1 = (1,2), P_2 = (3,1), P_3 = (-4,0)\}$ (b) $P = \{P_1 = (1,2), P_2 = (3,0), P_3 = (-4,0)\}$

Figure 5.3: Application of Wolfe's method for finding the nearest point in a polytope on two two-dimensional point sets, P. In both cases, P_1 is first considered, as point nearest to the origin. Not being a solution for the problem, P_3 is considered, with the nearest point in the convex hull of P_1 and P_3 being R. Then, not being the result either, P_2 is considered, whose affine hull spans the whole space, and thus the origin being the point in the affine hull of P. In one case, this point is also in the convex hull, while in the other case, we get S as point in the convex hull on the line between $\mathbf{0}$ and R. With S lying on the line segment between P_3 and P_2, we seek the point nearest to the origin here, as T, for our solution in the other case.

Step 3 *Otherwise, if* $Y \notin \operatorname{ri}\operatorname{conv} Q$*, let* Z *be the nearest point to* Y *on the line segment* $\overline{XY}$ *through* $\operatorname{conv} Q$*.*

Since X *is in* $\operatorname{conv} Q$ *and* Z *not,* Z *lies on the boundary of* $\operatorname{conv} Q$ *and a point in* Q *is not on the face of* $\operatorname{conv} Q$*. Delete from* Q *this point, replace* X *by* Z *and continue with Step 2.*

[a] A possible choice of Q in the initialisation phase would be a singleton containing the nearest point to the origin of P.

Two examples of application of this method can be found in Figure 5.3.

A proof of correctness of this algorithm can be found in [40], where it is shown that the algorithm visits only a finite number of combinations of points in Q, and thus terminates in finite time. In [19], it has then been shown that with some specific insertion rule, the method can take exponential time.

Based on this geometric interpretation, an algebraic method too was proposed in [40], which would be suitable for implementation in common programming patterns by a direct translation of the geometric steps. Exactly with the aim of an implementation, the version as stated there has got several safeguards that allow imprecision in finite-precision arithmetic, by allowing a wider range to pass through conditional statements. For the purpose of analysis, in the description below, those safeguards are left out, but are instead present in the MATLAB implementation from Implementation A.2.

The (exact) algebraic version of Algorithm 5.2 is given below. The sub-problem of this method, of finding the minimum-norm vector in an affine hull, is solved by solving the (easy) problem for the initial data, keeping data structures needed for solving this and updating those when points are moved in and out the current point set. Four different methods for doing this have been described in [40], of which 'method D' from Section 5 has been used in

MATLAB implementation in Implementation A.2.

Algorithm 5.3 (Wolfe's method for finding that point in a polytope nearest to the origin (algebraic) ([40] in exact version with modification on calculation of θ))).

Input *Let $P \subset \mathbb{R}^n$ be a set of m finite points.*

Output *The point in* $\operatorname{conv} P$ *nearest to the origin*

Initialisation[a] *Let* $J_0 = \operatorname{argmin}_{j\in\{0,\dots,m\}} \left\|P^j\right\|$.

Set $w_0 = \begin{bmatrix}1\end{bmatrix}^T$ *and* $S_0 = \begin{bmatrix}J\end{bmatrix}^T$ *and* $X_0 = P^{J_0}$. *Set* $m_0 = 1$ *and* $k = 0$.

Step 1 *Let* $J_{k+1} = \operatorname{argmin}_{j\in\{0,\dots,m-1\}} X_k^T P^j$.

If $X_k^T P_{k+1}^J > X_k^T X_k$, *return with* X_k.

Otherwise, we add the J*th point of* P *to our current set of points to take into account by setting* $w_{k_0} = \begin{bmatrix} w_k^0 & \dots & w_{m_k}^0 & 0\end{bmatrix}^T$ *and* $S_{k_0} = \begin{bmatrix} S_k^0 & \dots & S_{m_k}^0 & J_{k+1}\end{bmatrix}^T$. *Set* $m_{k_0} = m_k + 1$ *and set* l *to* $l+1$.

Step 2 *We define the matrix* $P^{(S_k)}$ *as the matrix* P *where every* i*th column corresponds to the* S_k^i*th vector from* P, *thus* $P^{(S_k)} \coloneqq \begin{bmatrix} P^{S_k^0} & \dots & P^{S_k^{m_k-1}}\end{bmatrix}$.

Then, solve for $(\lambda, v)_{k_l}$ *with* $\lambda_{k_l} \in \mathbb{R}$ *and* $v_{k_l} \in \mathbb{R}^{m_{k_l}}$

$$\begin{cases} \sum_{i=0}^{m_{k_l}-1} v_{k_l}^i & = 1; \\ \lambda_{k_l}\mathbf{1} + {P^{(S_{k_l})}}^T P^{(S_{k_l})} v_{k_l} & = \mathbf{0}. \end{cases} \tag{5.8}$$

If, for all $i \in \{0, m_{k_l} - 1\}$, $v_{k_l}^i > 0$, *set* $w_{k+1} = v_{k_l}$ *and* $S_{k+1} = S_{k_l}$, *set* k *to* $k+1$, *set* $X_k = P^{(S_k)} w_k$ *and continue with Step 1.*

Step 3 *Let*

$$\theta_{k_l} \coloneqq \max_{\substack{i\in\{0,\dots,m_{k_l}-1\},\\ w_{k_l}^i - v_{k_l}^i > 0}} v_{k_l}^i / \left(v_{k_l}^i - w_{k_l}^i\right). \tag{5.9}$$

and set $z_{k_{l+1}} = \theta w_{k_l} + (1-\theta)\, v_{k_l}$.

Then, let $j_{k_l} \in \{0, \dots, m_{k_l} - 1\}$ *such that* $w_{k_l}^{j_{k_l}} = 0$, *and remove the* j_{k_l}*th component from* z_{k_l} *and* S_{k_l} *for weights and points to apply the weights on in the next sub-iteration:* $w_{k_{l+1}} = \begin{bmatrix} w_{k_l}^0 & \dots & w_{k_l}^{j_{k_l}-1} & w_{k_l}^{j_{k_l}+1} & \dots & w_{k_l}^{m_{k_l}}\end{bmatrix}^T$ *and* $S_{k_{l+1}} = \begin{bmatrix} S_{k_l}^0 & \dots & S_{k_l}^{j_{k_l}-1} & S_{k_l}^{j_{k_l}+1} & \dots & S_{k_l}^{m_{k_l}}\end{bmatrix}^T$. *Decrement* m_{k_l} *by one, set* l *to* $l+1$ *and continue with Step 2.*

[a] Here we used the choice as mentioned in Footnote *a* in Algorithm 5.2.

Compared to the original method as described in Section 4 of [40] though, the calculation of θ_{k_l} with (5.9) in Algorithm 5.3 differs to match the geometric variant of the same algorithm.

In [40], θ_{k_l} is calculated with the value

$$\min\Big\{1, \min_{\substack{i\in\{0,\dots,m_{k_l}-1\},\\ w^i_{k_l}-v^i_{k_l}>0}} w^i_{k_l}/\big(w^i_{k_l}-v^i_{k_l}\big)\Big\}. \tag{5.10}$$

The following example shows us how using this original formula (5.10) not only results in an algorithm whose interpretation does not match the geometrical interpretation, but even results in never-ending execution:

Example 5.2. Let $P = \{P_1 = (1,2), P_2 = (3,0), P_3 = (-4,0)\}$, a set of $m = 3$ points in $\mathbb{R}^2$, which we will now treat individually as vectors in $\mathbb{R}^2$. We wish to compute the minimum norm vector in conv P, and use Algorithm 5.3 for this. We consider two versions of this algorithm: one by using the version as exactly stated here, and one by using (5.10) in place for (5.9), as done in [40]. A visualisation of this problem can be found in Figure 5.3b.

In the initialisation phase, we calculate $J_0 = 1$, since the norm of the first point, $\sqrt{5}$, is strictly smaller than the norm of the other points, namely 3 and 4. Thus, we set $w_0 = \begin{bmatrix}1\end{bmatrix}^T$ and $S_0 = \begin{bmatrix}1\end{bmatrix}$, with $m_0 = 1$. Our starting point is $X_0 = \begin{bmatrix}1 & 2\end{bmatrix}^T$.

Then, in Step 1, we calculate $J_1 = 3$, corresponding to the value of j resulting in the smallest value for $X_1^T P^j$ – compare $\begin{bmatrix}2 & 0\end{bmatrix}\begin{bmatrix}1 & 3 & -4\\ 2 & 0 & 0\end{bmatrix} = \begin{bmatrix}5 & 3 & -4\end{bmatrix}$. From this calculation, we can directly derive that X_0 is not the minimum norm vector according to our stopping condition, and we continue with Step 2 with $w_{1_0} = \begin{bmatrix}1 & 0\end{bmatrix}^T$ and $S_{1_0} = \begin{bmatrix}1 & 3\end{bmatrix}^T$.

In Step 2, we look for the minimum norm vector in the affine hull of $\{P^1, P^3\}$, e.g. a point v_{1_0} that fulfils (5.8). A calculation reveals that this holds for $(\lambda, v)_{1_0} = \left(64/29, \begin{bmatrix}20/29 & 9/29\end{bmatrix}^T\right)$, and since this point v_{1_0} is in the relative interior of the convex hull of $\{P^1, P^3\}$, we continue with Step 1 in iteration $k = 1$, with $w_1 = \begin{bmatrix}20/29 & 9/29\end{bmatrix}^T$ and $S_1 = \begin{bmatrix}1 & 3\end{bmatrix}^T$.

Then, the termination criterion is still not fulfilled, and with $J_2 = 2$ is Step 2 entered, with $w_{1_0} = \begin{bmatrix}20/29 & 9/29 & 0\end{bmatrix}^T$ and $S_{1_0} = \begin{bmatrix}1 & 3 & 2\end{bmatrix}$. The minimiser in the affine hull of P is this time calculated, resulting in $v_{1_0} = \begin{bmatrix}0 & 3/7 & 4/7\end{bmatrix}$ – for the origin as resulting point when taking v_{1_0} as weights over P using S_{1_0} as guide for which point corresponds to which point.

However, since the first component of v_{1_0} is zero, v_{1_0} is not in the relative interior of the convex hull represented by P, and Step 3 is entered, with computation of θ_{1_0}. Computation using (5.9) results in a value of 0, such that $w_{1_1} = v_{1_0}$, and with the removal of the first zero component, the minimum norm vector in Step 2 in the affine

hull of $\{P^2, P^3\}$ is computed, for $v_{1_1} = \begin{bmatrix} 3/7 & 4/7 \end{bmatrix}$. This lies within the relative interior of the convex hull of $\{P^2, P^3\}$, and corresponding to the point $X_2 = \mathbf{0}$. Clearly this point is the minimum norm vector, and we return with this point.
However, should we have computed θ_{1_0} using the original formula from [40] with (5.10), then we would have gotten $\theta = 1$ instead, resulting in $z_{1_1} = w_{1_0}$, or simply no change in our current point. With the zero component in z_{1_1} corresponding to the weight of the most recently added point, we enter Step 2 with the same parameters as we did the first time this step was entered – termination never occurs.

The reason for failing with the choice of value for θ_{k_l} from [40] is that it in general not results in the nearest point to Y on $\overline{XY}$, using the terminology from the geometrical variant. For a derivation of the value of θ_{k_l} for which this is done, we are looking instead for a minimal value for $0 \leq \theta \leq 1$ such that for all i, $0 \leq \theta w_i + (1-\theta)\, v_i \leq 1$ holds, and the resulting point is a convex combination of points. Since

$$\sum_i \theta w_i + (1-\theta)\, v_i = \theta \sum_i w_i + (1-\theta) \sum_i v_i = \theta + 1 - \theta = 1,$$

the requirement of $\theta w_i + (1-\theta)\, v_i \geq 0$ is clearly enough for implying $\theta w_1 + (1-\theta)\, v_i \leq 1$ too, and we will only show for which values of θ the first requirement explicitly holds.

Rewriting this requirement to

$$\begin{aligned} 0 &\leq \theta w_i + (1-\theta)\, v_i \Leftrightarrow \\ 0 &\leq v_i + \theta\,(w_i - v_i) \Leftrightarrow \\ -v_i &\leq \theta\,(w_i - v_i) \Leftrightarrow \\ v_i &\geq \theta\,(v_i - w_i)\,, \end{aligned}$$

we consider the three different possibilities of the values of v_i and w_i compared to each other. If $v_i - w_i = 0$, then the requirement is clearly fulfilled for all values of θ. Now, assume $v_i - w_i > 0$ instead. Then, $v_i/(v_i - w_i) \geq \theta$ is a requirement. We have already $v_i \geq v_i - w_i \Leftrightarrow v_i/(v_i - w_i) \geq 1$, such that the requirement is fulfilled for $\theta \leq 1$, and thus no additional restrictions on $\theta \in [0, 1]$ are imposed. Otherwise, for $v_i - w_i < 0$, the requirement is $v_i/(v_i - w_i) \leq \theta$, or, in different terms, $\max\{v_i/(v_i - w_i) \colon v_i - w_i < 0 \Leftrightarrow w_i - v_i > 0\} \leq \theta$. However, also here $v_i \geq v_i - w_i \Leftrightarrow v_i/(v_i - w_i) \leq 1$ holds.

As assumption for calculating θ at all, there is an j such that $v_j < 0$. Thus, since $w_i \geq 0$ for all i, $w_j - v_j > 0$ and $v_j/(v_j - w_j) \geq 0$, from which it follows that $\max\{v_i/(v_i - w_i) \colon w_i - v_i > 0\} \geq 0$.

All in all, choosing $\theta = \max\{v_i/(v_i - w_i) \colon w_i - v_i > 0\} \in [0, 1]$ fulfils the requirements and is clearly the minimum value for which the requirements are fulfilled.

Chapter 6

Numerical Experiments

In the previous chapters, three derivative-free optimisation methods for unconstrained optimisation have been described: the Nelder–Mead method, the trust-region method and the discrete gradient method. In this chapter, we describe the results of performing experiments on these different methods and between variants of the same method. For the experiments, the methods are applied on several problems with in their description an objective function and starting point; these problems either originate from real-world applications, or are specifically constructed for method comparison ('academically').

We first compare different variants of methods inspired by the Nelder–Mead method and then make a broader comparison between different methods by applying them on twice continuously differentiable objective functions. Lastly, we consider functions that are not differentiable, to motivate the discrete gradient method.

6.1 Nelder–Mead method variants

In the case of Nelder–Mead inspired methods, we have two substantially differing methods: the original method, Algorithm 3.1, from [16] and the FDSS methods with proven convergence for a broad class of functions, Algorithm 3.2, from [38]. As methods, there is some freedom left for implementations to choose specific behaviour, allowing more variants that are all in the spirit of the method by Nelder and Mead. The Nelder–Mead method in mostly its original form is implemented in the industrially-proven MATLAB software as the `fminsearch` function according to [36], with the explicit acknowledgement in [37] that '[the] algorithm is not guaranteed to converge to a local minimum', despite the FDSS method in the same spirit having such guarantee. In this sub-section, we compare the different algorithms based on the Nelder–Mead method and discuss advantages of some over others.

Obtaining the initial simplex in the `fminsearch` implementation is done by providing a single point $y^0 \in \mathbb{R}^n$, and constructing different points $y^1, y^2, \ldots, y^n$ by letting y^i be y^0 with the ith component multiplied by 1.05, or the ith component set to 0.00025 if the ith

component turned out to be zero. Thus, the initial simplex consist of $n + 1$ points where n points differ in a single dimension from the starting point. In [38], it has been suggested to construct the initial simplex in such a way that is similar, but by adding 1 to the ith component of y^i instead.

Another degree is freedom has to do with the expansion point. In the Nelder–Mead method as described in [16] and as implemented in the MATLAB function `fminsearch`, the reflection point is accepted if the resulting function value is comparable to the other points that are not the best or the worst, while if the reflection point was better, the expansion point is computed, and the better of those two points is calculated. In the version described in [38] though, it can be chosen to always compute the expansion point to be more in line with the original method as described by Nelder and Mead themselves in [24]. The expansion point was there accepted if it provides enough decrease compared to the current best point, independent of the decrease of the reflection point. The FDSS method in [38] leaves enough freedom to match the behaviour in [16] though, to not always compute the expansion point.

Lastly, different termination criteria can be chosen, as discussed in Section 3.4, where we use the natural choice of a stopping criterion for the mode radical FDSS method.

With these options to choose from, we consider three different algorithm inspired by the Nelder–Mead method: the MATLAB `fminsearch` implementation based on the method described in [16], an implementation of the FDSS method with the above options aligned with `fminsearch` and the more radical version where those options are not aligned with `fminsearch`. Both latter implementations are together attached in Implementation A.1. Those three algorithms are then tested on a sample of unconstrained problems described in the CUTEst test set, [5], which describe an objective function to minimise and a (in this unconstrained case trivially) feasible initial solution to start the minimisation process with.

We first compare the `fminsearch` variant with the FDSS variant with degrees of freedom aligned up with the `fminsearch` variant on 15 different problems, for which the results are shown in Table 6.1. For this randomly-chosen set of 15 problems, we can see that both methods terminate after the same number of function evaluations with the same objective function value for lower-dimensional problems. For higher-dimensional problems, the FDSS variant performs notably worse than `fminsearch` in terms of the final function value; even with the latter restricted to the same number of function evaluations as the first, the FDSS method provides for the higher-dimensional problems a worse function value than `fminsearch` does, with the `GROWHTLS` on the intersection between lower-dimensional and higher-dimensional problems as exception, where FDSS ends up with a better function value.

We now compare `fminsearch` with the other FDSS method which is less aligned with choices made for the `fminsearch` implementation on the same problem set as for our previous comparison, with the results of the current comparison shown in Table 6.2. In case both methods finish successfully, for the lower-dimensional problems, it varies which method, `fminsearch` or the more radical FDSS, is better in terms of function value and number of

problem (n)	`fminsearch`		FDSS (`fminsearch`-style)		`fminsearch` (eval. limited to FDSS)	
	f^*	eval.	f^*	eval.	f^*	eval.
`PARKCH` (15)	$1.7648 \cdot 10^3$	4653	$2.1157 \cdot 10^3$	384	$2.0996 \cdot 10^3$	384
`STRATEC` (10)	$2.3246 \cdot 10^3$	1296	$2.7060 \cdot 10^3$	282	$2.4028 \cdot 10^3$	282
`COOLHANSLS` (9)	0.0638	583	$8.9812 \cdot 10^5$	325	454.4300	325
`BIGGS6` (6)	0.0057	803	0.2927	225	0.2614	225
`HIMMELBF` (4)	318.5717	459	$1.6543 \cdot 10^3$	165	359.1637	165
`BROWNDEN` (4)	$8.5822 \cdot 10^4$	333	$8.5822 \cdot 10^4$	322	$8.5822 \cdot 10^4$	323
`GROWTHLS` (3)	1.2189	306	1.2120	270	1.2190	271
`BARD` (3)	0.0082	226	0.0082	226	”	”
`ENGVAL2` (3)	$8.8115 \cdot 10^{-10}$	279	$8.8115 \cdot 10^{-10}$	279	”	”
`HELIX` (3)	$3.5759 \cdot 10^{-4}$	142	$3.5759 \cdot 10^{-4}$	142	”	”
`CUBE` (2)	$2.5263 \cdot 10^{-10}$	166	$2.5263 \cdot 10^{-10}$	166	”	”
`CLUSTERLS` (2)	$6.8693 \cdot 10^{-12}$	117	$6.8693 \cdot 10^{-12}$	117	”	”
`BRKMCC` (2)	0.1690	76	0.1690	76	”	”
`ZANGWIL2` (2)	-18.2000	67	-18.2000	67	”	”
`CLIFF` (2)	0.2007	54	0.2007	54	”	”

Table 6.1: Application of the `fminsearch` implementation of the Nelder–Mead method and an implementation of FDSS (from Implementation A.1) aligned with `fminsearch` on 15 functions from the CUTEst data set ([5]). Reported for both implementations is the optimal solution found after default termination and the number of function evaluations it took for this solution to find; furthermore, the optimal solution found by `fminsearch` by restricting the number of evaluation to match that with FDSS is reported too.

evaluations. However, for the same problems as where the `fminsearch`-like FDSS provided worse results than `fminsearch`, the more radical version of FDSS doesn't provide better results either.

This time though, the method explicitly fails before successfully terminating, as the limited working precision of the device (using double-precision IEEE Standard 754 numbers) results in basic assumptions not holding anymore that prevent the algorithm from continuing in a proper way. One mode of failure is that of the normalised volume changing under a shrink step, when it should be unchanged. With the shrink step being a 'last resort' step that should always result in a valid configuration, the normalised volume can in that case become after a shrink step be under the threshold, which directly affects further steps taken, and could result in only shrink steps being taken if the normalised volume cannot be enough increased directly by other steps. Another mode of failure is when the affect of $\alpha(\Delta_{k_l})$ in (3.8) or (3.10) is too small, e.g. if $f\left(y_{k_l}^{m_{k_l}-1}\right) - \alpha(\Delta_{k_l}) = f\left(y_{k_l}^{m_{k_l}-1}\right)$, while $\alpha(\Delta_{k_l}) > 0$ should hold. A point that has been thought to provide sufficient decrease is then accepted when it not necessarily does provide sufficient decrease, which can result in the same points being brought in time after time and the method getting stuck in an infinite loop.

For both variants of FDSS, we can conclude that for higher-dimensional problems, FDSS

problem (n)	`fminsearch`		FDSS (radical)		`fminsearch` (eval. limited to FDSS)	
	f^*	eval.	f^*	eval.	f^*	eval.
`PARKCH` (15)	$1.7648 \cdot 10^3$	4653	$2.0810 \cdot 10^3$	1005*	$2.0837 \cdot 10^3$	1005
`STRATEC` (10)	$2.3246 \cdot 10^3$	1296	$2.5853 \cdot 10^3$	699*	$2.3502 \cdot 10^3$	700
`COOLHANSLS` (9)	0.0638	583	0.3493	803*	-	-
`BIGGS6` (6)	0.0057	803	0.2614	491*	0.0573	491
`HIMMELBF` (4)	318.5717	459	$1.3798 \cdot 10^3$	345*	318.5717	345
`BROWNDEN` (4)	$8.5822 \cdot 10^4$	333	$8.5822 \cdot 10^4$	393	-	-
`GROWTHLS` (3)	1.2189	306	12.8952	448†	-	-
`BARD` (3)	0.0082	226	0.0082	166	0.0082	167
`ENGVAL2` (3)	$8.8115 \cdot 10^{-10}$	279	$1.9819 \cdot 10^{-8}$	164	4.3897	164
`HELIX` (3)	$3.5759 \cdot 10^{-4}$	142	$1.4109 \cdot 10^{-7}$	142	”	”
`CUBE` (2)	$2.5263 \cdot 10^{-10}$	166	$1.2698 \cdot 10^{-8}$	157	$3.4292 \cdot 10^{-9}$	158
`CLUSTERLS` (2)	$6.8693 \cdot 10^{-12}$	117	$1.6480 \cdot 10^{-10}$	63	0.0013	64
`BRKMCC` (2)	0.1690	76	0.1690	57	0.1690	58
`ZANGWIL2` (2)	−18.2000	67	−18.2000	53	−18.2000	54
`CLIFF` (2)	0.2007	54	0.1998	107	-	-

* Termination because of the normalised volume.
† Termination because of the fortified-descent criterion vanishing.

Table 6.2: Application of the `fminsearch` implementation of the Nelder–Mead method and an implementation of FDSS (from Implementation A.1) not aligned with `fminsearch` on 15 functions from the CUTEst data set ([5]). Reported for both implementations is the optimal solution found after default termination and the number of function evaluations it took for this solution to find; furthermore, the optimal solution found by `fminsearch` by restricting the number of evaluation to match that with FDSS is reported too, if applicable.

performs worse than `fminsearch`, while for smaller-dimensional problems, it differs, with `fminsearch` not providing substantially worse results than FDSS in those cases they are not the same. Thus, although the FDSS has theoretical convergence, the more classical Nelder–Mead method through `fminsearch` provides better results for a randomly-chosen set of twice continuously differentiable functions and is less affected by practical limitations on number precision, which motivates the less widespread usage of FDSS over the more classical Nelder–Mead method, and the motivation for the MATLAB software having implemented only the original Nelder–Mead method.

6.2 Inter-method comparison

After having compared different versions of Nelder–Mead methods, in this section, we compare the different methods seen so far: besides (one of the versions of the) Nelder–Mead method, we consider the trust-region method – both a derivative-free version and a version using derivatives – and the discrete gradient method as well for our comparison. For being able to take the trust-region method using derivatives into account, we consider solely

problem (n)	PDFO		`fminunc`		DGM	
	f^*	eval.	f^*	eval.	f^*	eval.
`PARKCH` (15)	$1.6237 \cdot 10^3$	10 000	$1.6237 \cdot 10^3$	25	$2.0602 \cdot 10^3$	9492.5
`STRATEC` (10)	$2.2218 \cdot 10^3$	10 000	$2.2123 \cdot 10^3$	119	-	-*
`COOLHANSLS` (9)	0.0028	10 000	$5.5500 \cdot 10^{-4}$	816	0.0108	10 007
`BIGGS6` (6)	0	687	0.0057	42	$9.0000 \cdot 10^{-6}$	9597.5
`HIMMELBF` (4)	318.5717	292	318.7808	202	319.6807	5420.5
`BROWNDEN` (4)	$8.5822 \cdot 10^4$	100	$8.5822 \cdot 10^4$	12	$8.5822 \cdot 10^4$	2502
`GROWTHLS` (3)	1.0040	1526	1.1034	10 001	$2.7929 \cdot 10^3$	52.5
`BARD` (3)	0.0082	84	0.0086	193	0.0083	3943
`ENGVAL2` (3)	0	135	$6.7000 \cdot 10^{-5}$	109	0.0020	3000
`HELIX` (3)	0	55	$9.5000 \cdot 10^{-5}$	754	$2.2000 \cdot 10^{-5}$	4019
`CUBE` (2)	0	111	0	32	0.0017	2170
`CLUSTERLS` (2)	0	56	0	14	0	251.5
`BRKMCC` (2)	0.1690	20	0.1690	4	0.1690	392
`ZANGWIL2` (2)	−18.2000	17	−18.2000	2	−18.2000	243
`CLIFF` (2)	0.1998	100	0.1998	27	$1.3092 \cdot 10^3$	2018.5

* Aborted execution since invalid function values were encountered during application of the method on the unconstrained problem.

Table 6.3: Application of the PDFO implementation of the derivative-free trust-region method, the `fminunc` implementation of the derivative-using trust-region method and an implementation of DGM (from Implementation A.3) on 15 functions from the CUTEst data set ([5]). In the context of PDFO are `COOLHANSLS` and higher-dimensional problems solved using NEWUOA while smaller-dimensional problems were solved using UOBYQA. Reported for both implementations is the optimal solution found after default termination and the number of function evaluations it took for this solution to find; once the number of function evaluations exceeds 10000, the method is terminated too. Since the discrete gradient method is a non-deterministic method, the reported values for this method are the mean of a sample of four runs.

objective functions that are twice continuously differentiable.

For the trust-region method using derivatives, we consider two variants made by Powell: UOBYQA – 'unconstrained optimisation by quadratic approximation' as described in [27] – for lower-dimensional problems and NEWUOA – the newer version as described in [27] based on UOBYQA – for higher-dimensional problems, using the MATLAB interface in [29] to the otherwise mostly original implementation. An important difference between the two methods is that, for some n-dimensional problem, UOBYQA uses quadratic interpolation and requires $\frac{1}{2}(n+1)(n+2)$ points within the trust region of the model, while NEWUOA can work with an underdetermined model, with $2n+1$ points as common number. The difference in the number of points is especially noticeable for higher-dimensional problems, which NEWUOA preferred for such problems.

For the trust-region method that does use derivative information, the industrially-proven `fmincon` as described in [37] is used. Here, over the course of the application of this method on twice continuously differentiable functions, in addition to the objective function values, the gradient and Hessian of the objective function was made available.

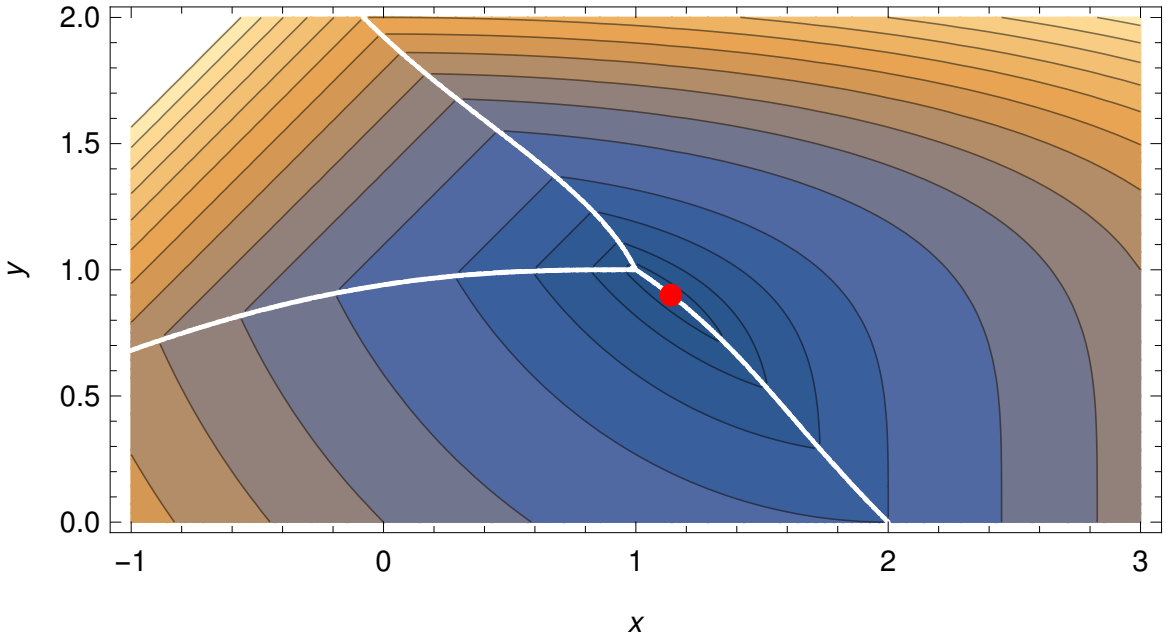


Figure 6.1: Contour plot of $(x, y) \mapsto \max\left\{x^2 + y^4, (2-x)^2 + (2-y)^2, 2\mathrm{e}^{-x+y}\right\}$, with the red dot indicating the (global) minimum of the function.

Lastly, for the discrete gradient method, we use our implementation from Implementation A.3. With the same problems as used in Table 6.1 and Table 6.2, Table 6.3 then shows the result of applying those three methods. We can see in those three tables that, under the default termination criteria, most of the methods are comparable in terms of the function value found for the same problem, with the same objective function and starting point. Moreover, in those cases, the derivative-free trust-region method is taking mostly less function evaluations than the Nelder–Mead method and the discrete gradient method do. When comparing both trust-region methods, one using derivatives and one not, the trust-region method not using derivative information takes mostly less function evaluations for a comparable result, providing a slight favour for the methods using derivative information when it is available in terms of execution time.

6.3 Non-differentiable functions

In the previous section, different methods were considered, including the discrete gradient method. Looking at Table 6.3 on twice continuously differentiable functions though, we could conclude that the discrete gradient method needs considerably more function evaluations than the other methods considered, which could raise the question on the practical purpose of the discrete gradient method.

An example of the problem of minimising a two-dimensional function that can be written as the maximum of quadratic functions, can be found in [8], with a visualisation of the contour

plot in the area of interest in Figure 6.1. Along the places marked white in this figure is the function non-differentiable, with a (global) minimum at $(x^*, y^*) = (1.13904, 0.89956)$ with a corresponding function value of 1.95222. Applying with starting point $(2, 2)$ `fminseearch` and the discrete gradient method (with four different seeds for the randomness) from Implementation A.3 results in this value being reached, while the PDFO trust-region method fails at $(1, 1)$, on the intersection between the three functions, with the minimum not being reached.

Appendix A

Algorithm Implementations

What follows is a listing of implementations of some reviewed algorithms, namely of the FDSS method and the discrete gradient method, including a method for finding the nearest point a polytope. The source code of each implementation is embedded in the digital PDF version of this document under the file name specified in the caption of each algorithm.

Listing A.1: MATLAB R2022b implementation of Algorithm 3.2 (`fdss.m`).

```
function [x,fval,exitflag,output] = fdss(fun,x0,options, ...
    showgraph,fminsearchstyle)
%FDSS Multidimensional unconstrained nonlinear minimisation (Fortified-
%   Descent Simplicial Search)

%   Implementation based on algorithm inspired by
%    Paul Tseng. Fortified-descent simplicial search method: A general
%    approach. SIAM J. Optim., 10(1):269-288, 1999.
%    doi:10.1137/S1052623495282857.

    alpha = @(t)(1e-5)*min(0.5*(t^2),t);
    beta = @(t)(1e6)*(t^2);
    nu = 1e-5;
    rho = 1;
    chi = 2;
    gamma = 1/2;
    sigma = 1/2;
    thetar = 0.01;
    epsilon = 1e-3;

    n = length(x0);

    if nargin < 5
        fminsearchstyle = true;
    end
    if nargin < 4
        showgraph = false;
    end
    defaultopt = optimset( ...
        'Display','notify', ...
        'MaxFunEvals',200*n, ...
        'MaxIter',200*n, ...
        'TolFun', 1e-4, ...
        'TolX', 1e-4 ...
        );
    if nargin < 3
        options = defaultopt;
    end

    Display = optimget(options,'Display',defaultopt,'fast');
    MaxFunEvals = optimget(options,'MaxFunEvals',defaultopt,'fast');
```

```
    MaxIter = optimget(options,'MaxIter',defaultopt,'fast');
    TolFun = optimget(options,'TolFun',defaultopt,'fast');
    TolX = optimget(options,'TolX',defaultopt,'fast');

    iterations = 0;
    funcCount = 0;
    function y = evalcount(x)
        funcCount = funcCount+1;
        y = fun(x);
    end

    function delta = diam(xs)
        delta=0;
        for i = 1:n+1
            delta=max(delta,max(sqrt(sum((xs(:,i) - xs).^2,2))));
        end
    end

    function [delta,von] = simplex(xs)
        delta = diam(xs);
        von = abs(det(xs(:,2:end) - xs(:,1)))/delta^n;
    end

    if fminsearchstyle
        xs = x0;
        usual_delta = 0.05;
        zero_term_delta = 0.00025;
        for j = 1:n
            newx = x0;
            if newx(j) ~= 0
                newx(j) = (1 + usual_delta)*newx(j);
            else
                newx(j) = zero_term_delta;
            end
            xs(:,j+1) = newx;
        end
    else
        xs = [x0 x0+eye(n)];
    end

    ys = cellfun(@evalcount,num2cell(xs.',2));

    if strcmp(Display,'iter')
        fprintf('\n')
        fprintf([' Iteration   Func-count     min f(x)         ' ...
            'Procedure\n']);
        fprintf(' %5.0f        %5.0f     %12.6g         %s\n', ...
            0, 1, ys(1), '')
    end

    [ys,ysI] = sort(ys);
    xs = xs(:,ysI);

    if n == 2 && showgraph
        figure('Name','FDSS Progress','NumberTitle','off');
        xlabel('x');
        ylabel('y');
        hold on
        axis equal
        title('Simplices over time')
    end

    [delta,von] = simplex(xs);
    nu = min(nu,von);

    k = 0;
    operation = 'initial simplex';
    unfinished = 1;
    while unfinished
        l = 0;
        lxs = xs;
        lys = ys;
```

```
        while 1
            iterations = iterations + 1;

            if iterations >= MaxIter
                output.message = ['Exiting: Maximum number ' ...
                    'of iterations has been exceeded' newline ...
                    '          - increase MaxIter option.' newline ...
                    '          Current function value: ' ...
                    num2str(lys(1),'%12.6f') ' ' newline ' '];
                exitflag = 0;
                unfinished = 0;
                break
            end

            if funcCount >= MaxFunEvals
                output.message = ['Exiting: Maximum number ' ...
                    'of function evaluations has been exceeded' ...
                    newline '          - increase MaxFunEvals ' ...
                    'option.' newline '          Current function ' ...
                    'value: ' num2str(lys(1),'%12.6f') ' ' newline ...
                    ' '];
                exitflag = 0;
                unfinished = 0;
                break
            end

            if fminsearchstyle ...
                    && max(max(abs(lxs(:,2:end)-lxs(:,1)))) ...
                    < max(TolX,10*eps(max(lxs(:,1)))) ...
                    && max(abs(lys(2:end)-lys(1))) ...
                    < max(TolFun,10*eps(lys(1)))
                output.message = ['Optimisation terminated:' ...
                    newline ' the current x satisfies the ' ...
                    'termination criteria using OPTIONS.TolX of ' ...
                    num2str(TolX,'%e') ' ' newline ...
                    ' and F(X) satisfies the convergence ' ...
                    'criteria using OPTIONS.TolFun of ' ...
                    num2str(TolFun,'%e') ' ' newline];
                exitflag = 1;
                unfinished = 0;
                break
            end

            % This condition can be made true with default
            % parameters from FDSS paper and initial simplex
            % initialisation and PARKCH problem from the CUTEst
            % dataset.
            if von < nu
                output.message = ['Return from fdss because the ' ...
                    'normalised volume is too small after shrink ' ...
                    'operation.' newline ' '];
                exitflag = 0;
                unfinished = 0;
                break
            end

            if n == 2 && showgraph
                DTo = delaunayTriangulation(lxs(1,:).',lxs(2,:).');
                Co = convexHull(DTo);
                plot(DTo.Points(:,1),DTo.Points(:,2),'.', ...
                    'MarkerSize',10)
                plot(DTo.Points(Co,1),DTo.Points(Co,2),'r')
            end

            if ~fminsearchstyle && delta <= epsilon ...
                && norm((lys(2:end,:)-lys(1,:)) ...
                ./ sqrt(sum((lxs(:,2:end) - lxs(:,1)).^2,2))) ...
                < epsilon
                output.message = ['Optimisation terminated:' ...
                    newline ' the norm current gradient-like ' ...
                    'matrix satisfies the termination criterium ' ...
                    'of ' num2str(epsilon,'%e') ' ' newline];
```

```
                exitflag = 1;
                unfinished = 0;
                break
            end

            if strcmp(Display,'iter')
                fprintf([' %5.0f        %5.0f     %12.6g         ' ...
                    '%s\n'],iterations,funcCount,lys(1),operation)
            end

            m = find(lys-ys > 0,1);
            if size(m,1) == 0 || m(1,1) == n+1
                m = n;
            end

            xm = sum(lxs(:,1:m),2)/m;
            ym = sum(lys(1:m))/m;

            xsr = xm + rho*(xm - lxs(:,m+1:end));
            ysr = cellfun(@evalcount,num2cell(xsr.',2));

            lxsr = [lxs(:,1:m) xsr];
            lysr = [lys(1:m);ysr];
            [deltar,vonr] = simplex(lxsr);

            % This condition can be made through with default
            % parameters from FDSS paper, Matlab's fminsearch R2022b
            % initial simplex initialisation and PARKCH problem from
            % the CUTEst dataset.
            if min(ysr) <= lys(m) - alpha(delta) ...
                    && min(ysr) >= lys(m)
                output.message = ['Return from fdss because the ' ...
                    'fortified-descent criterium vanishes in ' ...
                    'the current precision.' newline ' '];
                exitflag = 0;
                unfinished = 0;
                break
            end

            if vonr >= nu && min(ysr) <= lys(m) - alpha(delta) ...
                    && min(ysr) <= lys(m) - thetar*(lys(end)-ym) ...
                    + beta(delta)
                if fminsearchstyle && lys(1) <= min(ysr)
                    operation = 'reflect';
                    xs = lxsr;
                    ys = lysr;
                    delta = deltar;
                    von = vonr;
                else
                    xse = xm + chi*rho*(xm - lxs(:,m+1:end));
                    yse = cellfun(@evalcount,num2cell(xse.',2));

                    lxse = [lxs(:,1:m) xse];
                    lyse = [lys(1:m);yse];
                    [deltae,vone] = simplex(lxse);

                    if vone >= nu ...
                            && ((fminsearchstyle ...
                            && min(yse) <= min(ysr)) ...
                            || (~fminsearchstyle ...
                            && min(yse) <= lys(m) ...
                            - alpha(delta) ...
                            && min(yse) <= lys(m) ...
                            - thetar*(lys(end)-ym) + beta(delta)))
                        operation = 'expand';
                        xs = lxse;
                        ys = lyse;
                        delta = deltae;
                        von = vone;
                    else
                        operation = 'reflect';
                        xs = lxsr;
```

```
                        ys = lysr;
                        delta = deltar;
                        von = vonr;
                    end
                end
                [ys,ysI] = sort(ys);
                xs = xs(:,ysI);
                k = k + 1;
                break
            else
                if min(ysr) < lys(m+1)
                    operation = 'contract outside';
                    xsc = xm + gamma*rho*(xm - lxs(:,m+1:end));
                else
                    operation = 'contract inside';
                    xsc = xm - gamma*(xm - lxs(:,m+1:end));
                end
                ysc = cellfun(@evalcount,num2cell(xsc.',2));

                lxsc = [lxs(:,1:m) xsc];
                lysc = [lys(1:m);ysc];
                [deltac,vonc] = simplex(lxsc);

                [lysc,lyscI] = sort(lysc);
                lxsc = lxsc(:,lyscI);

                assert(isequal(size(lxsc),size(lxs)))

                if vonc >= nu && lysc(m+1) <= ys(m+1) ...
                        && sum(lysc(1:m+1)) <= sum(ys(1:m+1)) ...
                        - alpha(delta)
                    xs = lxsc;
                    ys = lysc;
                    delta = deltac;
                    von = vonc;
                    k = k + 1;
                    break
                else
                    operation = 'shrink';
                    xss = lxs(:,1) + sigma*(lxs(:,2:end) - lxs(:,1));
                    yss = cellfun(@evalcount,num2cell(xss.',2));

                    lxss = [lxs(:,1) xss];
                    lyss = [lys(1);yss];

                    [lyss,lyssI] = sort(lyss);
                    lxss = lxss(:,lyssI);

                    [deltas,vons] = simplex(lxss);

                    if lyss(1) <= ys(1)-alpha(delta)
                        xs = lxss;
                        ys = lyss;
                        delta = deltas;
                        von = vons;
                        k = k + 1;
                        break
                    else
                        lxs = lxss;
                        lys = lyss;
                        delta = deltas;
                        von = vons;
                        l = l + 1;
                        continue
                    end
                end
            end
        end
    end

    [ymin,yminI] = min(lys);
    xmin = lxs(:,yminI);
```

```
330     x = xmin;
331     fval = ymin;
332
333     if strcmp(Display,'iter')
334         fprintf(' %5.0f        %5.0f     %12.6g         %s\n', ...
335             iterations,funcCount,fval,operation)
336     end
337
338     if strcmp(Display,'iter') || strcmp(Display,'final')
339         fprintf('\n')
340         fprintf(output.message)
341         fprintf('\n')
342     end
343
344     if n == 2 && showgraph
345         xl = xlim;
346         yl = ylim;
347         [X,Y] = meshgrid(xl(1):((xl(2) - xl(1))/500):xl(2), ...
348             yl(1):((yl(2) - yl(1))/500):yl(2));
349         C = arrayfun(@(varargin)fun([varargin{:}]).^(0.1),X,Y);
350         im = imagesc(xl,yl,C);
351         uistack(im,'bottom');
352
353 %         set(gcf,'Units','inches');
354 %         screenposition = get(gcf,'Position');
355 %         set(gcf,...
356 %             'PaperPosition',[0 0 screenposition(3:4)],...
357 %             'PaperSize',[screenposition(3:4)]);
358 %         print -dpdf -vector filename
359     end
360
361     output.iterations = iterations;
362     output.funcCount = funcCount;
363     output.algorithm = 'Fortified-Descent Simplicial Search Method';
364 end
```

Listing A.2: MATLAB R2022b implementation of Algorithm 5.3 (`wolfe.m`).

```
1  function [X,error] = wolfe(P,acc)
2  %WOLFE Minimum-norm point finding in convex set.
3  %   X  = WOLFE(P) searches for the minimum-norm point in the convex set
4  %   described by the points in matrix P, for each point being a column
5  %   vector.
6  %
7  %   [X,error] = wolfe(...) also reports if the search was aborted
8  %   because of loss of numerical precision, rather than the optimality
9  %   conditions being fulfilled.
10
11 %   Implementation based on algorithm ('alternative D') inspired by
12 %    Philip Wolfe. Finding the nearest point in a polytope. Math.
13 %    Program., 11:128-149, December 1976. doi:10.1007/BF01580381.
14 %   with modifications as mentioned in master's thesis of Pim Heeman.
15
16
17     if nargin < 2
18         Z1 = 1e-12;
19     else
20         [maxnorm,maxnormI] = max(vecnorm(P));
21         if maxnorm < acc
22             X = P(:,maxnormI);
23             error = false;
24             return
25         end
26         Z1 = (acc/(2*max(vecnorm(P))))^2;
27     end
28
29     Z3 = 1e-10;
30     Z2 = 1e-16;
31
32     warnings = warning('error','MATLAB:illConditionedMatrix');
33     warning('error','MATLAB:singularMatrix');
```

```
    function finalise
        warning(warnings)
    end

    % If there's only a single element, we immediately return.
    % This way, we never get into problems of numerical inprecision,
    % at least.
    if size(P,2) == 1
        X = P(:,1);
        error = false;
        return
    end

    % Initialisation
    [~,J] = min(vecnorm(P));
    S = J;
    w = 1;
    R = sqrt(1+norm(P(:,J))^2);

    while 1
        % Step 1
        X = P(:,S)*w;
        [~,J] = min(X.'*P);

        threshold = Z1*max(norm(P(:,J)),max(vecnorm(P(:,S))))^2;
        if X.'*P(:,J) >= X.'*X - threshold
            error = false;
            finalise
            return
        end

        if ismember(J,S)
            error = true;
            finalise
            return
        end
        assert(~ismember(J,S))

        r = linsolve(R.',1+P(:,S).'*P(:,J));
        rho = sqrt(1+P(:,J).'*P(:,J)-r.'*r);
        R(:,end+1) = r; %#ok<*AGROW>
        R(end+1,:) = 0;
        R(end,end) = rho;

        S(end+1,1) = J;
        w(end+1,1) = 0;

        while 1
            % Step 2
            try
                ubar = linsolve(R.',ones(size(R,2),1));
            catch ME
                if strcmp(ME.identifier, ...
                        'MATLAB:illConditionedMatrix') ...
                    || strcmp(ME.identifier, ...
                    'MATLAB:singularMatrix')
                    error = true;
                    finalise
                    return
                end
                rethrow(ME)
            end

            u = linsolve(R,ubar);
            v = u/sum(u);

            if all(v > Z2*ones(size(v)))
                w = v;
                break
            end

            % Step 3
```

```
            POS = find(w-v > -Z3);
            theta = max((v(POS))./(v(POS)-w(POS)));
            % Below is the expression as found in the original
            % paper for computing theta, which could lead to the wrong
            % results.
            %
            % For instance, with input of
            %  [1 3 -4; 2 0 0]
            % an infinite loop is entered.
            % theta = min(1,min((w(POS))./(w(POS)-v(POS))));
            w = theta*w+(1-theta)*v;

            % This won't change the point. In other words, taking
            % another point into account that made the optimality
            % conditions not being fulfilled didn't made it any better,
            % and the point was optimal, after all, in our working
            % precision.
            %
            % This condition is triggered by
            %  [-0.024264412760097 0.00818576756098157 ...
            %     0.00308361747359006 0.00717008694718105 ...
            %     ; 0.0144869387432543 -0.0212952895278437 ...
            %     -0.0039593859212008 -0.00396459323288156]
            % with the default input parameters.
            if theta == 1
                error = true;
                finalise
                return
            end

            w(w <= Z3) = 0;
            I = find(w == 0,1);
            w(I) = [];
            S(I) = [];
            R(:,I) = [];

            while I <= size(R,2)
                a=R(I,I);
                b=R(I+1,I);
                c=sqrt(a^2+b^2);

                assert(c ~= 0)

                RI = R(I,:);
                RI1 = R(I+1,:);
                R(I,:) = (a*RI+b*RI1)/c;
                R(I+1,:) = (-b*RI+a*RI1)/c;
                I = I+1;
            end
            if ~isempty(I)
                R(end,:) = [];
            end
        end
    end
end
```

Listing A.3: MATLAB R2022b implementation of Algorithm 5.1 (`dgm.m`).

```
function [bestx,bestfval,exitflag,output] = dgm(fun,x0,options, ...
    showgraph)
%   Implementation based on algorithm inspired by
%    A. M. Bagirov, B. Karasoezen, and M. Sezer. Discrete gradient
%    method: Derivative-free method for nonsmooth optimization.
%    J. Optim. Theory. Appl., 137:317-334, 2008.
%    doi:10.1007/s10957-007-9335-5.

    lambda0 = 1;
    beta = 5e-1;
    c = 2e-01;
    c2 = 1e-04;
    alpha = 0.01;
```

```
    delta0 = 1e-7;
    betadelta = 0.9;
    epsilon = lambda0*1e-4;

    n = length(x0);

    if nargin < 4
        showgraph = false;
    end
    defaultopt = optimset( ...
        'Display','notify', ...
        'MaxFunEvals',200*n, ...
        'MaxIter',200*n ...
        );
    if nargin < 3
        options = defaultopt;
    end

    Display = optimget(options,'Display',defaultopt,'fast');
    MaxFunEvals = optimget(options,'MaxFunEvals',defaultopt,'fast');
    MaxIter = optimget(options,'MaxIter',defaultopt,'fast');

    iterations = 0;
    funcCount = 0;
    function fval = evalcount(x)
        funcCount = funcCount+1;
        fval = fun(x);

        if fval < bestfval
            bestfval = fval;
            bestx = x;
        end
    end

    function v = gamma(i,d,e,x,lambda,alpha,fx)
        xs = zeros(n,n+1);
        xs(:,1) = x+lambda*d;
        if n == 2 && showgraph
            plot(xs(1,1),xs(2,1),'g*')
        end
        for j = 1:n
            xs(:,j+1) = xs(:,j);
            xs(j,j+1) = xs(j,j+1)+lambda*(alpha^j)*e(j);
        end
        fxs = cellfun(@evalcount,mat2cell(xs,n,ones(1,n+1)));

        v = zeros(n,1);
        for j = 1:n
            if j == i
                continue
            end
            v(j) = (fxs(j+1)-fxs(j))/(lambda*(alpha^j)*e(j));
        end
        v(i) = (fxs(1)-fx-lambda*v.'*d)/(lambda*d(i));
    end

    x = x0;
    fval = fun(x);
    funcCount = funcCount+1;
    bestx = x;
    bestfval = fval;

    if strcmp(Display,'iter')
        fprintf('\n')
        fprintf([' Iteration   Func-count     min f(x)         ' ...
            'Procedure\n']);
        fprintf(' %5.0f        %5.0f     %12.6g         %s\n', ...
            0, 1, bestfval, '')
    end

    if n == 2 && showgraph
        figure('Name','DGM Progress','NumberTitle','off');
```

```
        xlabel('x');
        ylabel('y');
        hold on
        axis equal
        title('Base points over time')

        points = x0;
    end

    lambdainput = lambda0;
    delta = delta0;

    operation = 'initial point';
    unfinished = 1;
    while unfinished
        d = rand(n,1) - 0.5;
        d = d / norm(d);
        e = 2*(randi(2,n,1)-1.5);
        [~,i] = max(abs(d));
        lambdainput = beta*lambdainput;
        lambda = lambdainput^1.4;
        delta = betadelta*delta;

        if lambda < epsilon
            output.message = ['Optimisation terminated:' ...
                newline ' the lambda value satisfies the ' ...
                'termination criterium of ' ...
                num2str(epsilon,'%e') ' ' newline];
            exitflag = 1;
            break
        end

        if ~(funcCount >= MaxFunEvals || (iterations+1) >= MaxIter)
            v = gamma(i,d,e,x,lambda,alpha,fval);
            vbar = v;
        end

        while 1
            iterations = iterations+1;

            if iterations >= MaxIter
                output.message = ['Exiting: Maximum number ' ...
                    'of iterations has been exceeded' newline ...
                    '         - increase MaxIter option.' newline ...
                    '         Current function value: ' ...
                    num2str(bestfval,'%12.6f') ' ' newline ' '];
                exitflag = 0;
                unfinished = 0;
                break
            end

            if funcCount >= MaxFunEvals
                output.message = ['Exiting: Maximum number ' ...
                    'of function evaluations has been exceeded' ...
                    newline '         - increase MaxFunEvals ' ...
                    'option.' newline '         Current function ' ...
                    'value: ' num2str(bestfval,'%12.6f') ' ' ...
                    newline ' '];
                exitflag = 0;
                unfinished = 0;
                break
            end

            if strcmp(Display,'iter')
                fprintf([' %5.0f        %5.0f     %12.6g         ' ...
                    '%s\n'],iterations,funcCount,bestfval,operation)
            end

            if n == 2 && showgraph
                plot(x(1),x(2),'ro')
                points(:,end+1) = x; %#ok<AGROW>
            end
```

```
158
159             [vmin,~] = wolfe(vbar);
160
161             normvmin = norm(vmin);
162             if normvmin <= delta
163                 break
164             end
165
166             d = -vmin/norm(vmin);
167             xlinelambda = x+lambda*d;
168             fvallinelambda = evalcount(x+lambda*d);
169             if fval-fvallinelambda >= c*lambda*normvmin
170                 step = normvmin;
171                 while 1
172                     if step <= lambda
173                         x = xlinelambda;
174                         fval = fvallinelambda;
175                         break
176                     end
177                     xline = x+step*d;
178                     fvalline = evalcount(xline);
179                     if fval-fvalline >= c2*step*normvmin
180                         x = xline;
181                         fval = fvalline;
182                         break
183                     end
184                     step = 0.5*step;
185                 end
186
187                 vbar = [];
188                 d = rand(n,1) - 0.5;
189                 operation = 'serious step';
190             else
191                 operation = 'null step';
192             end
193
194             [~,i] = max(abs(d));
195             v = gamma(i,d,e,x,lambda,alpha,fval);
196             if ~isempty(vbar) && isequal(vbar(:,end),v)
197                 break
198             end
199             vbar(:,end+1) = v; %#ok<AGROW>
200         end
201     end
202
203     if n == 2 && showgraph
204         plot(points(1,:),points(2,:),'r--')
205     end
206
207     if strcmp(Display,'iter') && exitflag ~= 1
208         fprintf(' %5.0f        %5.0f     %12.6g         %s\n', ...
209             iterations,funcCount,bestfval,operation)
210     end
211
212     if strcmp(Display,'iter') || strcmp(Display,'final')
213         fprintf('\n')
214         fprintf(output.message)
215         fprintf('\n')
216     end
217
218     if n == 2 && showgraph
219         xl = xlim;
220         yl = ylim;
221         [X,Y] = meshgrid(xl(1):((xl(2) - xl(1))/500):xl(2), ...
222             yl(1):((yl(2) - yl(1))/500):yl(2));
223         C = arrayfun(@(varargin)fun([varargin{:}]).^(0.1),X,Y);
224         im = imagesc(xl,yl,C);
225         uistack(im,'bottom');
226
227 %         set(gcf,'Units','inches');
228 %         screenposition = get(gcf,'Position');
229 %         set(gcf,...
```

```
230 %                 'PaperPosition',[0 0 screenposition(3:4)],...
231 %                 'PaperSize',[screenposition(3:4)]);
232 %             print -dpdf -vector filename
233       end
234
235       output.iterations = iterations;
236       output.funcCount = funcCount;
237       output.algorithm = 'Discrete Gradient Method';
238   end
```

Symbols

$A \oplus B$	Minkowski sum; element-wise addition of two sets creating a new one ($A \oplus B = \{a + b \colon a \in A, b \in B\}$)
$\operatorname{aff} A$	affine hull of set A
$\nabla f(x)$	gradient of a function f
$\nabla^2 f(x)$	Hessian of a function f
$B(x; r)$	closed ball around $x \in \mathbb{R}^n$ of radius $r > 0$: $B(x; r) = \{y \in \mathbb{R}^n \colon \|x - y\| \leq r\}$
$\mathcal{C}^k(S, S')$	set of k-times continuously differentiable functions $f \colon S \to S'$ (parameter sometimes omitted when clear from context)
$\operatorname{conv} A$	convex hull of set A
$\partial f(x)$	subdifferential of function f
$\partial_c f(x)$	subdifferential of a convex function f
$\mathbf{e}$	vector with each element set to 1
$\mathbf{e}_j$	vector with the jth element set to 1 and the other elements being set to 0
$f(x)$	objective function $f \colon \mathbb{R}^n \to \mathbb{R}$
$f'(x; d)$	one-sided directional derivative of a function f
$L(x_0)$	sublevel set: $L(x_0) \coloneqq \{x \in \mathbb{R}^n \colon f(x) \leq f(x_0)\}$
n	dimension of optimisation problem
$\mathcal{P}_n^d$	linear space of all polynomial functions on $\mathbb{R}^n$ of a degree up to d
$\operatorname{ri} A$	relative interior of set A
S^n	unit sphere in the Euclidean $n + 1$-dimensional vector space: $S^n = \left\{e \in \mathbb{R}^{n+1} \colon \|e\| = 1\right\}$
x_k	iteration point of iteration k of an iterative optimisation method

Bibliography

[1] Charles Audet and Warren Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Cham, 2017.

[2] A. M. Bagirov, B. Karasözen and M. Sezer. Discrete gradient method: Derivative-free method for nonsmooth optimization. *J. Optim. Theory. Appl.*, 137:317–334, 2008. doi:10.1007/s10957-007-9335-5.

[3] Adil Bagirov, Napsu Karmitsa and Marko M. Mäkelä. *Introduction to Nonsmooth Optimization*. Springer, Cham, 2014. doi:10.1007/978-3-319-08114-4.

[4] Adil M. Bagirov, Sona Taheri and Napsu Karmitsa. Discrete gradient methods. In Adil M. Bagirov, Manlio Gaudioso, Napsu Karmitsa and Marko M. Mäkelä, editors, *Numerical Nonsmooth Optimization*, pages 621–654. Springer, Cham, 2020. doi:doi.org/10.1007/978-3-030-34910-3.

[5] I. Bongartz, A. R. Conn, N. I. M. Gould, D. Orban and Ph. L. Toint. CUTEst - a constrained and unconstrained testing environment for optimization software using safe threads. `https://github.com/ralna/CUTEst/wiki`, December 2015.

[6] Andrew J. Booker, J. E. Dennis, Jr., Paul D. Frank, David B. Serafini and Virginia Torczon. Optimization using surrogate objectives on a helicopter test example. In Jeff Borggaard, John Burns, Eugene Cliff and Scott Schreck, editors, *Computational Methods for Optimal Design and Control*, volume 24 of *Progress in Systems and Control Theory*, pages 49–58. Birkhäuser, Boston, MA, 1998. doi:10.1007/978-1-4612-1780-0_3.

[7] Jonathan M. Borwein and Adrian S. Lewis. *Convex Analysis and Nonlinear Optimization*. CMS Books in Mathematics. Springer, New York, 2nd edition, 2006. doi:10.1007/978-0-387-31256-9.

[8] C. Charalambous and A. R. Conn. An efficient method to solve the minimax problem directly. *SIAM J. Numer. Anal.*, 15(1):162–187, February 1978. doi:10.1137/0715011.

[9] Frank H. Clarke. Generalized gradients and applications. *Trans. Am. Math. Soc.*, 205: 247–262, 1975. doi:10.1090/S0002-9947-1975-0367131-6.

[10] A. R. Conn, K. Scheinberg and Luís N. Vicente. Geometry of interpolation sets in derivative free optimization. *Math. Program.*, 111(1–2):141–172, 2008. doi:10.1007/s10107-006-0073-5.

[11] Andrew R. Conn, Nicholas I. M. Gould and Philippe L. Toint. *Trust-Region Methods*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics and Mathematical Programming Society, Philadelphia, 2000.

[12] Andrew R. Conn, Katya Scheinberg and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics and Mathematical Programming Society, Philadelphia, second edition, 2009.

[13] V. F. Demyanov, L. N. Polyakova and A. M. Rubinov. Nonsmoothness and quasidifferentiability. In V. F. Demyanov and L. C. W. Dixon, editors, *Quasidifferential Calculus*, volume 29 of *Mathematical Programming Studies*, pages 1–19. Springer, Berlin Heidelberg, 1986. doi:10.1007/BFb0121133.

[14] Aurél Galántai. Convergence of the Nelder-Mead method. *Numer. Algorithms*, 90(3): 1043–1072, 2022. doi:10.1007/s11075-021-01221-7.

[15] W. Kahan. Numerical linear algebra. *Can. Math. Bull.*, 9(5):757–801, 1966. doi:10.4153/CMB-1966-083-2.

[16] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright and Paul E. Wright. Convergence properties of the Nelder–Mead simplex method in low dimensions. *SIAM J. Optim.*, 9(1):112–147, 1998. doi:10.1137/S1052623496303470.

[17] Jeffrey C. Lagarias, Bjorn Poonen and Margaret H. Wright. Convergence of the restricted Nelder–Mead algorithm in two dimension. *SIAM J. Optim.*, 22(2):501–532, 2012. doi:10.1137/110830150.

[18] Jeffrey Larson, Matt Menickelly and Stefan M. Wild. Derivative-free optimization methods. *Acta Numer.*, 28:287–404, May 2019. doi:10.1017/S0962492919000060.

[19] Jesús A. De Loera, Jamie Haddock and Luis Rademacher. The minimum euclidean-norm point in a convex polytope: Wolfe's combinatorial algorithm is exponential. *SIAM J. Comput.*, 49(1):138–169, 2020. doi:10.1137/18M1221072.

[20] K. I. M. McKinnon. Convergence of the Nelder–Mead simplex method to a nonstationary point. *SIAM J. Optim.*, 9(1):148–158, 1998. doi:10.1137/S1052623496303482.

[21] Robert Mifflin. Semismooth and semiconvex functions in constrained optimization. *SIAM J. Control Optim.*, 15(6):959–972, November 1977. doi:10.1137/0315061.

[22] Robert Mifflin. An algorithm for constrained optimization with semismooth functions. *Math. Oper. Res.*, 2(2):191–207, May 1977.

[23] James R. Munkres. *Analysis on Manifolds*. Addison–Wesley Publishing Company, Redwood City, CA, 1991.

[24] J. A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7 (4):308–313, 1965. doi:10.1093/comjnl/7.4.308.

[25] J. M. Parkinson and D. Hutchinson. An investigation into the efficiency of variants on the simplex method. In F. A. Lootsma, editor, *Numerical Methods for Non-linear Optimization*, pages 115–135, London, 1972. Academic Press.

[26] Jean-Paul Penot. Differentiability of relations and differential stability of perturbed optimization problems. *SIAM J. Control. Optim.*, 22(4):529–551, July 1984. doi:10.1137/0322033.

[27] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.*, 92:555–582, May 2002. doi:10.1007/s101070100290.

[28] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In Gianni Di Pillo and Massimo Roma, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, chapter 16, pages 255–297. Springer, Boston, MA, 2006. doi:10.1007/0-387-30065-1_16.

[29] T. M. Ragonneau and Z. Zhang. PDFO: Cross-platform interfaces for Powell's derivative-free optimization solvers (version 1.2). October 2021. doi:10.5281/zenodo.5554142.

[30] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comput. J.*, 3(3):175–184, 1960. doi:10.1093/comjnl/3.3.175.

[31] P. H. Schoute. *Mehrdimensionale Geometrie I: De linearen Räume*, volume 35 of *Sammlung Schubert*. G. J. Göschensche Verlagshandlung, Leipzig, 1902.

[32] Yaron Singer. AM 221: Advanced optimization — lecture notes: Lecture 10 (February 29th, Spring 2016). `https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture10.pdf`, February 2016.

[33] W. Spendley, G. R. Hext and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, November 1962. doi:10.2307/1266283.

[34] P. Stein. A note on the volume of a simplex. *Amer. Math. Monthly*, 73(3):299–301, March 1966. doi:10.2307/2315353.

[35] Endre Süli and David Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003. doi:10.1017/CBO9780511801181.

[36] *MATLAB: Mathematics*. The Mathworks, Inc., Natic, MA, USA, revised for MATLAB 9.13 (Release 2022b) edition, September 2022. Online available at: `https://mathworks.com/help/releases/R2022b/pdf_doc/matlab/matlab_math.pdf`.

[37] *MATLAB: Function Reference*. The Mathworks, Inc., Natic, MA, USA, revised for MATLAB 9.13 (Release 2022b) edition, September 2022. Online available at: `https://mathworks.com/help/releases/R2022b/pdf_doc/matlab/matlab_ref.pdf`.

[38] Paul Tseng. Fortified-descent simplicial search method: A general approach. *SIAM J. Optim.*, 10(1):269–288, 1999. doi:10.1137/S1052623495282857.

[39] Stefan Martin Wild. *Derivative-Free Optimization Algorithms for Computationally Expensive Functions*. PhD thesis, Graduate School of Cornell University, January 2009.

[40] Philip Wolfe. Finding the nearest point in a polytope. *Math. Program.*, 11:128–149, December 1976. doi:10.1007/BF01580381.

[41] Daniel John Woods. *An Interactive Approach for Solving Multi-Objective Optimization Problems*. PhD thesis, Rice University, May 1985.

[42] zhw. (https://math.stackexchange.com/users/228045/zhw). Is there a function that's continuous and has all directional derivatives as a linear function of direction, but still fails to be differentiable? Mathematics Stack Exchange, October 2015.
URL: `https://math.stackexchange.com/q/1497794`.