Imperial College London

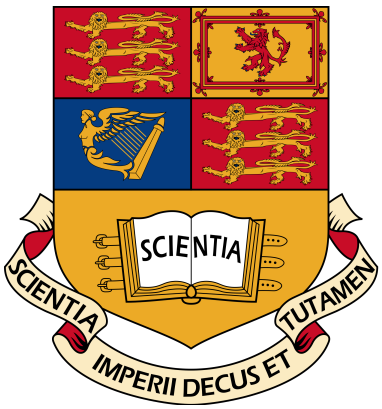
Department of Computing

MEng Individual Project

Linear Programming for Piecewise Linear Geometric Objects with Function and Derivative Constraints

Author:
Constantin Mateescu

Supervisor:
Prof. Abbas Edalat



June, 2016

Abstract

A piecewise linear function of two real variables, whose graph is a piecewise linear surface in the Euclidean space, can be defined by the lower and upper constraints on its value and its partial derivatives inside each sub-rectangle of a two-dimensional grid. If both the function and derivative information are consistent, then we can construct such a map by joining together local patches of the surface in each sub-rectangle of the grid. A similar result holds in higher dimensions, by considering a partitioning of sub-hyper-rectangles of an n-dimensional domain, where the function value and all the partial derivatives lie within closed and compact intervals.

In this project we develop a framework, using linear programming, that can check consistency in the general case of an n-dimensional domain, $n \geq 2$. Whenever the test indicates consistency, we also determine the minimal and maximal bounding surfaces. We achieve this by extending the known linear programming algorithm which decides consistency and we provide a simple analytical proof of this new result. Also, we implement a graphical user interface to help visualise the 3D piecewise linear surfaces, in the case when the domain is two-dimensional. During evaluation, we extensively test the correctness of our implementation by reverse engineering the constraints of the linear programming algorithms.

Finally, we investigate the problem when the constraint on the gradient of the surface is contained in a convex polygon, rather than a simple rectangle. Here we develop an algorithm that can decide consistency whenever the domain is given by a two-dimensional triangle.

Acknowledgements

I would like to express my gratitude to my project supervisor, Professor Abbas Edalat for all the support provided throughout the year as well as for his valuable suggestions and comments.

Contents

Chapter 1

Introduction

Many of the recent advances in modern sciences have been possible due to the improvements made in computational and optimisation theory developed over the course of the last decades. Various fields ranging from electrical engineering and mechanics, to economics and molecular modelling, have been able to take advantage of the latest developments in the sphere of mathematical optimisation. Mostly, this area provides mechanisms through which the optimal solution to a problem, also known as optimum, is produced.

Since very ancient times, people have been developing models and theories to deal with various optimisations problems in order to achieve the best possible results for specific tasks in everyday life. While interesting in theory, those ideas had very little practical use due to the daunting amount of computational effort required. It was not until the advent of the computer that those early thoughts have been resurrected and resulted in what is now regarded as a growing branch of applied mathematics.

On the other hand, whilst computers have evolved rapidly over the last years and are able to cope with an enormous amount of computation, the problem of precision, that is, accurate calculation, is still one of great significance even today. The issue arises from the floating-point representation widely available in today's computers which is known to have limited precision. However, certain scientific and engineering applications cannot tolerate the presence of any such rounding errors, as these may lead to catastrophic events. For example, in computational geometry we may need to deal with infinite amount of precision in order to accurately track the position of objects and points in space. This is known as an emerging trend in exact computation, explained further in this work.

1.1 Motivation

In this context of approximation and computability, differential equations, introduced in the 17th century by Newton and Leibniz, play a central role in modern mathematics and have countless applications in almost all branches of contemporary science. Several numerical approaches for computing the

solutions to differential equations have been developed, including the well-known Euler and Runge-Kutta methods. These have been shown, however, to suffer from a great loss of precision as their error estimation is too conservative to be of any practical use [1].

One classical problem is the famous *initial-value problem*, which states that, given an initial condition and an ordinary/partial differential equation that models some evolution of a system, we can determine the unknown function describing the underlying process. For such problems, a novel technique for computing the unique solution up to any desired accuracy has been proposed in [1] (such a solution is guaranteed to exist under certain assumptions). The basic idea is that, at each stage of the computation, one can obtain two continuous maps which provide upper and lower bounds for the solution, essentially giving the precise error.

A related problem to the one described above – and formulated again in [1] and [2] –, concerns the idea of *consistency* of two given maps subject to some constraints and will make most of the subject of the present work. A pair of functions (f, g), representing function and derivative information, respectively, is said to be *consistent* if one can find a third map h which is approximated by the first element of the pair and whose derivative information is approximated by the second component of the pair.

Furthermore, assuming that f is defined by lower and upper constraints on its value and g is restricted to lie within rectangles (or hyper-rectangles, if referring to higher dimensional spaces), then the property of consistency is *decidable* upon solving a finite set of inequalities which represent the constraints of a linear programming problem. Finally, if such a map h exists, then we can construct the minimal and maximal piecewise linear surfaces which are consistent with the information from both f and g.

1.2 Objectives

This discussion brings us to the aim of this project, which is twofold:

- Firstly, to implement the linear programming test that decides consistency when the function approximation is interval-valued and the gradient constraints lie within hyper-rectangles, for a certain partitioning of an n-dimensional domain. Also, in case the test indicates consistency, the task is to determine the global bounding surfaces;

- Secondly, to study a more general setting in which the derivative constraints are considered to lie within convex polyhedra, rather than hyper-rectangles, which is a more challenging problem.

1.3 Contributions

As a project which required both practical implementation and deep theoretical study in the field of multi-variable differentiable calculus, we made the following contributions:

- **Linear Programming Algorithms for Determining the Least and Greatest Piecewise-Linear Maps in the Rectangular Case**
  We extended the linear programming test presented in [2], which can decide consistency for rectangular derivative constraints in an arbitrary dimension $n \geq 2$, to also account for the explicit construction of the minimal and maximal piecewise-linear surfaces. We also provide a simple analytical proof of the result.

- **Framework for Deciding Consistency in the Rectangular Case Using Linear Programming**
  We implemented a general framework to check consistency of function and rectangular derivative information in an arbitrary n-dimensional domain, $n \geq 2$, by using the linear programming test from [2]. Whenever the test indicates consistency, we also determine the least and greatest surfaces by means of the above linear programming algorithms. We also created a simple GUI to allow for the visualisation of the 3D piecewise linear surfaces in the case of a two-dimensional domain.

- **Decidability of Consistency for a Two-Dimensional Triangle with Convex Derivative Constraints**
  We also develop a simple algorithm for deciding consistency in the particular instance when the domain is given by a two-dimensional triangle in which the function information is also interval-valued, but the derivative constraints lie within a convex polygon, rather than a simple rectangle.

Chapter 2

Background

This section provides the necessary background for understanding the theoretical aspects of this project. We begin by introducing the notion of exact computation [3], a new emerging paradigm in the field of modern computation. We also discuss the significance of this work in the context of exact geometric computation [4]. Finally, we introduce the main concepts of optimisation and linear programming that the reader should be familiar with.

2.1 Exact Computation

The underlying nature of computation is par excellence numerical: numbers have been at the heart of all calculations since very ancient times, while modern mathematics developed a whole range of theories to formalise many aspects of computable numbers. Early computers had the sole purpose of performing large complex calculations (the so-called number crunchers), which then turned into the original mass-produced computers, in the form of friendly pocket calculators. Although computers have evolved significantly and moved towards more abstracted and higher-level programs, numerical computation remains a major pillar of modern computer technology.

In particular, scientific computation is a rapidly growing inter-disciplinary field that makes use of advanced numerical capabilities. It is considered as adding a new dimension to the classical methods of theory and experimentation, sometimes referred to as the "third scientific method" of computation [3].

2.1.1 Fixed-Point Paradigm

At its core, scientific computation is subject to the *fixed-precision paradigm* of computation. Under this representation, numbers are expressed using a fixed number of digits after the decimal/radix point, thus providing fixed computational precision (usually machine-dependant).

Based on this specification, one can use several approaches to limit the unavoidable rounding errors. For instance, a *mild form* of fixed-precision

can be applied such that computations are performed up to a user-specified precision level. Although we can set a very high level of precision that we may think is satisfactory, the build-up of *round-off errors* that accumulate may produce totally unexpected results. We can illustrate this with an example from [5]. Consider the sequence a_n defined recursively as:

$$a_n = \begin{cases} \dfrac{11}{2} & , \quad n = 0 \\ \dfrac{61}{11} & , \quad n = 1 \\ 111 - \dfrac{1130 - \frac{3000}{a_{n-1}}}{a_n} & , \quad n \geq 2 \end{cases} .$$

Using the Unix utility `bc`, we can compute the terms of the sequence a_n up to some fixed precision of k decimal places, which we shall denote by $a_n^{(k)}$. Performing calculations with 5 decimal places, gives (note that the results have been rounded for presentation purposes, as in [5]):

$a_0^{(5)}$	5.500
$a_1^{(5)}$	5.500
$a_2^{(5)}$	5.500
$a_3^{(5)}$	5.500
$a_4^{(5)}$	5.648
$a_5^{(5)}$	5.242

$a_6^{(5)}$	-3.241
$a_7^{(5)}$	283.1
$a_8^{(5)}$	103.738
$a_9^{(5)}$	100.209
$a_{10}^{(5)}$	100.012
$a_{11}^{(5)}$	100.001

One would therefore believe that the sequence converges to 100. However, computing the number a_{100} with higher and higher precisions will contradict our expectations (the "exponents" below indicate repeating digits, e.g. $1.2^{3}4 = 1.2224$):

$a_{100}^{(5)}$	$100.0^{4}1$
$a_{100}^{(30)}$	$100.0^{29}1$
$a_{100}^{(60)}$	$100.0^{57}997$
$a_{100}^{(100)}$	$100.0^{17}98\ldots$

$a_{100}^{(110)}$	$100.0^{7}92$
$a_{100}^{(120)}$	$-3.790\ldots$
$a_{100}^{(130)}$	$5.9^{7}8697\ldots$
$a_{100}^{(140)}$	$5.9^{7}87925\ldots$

Here we can notice that if the precision is either 5, 30, 60, 100 or even 110 decimal places, then our expectation from above holds ($a_n \to 100$ as n grows larger). However, when increasing the precision even further, we obtain rather spurious results.

The actual limit of the sequence is equal to 6, since we can evaluate the general term a_n as:

$$a_n = \frac{6^{n+1} + 5^{n+1}}{6^n + 5^n}.$$

This example therefore shows how even the *mild form* of fixed-precision can lead to flawed results. It is therefore unclear what level of precision needs to be set in advance to a program such as `bc` in order to obtain the correct answer. As we could see, up until 110 decimal places we got roughly the same (wrong) result and we actually had to consider precision above 130 decimal places to arrive at the right answer.

2.1.2 Floating-Point Representation

Similar to the fixed precision paradigm, modern computers use floating-point arithmetic to perform real number calculations. This representation can be seen as a trade-off between range and precision when approximating real number: with a given precision, the floating-point model is able to represent both numbers of small magnitude with many bits of significance or conversely, large magnitude and few bits of significance. While many different representations have been developed in the past, the one defined by IEEE 754 has emerged as the industry standard. This standard is a step forward towards addressing the issue of portability and therefore makes errors in floating point computation *machine-independent*.

However, this representation is essentially just as inaccurate as the fixed-point model, since we must approximate real numbers by their nearest representable one. Thus, the same rounding errors will occur in practice, e.g. when small inaccuracies propagate in successive iterations like the one that we have just seen when computing a simple mathematical limit.

2.1.3 Towards an Alternative to the f.p. Paradigm

Even with industry-leading standards, rather intractable problems arise from the presence of round-off errors and compromise the *robustness* of the f.p.

paradigm. We can, at the arithmetic level, increase precision via techniques such as double extended precision, guard-bits, gradual underflow etc, which can usually be implemented in hardware. Interval arithmetic is another well-known method. Looking from a geometric perspective, an idea would be to divide the input and computed data into combinatorial and numerical, and to give precedence to the former when making decisions. An argument in favour of this approach is that we can allow the numerical data to be perturbed in order to maintain the combinatorial data. This avoids "topological inconsistencies" and can be implemented for simple cases, but the intractable nature of combinatorial problems makes it rather difficult to deal with more general cases [3].

Therefore, in the light of these non-robustness issues, one needs a completely different approach to handle cases in which the correct and exact answer is required. A new direction in the literature of computation is called, unsurprisingly, the *exact computation paradigm*. According, to [3] or [4], this paradigm assumes a computational process that:

1. represents all the underlying mathematical objects *exactly*;

2. all branching decisions are error-free.

As a result, multi-precision arithmetic is a necessary condition (but not sufficient) for exact computation. A different issue is that exact computation will naturally come at the expense of performance. It therefore makes sense to target only those applications which are not *cycle-critical* (i.e. we afford to incur some sort of computational slow-down). For example, exact computation cannot be avoided in computational number theory and in many aspects of algebra (e.g. testing the irreducibility of a polynomial).

Finally, we conclude this section with the idea of *weak exact computation*, similar in intent with the mild form of the f.p. paradigm. As explained in the numerical example above, one starts the computations using some fixed bound k on the precision. However, one would need to perform a thorough analysis as to what minimal values of k will indeed give the exact results. This is clearly a non-trivial problem and several suggestions have been proposed: in the same [3], the theory of root bounds has been developed; in [5], the exact real arithmetic offers lower and upper bounds guarantees that are trustworthy.

2.2 Linear Programming

A great deal of problems encountered in the real world involve maximisation or minimisation of certain quantities. Most often, these quantities we seek to optimise are profits (in the case of maximisation) as well as costs (in the case of minimisation). In linear programming we therefore aim to minimise/maximise a certain linear function (which we usually call objective function) subject to some linear constraints that describe the restrictions of our problem.

One would immediately think that calculus, developed by Leibniz and Newton in the 17th century, can deal with this types of problems very elegantly given the arsenal of techniques readily available. However, calculus is inadequate since it can only imply that the maxima and/or minima of some objective function lie on the boundaries of the sets determined by the constraints. Thus, a special set of techniques and algorithms is needed to deal with such linear programming problems (this is what the term "programming" really refers to in this case).

As such, linear programming plays a very important role in the fields of mathematics and business, finding many applications in management science and operations research.

2.2.1 Linear Programming in Standard Form

A linear program (LP) is an optimisation problem in which the *objective function* (i.e. the quantity we want to minimise/maximise) is linear in the unknowns and the constraints consist of linear equalities and/or linear inequalities. The exact form of these constraints may be problem-dependant, but one can always rewrite a linear program in the following *standard form*:

$$\begin{array}{lccccccccc} \text{Minimise} & c_1x_1 & + & c_2x_2 & + & \dots & + & c_nx_n & & \\ \text{subject to} & a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ & a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ & & & & \vdots & & & & & \\ & a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m \\ & x_1, x_2, \dots x_n \geq 0 & & & & & & & & \end{array} \tag{2.1}$$

where the coefficients b_i, c_j, a_{ij}, $i = \overline{1, m}$, $j = \overline{1, n}$ are fixed real constants and the *decision variables* x_i, $i = \overline{1, n}$, are yet to be found. For simplicity, we will

assume that all $b_i \geq 0$ (if necessary, one can multiply by -1 the equations for which $b_i \leq 0$). Lastly, the first m constraints are said to be *main constraints*, while the second n constraints are called *non-negativity constraints*.

We can rewrite the standard form given by 2.1 using matrix notation to derive the following compact statement:

$$\begin{array}{ll} \text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \tag{2.2}$$

where $\boldsymbol{x}$ is an n-dimensional column vector, $\boldsymbol{c}^\top$ is an n-dimensional row vector, $\boldsymbol{A}$ is an $m \times n$ matrix and $\boldsymbol{b}$ is an m-dimensional column vector with $\boldsymbol{b} \geq \boldsymbol{0}$. Inequalities of the type $\boldsymbol{x} \geq \boldsymbol{0}$ are understood to hold component-wise.

To illustrate the point that we can convert any LP problem into standard form, we need to consider various scenarios that may arise in practice. Apart from the fact that we may have equality constraints with negative right hand sides (which we can mitigate upon multiplication with -1, as previously mentioned), general LP problems can:

1. be maximisation (instead of minimisation) problems;
2. have inequality (instead of equality) constraints;
3. have free (instead of non-negative) decision variables.

In order to deal with maximisation problems, we need to observe that we can simply invert the objective function and then formulate the corresponding LP problem as a minimisation problem. We thus state the following:

Theorem 2.2.1. *Let $f : \Omega \to \mathbb{R}$ be a real valued function and assume that both the minimum and maximum of f are attained within the set Ω. Then:*

$$\min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x}) = -\max_{\boldsymbol{x} \in \Omega} -f(\boldsymbol{x}).$$

Proof. The minimum of f satisfies:

$$\min f(\boldsymbol{x}) = f^\star \leq f(\boldsymbol{x}), \forall\ x \in \Omega.$$

Similarly, the maximum of $-f$ satisfies:

$$\max -f(\boldsymbol{x}) = F^\star \geq -f(\boldsymbol{x}), \forall\ x \in \Omega.$$

But if $-F^\star < f^\star$ then there exists an $\overline{\boldsymbol{x}} \in \Omega$ such that $f(\overline{\boldsymbol{x}}) < f^\star$ contradicting the optimality of $f^\star$. Conversely, if $-f^\star > F^\star$ then there exists an $\overline{\boldsymbol{x}} \in \Omega$ such that $-f(\overline{\boldsymbol{x}}) > f^\star$ contradicting the optimality of $F^\star$. In conclusion $f^\star = -F^\star$. □

Now, to address the issue of LP problems with inequality constraints, we need to consider the following problem in which the constraints are given by linear inequalities:

$$\begin{array}{lllllllll}
\text{Minimise} & c_1x_1 & + & c_2x_2 & + & \ldots & + & c_nx_n & \\
\text{subject to} & a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \leq & b_1 \\
& a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & \leq & b_2 \\
& & & & \vdots & & & & & \\
& a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & \leq & b_m \\
& x_1, x_2, \ldots x_n \geq 0. & & & & & & & &
\end{array}$$

In this case, one can introduce new positive variables $s_i \geq 0$ (called *slack variables*), where $i = \overline{1, m}$, to convert each inequality constraint into an equality in the following manner:

$$\begin{array}{lllllllllll}
\text{Minimise} & c_1x_1 & + & c_2x_2 & + & \ldots & + & c_nx_n & & & \\
\text{subject to} & a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & + & s_1 & \leq & b_1 \\
& a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & + & s_2 & \leq & b_2 \\
& & & & \vdots & & & & & & & \\
& a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & + & s_m & \leq & b_m \\
& x_1, x_2, \ldots x_n \geq 0, & & & & & & & & & & \\
& s_1, s_2, \ldots s_m \geq 0. & & & & & & & & & &
\end{array}$$

It is now easy to see that this is an LP problem in standard form with $m+n$ decision variables and coefficient matrix having the form $[\boldsymbol{A}, \boldsymbol{I}]$ (where $\boldsymbol{A}$ is the original matrix as defined in the standard form 2.1, and $\boldsymbol{I}$ is the $m \times m$ identity matrix – corresponding to the slack variables s_i that have been added). In matrix notation, we can write that:

$$\begin{array}{lll}
\text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} & \\
\text{subject to} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} & \Longleftrightarrow \\
& \boldsymbol{x} \geq \boldsymbol{0}, &
\end{array}
\qquad
\begin{array}{ll}
\text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\
\text{subject to} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b} \\
& \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{s} \geq \boldsymbol{0},
\end{array}$$

where $\boldsymbol{s} = [s_1, s_2, \ldots, s_m]^\top$. Hence, the slack variables characterise the difference $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$. Similarly, one can deal with reversed inequality constraints such as:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i$$

by introducing *surplus variables* $s_i \geq 0$ (also known as *excess variables*):

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n - s_i = b_i.$$

Again, the resulting LP problem will have $n + m$ decision variables and we can write it in compact matrix form as follows:

$$\begin{array}{lll} \text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} & \Longleftrightarrow \\ & \boldsymbol{x} \geq \boldsymbol{0} & \end{array} \quad \begin{array}{lll} \text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{s} = \boldsymbol{b} & \\ & \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{s} \geq \boldsymbol{0}, & \end{array}$$

where surplus variables characterise the difference $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$.

Finally, let us deal with LP problems in which decision variables x_i are free, i.e. the unknowns can either be positive or negative. There are two different techniques that allow one to convert such a problem into standard form:

(a) If x_i is unconstrained, then we can use the substitution: $x_i = x_i^+ - x_i^-$, where $x_i^+, x_i^- \geq 0$. This means that the LP problem is now in standard form and has $n + 1$ decision variables, namely: $x_1, x_2, \ldots, x_{i-1}, x_i^+, x_i^-, x_{i+1}, \ldots, x_n$.

(b) If x_i is unconstrained, then we can use any equality constraint to eliminate x_i so that the standardised LP problem will now have $n-1$ decision variables and $m - 1$ constraints. The value of the free variable x_i can then be determined from the equation used to eliminate the variable in the first place.

Example 2.2.1.1. *Here is a simple example which illustrates this technique. Consider the following LP program which we wish to standardise:*

$$\begin{array}{lrcrcrcl} \text{Minimise} & x_1 & + & 3x_2 & + & 4x_3 & & \\ \text{subject to} & x_1 & + & 2x_2 & + & x_3 & = & 5 \\ & 2x_1 & + & 3x_2 & + & x_3 & = & 6 \\ & & & x_2, x_3 \geq 0. & & & & \end{array}$$

Since x_1 is an unconstrained variable, use the first equation to substitute $x_1 = 5 - 2x_2 - x_3$ and convert the original LP problem into:

$$\begin{array}{lrcrcl} \text{Minimise} & x_2 & + & 3x_3 & + & 5 \\ \text{subject to} & x_2 & + & x_3 & = & 4 \\ & & & x_2, x_3 \geq 0, & & \end{array}$$

which is now in standard form.

2.2.2 The Fundamental Theorem of LP

In this section we will develop the necessary background on basic solutions for linear programming problems that will allow us to state the Fundamental Theorem of Linear Programming (see Section 2.3 from [6]).

Let us therefore focus on the standard form given by 2.2. Recall that $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m_+$ and $\boldsymbol{c} \in \mathbb{R}^n$. In what follows, we will assume that the number of decision variables is always greater than or equal to the number of equations, i.e. $n \geq m$, as otherwise the system $\boldsymbol{Ax} = \boldsymbol{b}$ will be overdetermined. Also, we assume that a problem written in standard form has no redundant or inconsistent constraints, i.e. the rows of matrix $\boldsymbol{A}$ are linearly independent and hence $\text{rank}(\boldsymbol{A}) = m$.

Now, consider the system of linear equations from the standard form 2.2:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{2.3}$$

and suppose, for convenience, that we denote by $\boldsymbol{B}$ the $m \times m$ matrix formed by the first m columns of matrix $\boldsymbol{A}$. Then it follows that the matrix $\boldsymbol{B}$ is non-singular, so the equation:

$$\boldsymbol{Bx_B} = \boldsymbol{b}$$

is guaranteed to have a unique solution $\boldsymbol{x_B}$. By setting the first m elements of the vector $\boldsymbol{x}$ to be equal to those of $\boldsymbol{x_B}$ and filling the remaining $n - m$ entries with 0, i.e. $\boldsymbol{x} = [\boldsymbol{x_B}, \boldsymbol{0}]$, we have a solution for the linear system $\boldsymbol{Ax} = \boldsymbol{b}$. We are now in a position to formulate:

Definition 2.2.1. Given the set of m simultaneous linear equations in n unknowns 2.3, let $\boldsymbol{B}$ be any nonsingular $m \times m$ sub-matrix consisting of columns of $\boldsymbol{A}$. Then, if all $n - m$ entries of $\boldsymbol{x}$ not associated with columns of $\boldsymbol{B}$ are set equal to zero, the solution to the resulting set of equations is said to be a *basic solution* to 2.3 with respect to the basis $\boldsymbol{B}$. The entries of $\boldsymbol{x}$ associated with columns of $\boldsymbol{B}$ are called *basic variables*.

We also need to note that, under the rank assumption for matrix $\boldsymbol{A}$, i.e. $\text{rank}(\boldsymbol{A}) = m$, the system of linear equations 2.3 will always have at least one basic solution. However, basic variables in a basic solution can potentially be zero as well. We thus introduce the following:

Definition 2.2.2. If one or more of the basic variables in a basic solution is equal to zero, that solution is called *degenerate basic solution.*

Until this point, we have not yet treated the non-negativity constraints of the decision variables, i.e. $\boldsymbol{x} \geq 0$. So let:

$$\begin{aligned} \boldsymbol{Ax} &= \boldsymbol{b} \\ \boldsymbol{x} &\geq \boldsymbol{0} \end{aligned} \tag{2.4}$$

be the system of constraints for any LP problem in the standard form 2.2. We can now state similar definitions that apply when these restrictions are considered together, so let us present:

Definition 2.2.3. A vector $\boldsymbol{x}$ satisfying 2.4 is called *feasible* for these constraints. A feasible solution to the constraints 2.4 that is also basic is called *basic feasible solution*; if this solution is also a degenerate basic solution, it is called a *degenerate basic feasible solution.*

Before introducing the fundamental theorem of LP, we require one more definition to account for the optimality of solutions for an LP problem, so for this reason we present:

Definition 2.2.4. Given an LP in standard form, a feasible solution to the constraints 2.4 that achieves the optimal value of the objective function is called an *optimal feasible solution.* If the solution is basic then it is an *optimal basic feasible solution.*

In this moment we have all the necessary concepts to formulate the main result from this section:

Theorem 2.2.2. *(Fundamental Theorem of Linear Programming) Given a linear program in standard form 2.2, where* $\boldsymbol{A}$ *is an* $m \times n$ *matrix with* $\text{rank}(A) = m$*,*

1. *if there is a feasible solution, there is a basic feasible solution;*
2. *if there is an optimal feasible solution, there is an optimal basic feasible solution.*

2.2.3 Polyhedral Convex Sets

In this section we will depart from the elementary properties of linear systems that we employed in the study of the fundamental theorem of linear programming and rather focus on a more intuitive geometrical interpretation. Concepts from the standard theory of convex sets will be used to give an alternative definition to the fundamental theorem, which will lead to a clearer geometric understanding of the result. As it will soon become apparent, there is an intimate link between the basic feasible solutions of the algebraic theory and extreme points of convex polygons in the geometric approach. We proceed with the following definitions and theorems, followed in the next section by a concrete example.

Definition 2.2.5. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, $\boldsymbol{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$. Then the *line segment* between $\boldsymbol{x}$ and $\boldsymbol{y}$ (including the endpoints) is given by:

$$\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y},\ 0 \leq \alpha \leq 1.$$

Definition 2.2.6. Let $S \subset \mathbb{R}^n$. The set S is said to be *convex* if for all $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, $\boldsymbol{y} = (y_1, y_2, \ldots, y_n) \in S$ it holds that:

$$\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y},\ 0 \leq \alpha \leq 1.$$

Definition 2.2.7. The set of points $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ satisfying an equation of the form

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$$

is said to be a *hyperplane* of $\mathbb{R}^n$. The set of points $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ satisfying an inequality of the form

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \lesseqgtr b$$

is said to be a *closed half-space* of $\mathbb{R}^n$.

Theorem 2.2.3. *The constraint set of a canonical maximisation/minimisation linear programming problem is convex. Such a set is said to be a polyhedral convex set.*

Definition 2.2.8. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. The norm of $\boldsymbol{x}$, denoted $\|\boldsymbol{x}\|$, is given by:

$$\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}.$$

Definition 2.2.9. Let $r \geq 0$. The set of points $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ such that

$$\|\boldsymbol{x}\| \leq r$$

is said to be the *closed ball of radius r centred at the origin.*

Definition 2.2.10. A set $S \subset \mathbb{R}^n$ is said to be *bounded* if there exists $r \geq 0$ such that every element of S is contained in the closed ball or radius r centred at the origin. A set $S \subset \mathbb{R}^n$ is said to be *unbounded* if it is not bounded.

Definition 2.2.11. Let S be a convex set in $\mathbb{R}^n$. A point $e \in S$ is said to be an extreme point of S if there do not exist $\boldsymbol{x}, \boldsymbol{y} \in S$ and $\alpha \in (0, 1)$ such that

$$\mathbf{e} = \alpha\mathbf{x} + (1 - \alpha)\mathbf{y}.$$

Theorem 2.2.4. *If the constraint set S of a linear programming problem in standard form is bounded, then the maximum/minimum value of the objective function is attained at an extreme point of S.*

Theorem 2.2.5. *If the constraint set S of a linear programming problem in standard form is unbounded, then there exists some $M \in \mathbb{R}$ such that the objective function f satisfies $f(x_1, x_2, \ldots, x_n) \lesseqgtr M$ for all $(x_1, x_2, \ldots, x_n) \in S$, i.e. f is bounded above/below (by M), them the maximum/minimum value of the objective function is attained at an extreme point of S.*

2.2.4 A Geometric Approach

Let us illustrate the above concepts with a typical example of resource allocation problem.

Example 2.2.4.1. *A furniture company manufactures sofas and armchairs. The production of one sofa requires 2 hours in the parts division and 1 hour on the assembly line of the company. The production of one armchair requires 1 hour in the parts division and 2 hours on the assembly line. The parts department of the company operates at most 8 hours per day, while the assembly division is active at most 10 hours per day. Knowing that the profit of selling one sofa is £30 and the profit of selling one armchair is £50, what are the optimal quantities of sofas and armchairs that the company should produce in order to maximise profits?*

We start by translating the problem in mathematical terms. Note that we are interested in the number of sofas and armchairs to be produced, so let us denote:

$$\begin{aligned} x_1 &= \#\text{ of sofas per day;} \\ x_2 &= \#\text{ of armchairs per day.} \end{aligned}$$

According to the above definitions, x_1 and x_2 are the **decision variables** of our problem. Next, we wish to maximise the profits of the company, namely:

$$f(x_1, x_2) = 30x_1 + 50x_2.$$

This is the **objective function** that we want to optimise. But we cannot have unlimited quantities and hence infinite profits, since the company is constrained by the availability of the parts and assembly divisions. We observe that 2 hours are spent in the parts division for one sofa and 1 hour is spent in the same division for one armchair. Together with the constraint of 8 hours per day in this department, we derive the constraint:

$$2x_1 + x_2 \leq 8.$$

Similarly, we can derive the other constraint for the assembly line, which gives a second inequality:

$$x_1 + 2x_2 \leq 10.$$

Finally, we also have the obvious constrains:

$$\begin{aligned} x_1 &\geq 0, \\ x_2 &\geq 0. \end{aligned}$$

since the company cannot produce negative quantities of sofas or armchairs. The above 4 constraints represent the **feasible set** that enforce the admissible production plans $x = (x_1, x_2)$. Note that, for example, $x = (20, 30)$ cannot belong to this feasible set, as there is not enough availability in both the parts and assembly divisions to satisfy that level of production. The final optimisation problem is a maximisation problem that can be formulated mathematically as follows:

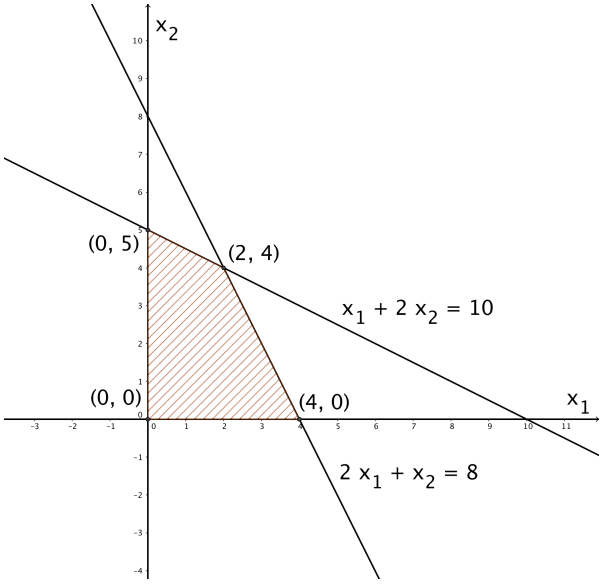
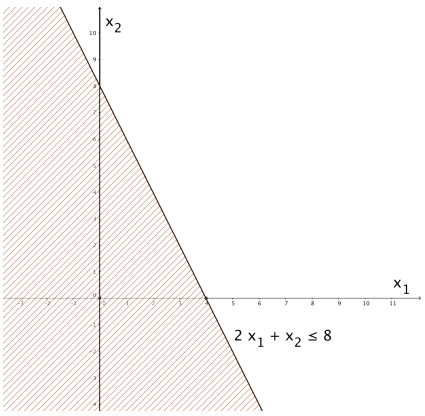
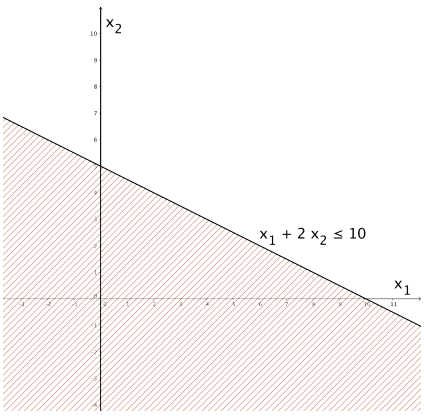


Figure 2.1: Feasible set of points satisfying all constraints

$$\begin{array}{ll} \text{Maximise} & f(x_1, x_2) = 30x_1 + 50x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 8 \\ & x_1 + 2x_2 \leq 10 \\ & x_1 \geq 0 \\ & x_2 \geq 0. \end{array}$$

The set of points (x_1, x_2) satisfying the above constraints is given by the shaded region from Figure 2.1.





The region in Figure 1 above was obtained by intersecting the four 2-dimensional regions determined by the constraints $2x_1+x_2 \leq 8$, $x_1+2x_2 \leq 10$,

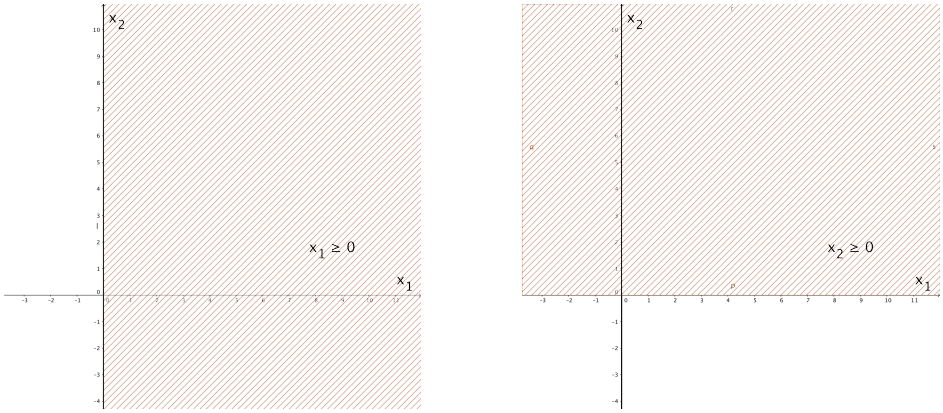


Figure 2.2: All individual constraints

$x_1 \geq 0$, and $x_2 \geq 0$ (Figure 2.2). Therefore, the question boils down to determining which point in the shaded region in Figure 1 maximises $f(x_1, x_2)$.

Since the constraint set is bounded, we can use Theorem 2.2.4 to deduce that the maximum value of $f(x, y)$ is attained at an extreme point in Figure 1, namely at either $(0, 0)$, $(4, 0)$, $(0, 5)$, or $(2, 4)$. A simple analysis shows that the maximum is attained at $(2, 4)$ where $f(2, 4) = 260$, and as a result the company should produce 2 sofas and 4 armchairs in order to maximise their profits (and get £260 in return).

2.2.5 Complexity Limitations and Alternative Approaches

Having seen the geometric method which allows us to reason about LP problems, we can already notice some of its limitations. In real-life LP problems, one would typically deal with potentially tens if not hundreds of variables.

For example, assuming m constraints and n decision variables, we will have at most

$$\binom{m+n}{n} = \frac{(m+n)!}{m! \cdot n!}$$

candidate extrema points to be tested. This clearly poses a problem and makes the visualisation of the constraint set impossible.

Another more relevant problem concerns the cases when the constraint set is unbounded. In such scenarios, according to Theorem 2.2.5 we would also need to ensure that the objective function is bounded by above or be-

low, depending on whether the LP problem at hand is a maximisation or a minimisation problem, respectively. This is largely problem dependant and can complicate the analysis significantly. One such example is given by the following LP problem:

$$\begin{array}{ll} \text{Maximise} & f(x,y,z) = 2x + 3y + 4z \\ \text{subject to} & y + 5x \leq 10 \\ & 2y + 3z \leq 15 \\ & x, y, z \geq 0 \end{array}$$

Here there is no restriction on x other than being positive, and as such for $x \to \infty$ we will have $f(x,y,z) \to \infty$, and hence the objective function is unbounded.

Given these obvious limitations, several algorithms have been developed to address this rather exponential nature of linear programming problems. The famous *simplex algorithm* is able to find optimal solutions to LP problems without testing a large number of candidate extrema points. It is also able to detect edge cases where the constraint sets are empty and the objective functions are unbounded (see [7]). The basic idea is that the simplex method is capable of continually decreasing the value of the objective function by intelligently *pivoting* from one feasible solution (i.e. one extreme point) to another. This reduces the search space considerably and leads to a more computationally-friendly algorithm. However, even with such optimisations in place, the simplex algorithm can still exhibit exponential time complexity, as shown by Klee-Minty (see Section 5.2 from [7] for such an example).

Cutting-plane algorithms such as the ellipsoid method have been found to have polynomial-time complexity in the size of the problem ($\mathcal{O}((m+n)^4)$), but it was discovered that in practice they do not perform any better than the simplex algorithm.

Currently, the state-of-the-art algorithm that is being used by major LP software packages relies on the interior-point method belonging to Karmarkar which has total complexity of the order $\mathcal{O}(nm^2 \log(n/\varepsilon))$, where ε is a tolerance parameter (see also [7]). This method was found to behave much better in practice than the simplex algorithm and it is also reasonably effective when applied to large-scale LP problems.

2.2.6 Duality Theory

In this last subsection on linear programming, we briefly discuss the duality relationship that holds for certain pairs of LP problems. To this end, we will deviate from the standard form given by 2.1, as duality arises naturally from the symmetry of LP problems expressed only in terms of inequalities. We now define duality through the following pair of programs:

$$\begin{array}{lll} \text{Maximise} & \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} & \\ & \boldsymbol{x} \geq \boldsymbol{0} & \end{array} \qquad \begin{array}{lll} \text{Minimise} & \boldsymbol{b}^\top \boldsymbol{y} & \\ \text{subject to} & \boldsymbol{A}^\top \boldsymbol{y} \geq \boldsymbol{c} & , \\ & \boldsymbol{y} \geq \boldsymbol{0} & \end{array} \tag{2.5}$$

where $\boldsymbol{A}$ is an $m \times n$ matrix, $\boldsymbol{x}$ is an n-dimensional column vector and $\boldsymbol{c}, \boldsymbol{y}$ are m-dimensional column vectors. The symmetry of the above LP problems allows us to define a *duality* relationship between the two programs: one of them will be called *primal*, while its counterpart will be named *dual*. It is also important to note that the roles of the primal and dual problems can be swapped. We are now able to state the fundamental theorems on duality:

Theorem 2.2.6. *(Weak Duality). If $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible solution for the LP problems defined by 2.5, then the following inequality holds: $\boldsymbol{c}^\top \boldsymbol{x} \leq \boldsymbol{b}^\top \boldsymbol{y}$.*

Proof. Since $\boldsymbol{x} \geq 0$ we have: $\boldsymbol{c}^\top \boldsymbol{x} \leq \left(\boldsymbol{A}^\top \boldsymbol{y}\right)^\top \boldsymbol{x} = \boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{y}^\top \boldsymbol{b} = \boldsymbol{b}^\top \boldsymbol{y}$. □

Theorem 2.2.7. *(Strong Duality). If either of the problems defined in 2.5 has a finite optimal solution, so does the other, and the corresponding values of the objective functions coincide. If either problem has an unbounded objective, the other problem has no feasible solution as well.*

Proof. The interested reader can refer to Section 4.2 from [6]. □

We conclude this brief tour of linear programming with some remarks that are connected to the duality relationship presented above. In the view of Theorem 2.2.1 presented in an earlier section, one can simplify the computational framework for LP problems by treating all such problems as being either minimisation problems or maximisations problems. Usually, most optimisations problems are formulated as minimisations problems and a convention is followed as to whether the column vector that defines the inequality constraints represents the lower bounds or the upper bounds of those constraints. Generally, the preferred way is to define the constraints as: $\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}$, so that $\boldsymbol{h}$ represents the column vector of right hand side terms.

2.3 CVXOPT Framework

In order to implement the linear programming algorithms that decide consistency of function and derivative information (which will be described in a later section), we had to choose a library that is specifically tailored for dealing with LP problems. Previous experience with GNU's GLPK package [8], which uses a rather obfuscated syntax with many intricate constructs, made us look for an alternative package in the open-source landscape. Given that Python is a well-established programming language, which is known for its ease of use which allows fast and convenient code development, we decided to narrow down our search to linear programming solvers available in Python.

After a quick search for such libraries, we discovered that the CVXOPT library [9] will be suitable for the purposes of this project, since it combines both decent performance and convenient formulation of the LP problems. In addition to handling problems with linear objectives, CVXOPT also features support for more general convex optimisation problems which have non-linear objective functions [citation needed].

2.3.1 Formulation of LP Problems in CVXOPT

The API for specifying the constraints and the objective function of a linear programming problem is very simple and has the following function signature (which is simplified here for presentation purposes) [citation needed]:

```
cvxopt.solvers.lp(c, G, h[, A, b[, solver]]),
```

and is designed for solving the following minimisation problem:

$$\begin{array}{ll} \text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \end{array} . \tag{2.6}$$

Note that the parameters `c`, `G`, `h` are compulsory, while the remaining ones are optional. Depending on whether the LP problem at hand features equality constraints or not, one would need to specify both the following parameters `A` and `b`. In addition, external solvers such as '`glpk`' or '`mosek`' (if installed) can be used instead of the default one used internally, by means of the argument `solver`. For completeness, assuming that there are n decision variables, p inequality constraints and $m-p$ inequality constraints, it holds that: $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{G} \in \mathbb{R}^{p \times n}$, $\boldsymbol{h} \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{(m-p) \times n}$ and $\boldsymbol{b} \in \mathbb{R}^{m-p}$.

2.3.2 Simple Example

We can now illustrate the strengths of the CVXOPT framework with a simple example. Given the following optimisation problem:

$$\begin{array}{ll} \text{Minimise} & f(x_1, x_2) = 2x_1 + x_2 \\ \text{subject to} & -x_1 + x_2 \leq 1 \\ & x_1 + x_2 \geq 2 \\ & x_2 \geq 0 \\ & x_1 - 2x_2 \leq 4, \end{array}$$

we ask for the optimal solution vector $\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$ which minimises the given linear objective function. Since we are only dealing with inequalities constraints, we simply need to construct the vector $\boldsymbol{c} \in \mathbb{R}^2$, the coefficient matrix $\boldsymbol{G} \in \mathbb{R}^{2\times 4}$ and the column vector of right hand sides $\boldsymbol{h} \in \mathbb{R}^4$. By transforming the above optimisation problem into the form 2.6, we have:

$$\boldsymbol{c} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \qquad \boldsymbol{G} = \begin{bmatrix} -1 & 1 \\ -1 & -1 \\ 0 & -1 \\ 1 & 2 \end{bmatrix} \qquad \boldsymbol{h} = \begin{bmatrix} 1 \\ -2 \\ 0 \\ 4 \end{bmatrix},$$

which we can now input in the interactive Python interpreter, followed by the call to `solvers.lp` to retrieve the optimal solution to the given problem:

```
>>> from cvxopt import matrix, solvers
>>> c = matrix([2.0, 1.0])
>>> G = matrix([[-1.0, -1.0, 0.0, 1.0], [1.0, -1.0, -1.0, -2.0]])
>>> h = matrix([1.0, -2.0, 0.0, 4.0])
>>> sol = solvers.lp(c, G, h)

     pcost       dcost       gap    pres   dres   k/t
 0:  2.6471e+00 -7.0588e-01  2e+01  8e-01  2e+00  1e+00
 1:  3.0726e+00  2.8437e+00  1e+00  1e-01  2e-01  3e-01
 2:  2.4891e+00  2.4808e+00  1e-01  1e-02  2e-02  5e-02
 3:  2.4999e+00  2.4998e+00  1e-03  1e-04  2e-04  5e-04
 4:  2.5000e+00  2.5000e+00  1e-05  1e-06  2e-06  5e-06
 5:  2.5000e+00  2.5000e+00  1e-07  1e-08  2e-08  5e-08

>>> print(sol['x'])
[ 5.00e-01]
[ 1.50e+00]
```

2.4 Domain Theory

We introduce basic domain theory knowledge that will become useful when dealing with the problem of consistency in a later section. We use [5] and [10] when referencing these results, unless otherwise stated.

2.4.1 Introduction

Domain theory was introduced in 1970 by Dana Scott as a mathematical theory of programming languages. The basic idea of domain theory is to provide better and better approximations to an object by means of simple recursion. It has applications in the field of computer science, e.g. solving canonically fixed point equations or recursive equations of procedures and data structures.

2.4.2 Main Definitions and Examples

Definition 2.4.1. A *partial order* (or a *partially ordered set* or *poset*) $(D, \leq)$ is a set D together with a binary relation $\leq$ which is:

1. reflexive: $a \leq a$,
2. anti-symmetric: $a \leq b \wedge b \leq a \implies a = b$, and
3. transitive: $a \leq b \wedge b \leq c \implies a \leq c$.

Most often, the binary relation of a partial order is written as $\sqsubseteq$. Then $a \sqsubseteq b$ can be interpreted as a having less information than b.

Definition 2.4.2. A subset A of an ordered set $(P, \sqsubseteq)$ is an *upper set* if $x \in A \implies y \in A$, for all $y \sqsupseteq x$. We denote by $\uparrow A$ the set of all elements above some element of A. For convenience, we abbreviate $\uparrow \{x\}$ as $\uparrow x$. The dual notions are lower set and $\downarrow A$ [11].

Definition 2.4.3. A non-empty subset $A \subset P$ is said to be *directed* if for any pair of elements $a, b \in A$ there exists $c \in A$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$.

Definition 2.4.4. A *directed complete partial order* (*dcpo*) or a domain is a partial order in which every directed subset has a least upper bound (lub). A dcpo is said to be *pointed* if it has a least element which is denoted by $\perp$ and is called bottom.

Definition 2.4.5. Given two elements a and b of a dcpo we say that a is *way-below* or *approximates* b, denoted by $a \ll b$, if for every directed subset A with $b \sqsubseteq \bigsqcup A$ there exists $c \in A$ with $a \sqsubseteq c$.

Definition 2.4.6. A basis of a domain D is a subset $B \subset D$ such that for every element $x \in D$ of the domain the set $B_x = \{y \in B \mid y \ll x\}$ of elements in the basis way-below x is directed with $x = \bigsqcup B_x$. Also, a dcpo with a (countable) basis is said to be an *(ω)-continuous* domain.

Definition 2.4.7. A function $f : D \to E$ between dcpo's is said to be *Scott-continuous* if and only if it is *monotone* (i.e. $a \sqsubseteq b \implies f(a) \sqsubseteq f(b)$) and preserves' lub's of directed sets i.e. for any directed $A \subseteq D$, we have $f\left(\bigsqcup_{a \in A} a\right) = \bigsqcup_{a \in A} f(a)$. Moreover, if D is an ω-continuous dcpo, then f is continuous if and only if it is monotone and preserves the lub's of increasing sequences (i.e. $f\left(\bigsqcup_{i \in \omega} x_i\right) = \bigsqcup_{i \in \omega} f(x_i)$, for any increasing $(x_i)_{i \in \omega}$).

Definition 2.4.8. Let D be a dcpo. A subset A is called *(Scott-)closed* if it is a lower set and is closed under suprema of directed subsets. Complements of closed sets are called *(Scott-)open*; they are the elements of ω_D, the *Scott-topology* on D [11].

Example 2.4.2.1. *[5] The interval domain $\mathbf{I}[0,1]^n$ of the unit box $[0,1]^n \subseteq \mathbb{R}^n$ is the set of all non-empty n-dimensional sub-rectangles in $[0,1]^n$ ordered by reverse inclusion. A basic Scott open set is given, for every open subset O of $\mathbb{R}^n$, by the collection of all rectangles contained in O.*

The map $x \mapsto \{x\} : [0,1]^n \to \mathbf{I}[0,1]^n$ is an embedding onto the set of maximal elements of $\mathbf{I}[0,1]^n$. Every maximal element $\{x\}$ can be obtained as the least upper bound (lub) of an increasing chain of elements, that is a shrinking, nested sequence of sub-rectangles, each containing $\{x\}$ in its interior and thereby giving an approximation to $\{x\}$ or equivalently to x. The set of sub-rectangles with rational coordinates provides a countable basis. One can similarly define, for example, the interval domain $\mathbf{I}\mathbb{R}^n$ of $\mathbb{R}^n$.

Chapter 3

Consistency of function and derivative constraints

In this section we focus on the main idea of the proposed project, which involves the concept of *consistency* between function and derivative information for a given real-valued map defined on an n-dimensional domain. More specifically, the basic idea is to determine whether a map $h : \mathbb{R}^n \to \mathbb{R}$ can be constructed given restrictions on the function value and derivative information, respectively. The nature of the derivative constraints can pose a major challenge towards the linear-programming algorithms that will be developed further, depending on whether we consider hyper-rectangles or convex-polyhedra in an n-dimensional setup.

We begin by stating the main notations and terminology on interval-valued maps, Lipschitz functions and their derivatives (see [2]). We then proceed to defining the consistency relationship in a domain-theoretical setting and towards the end of the chapter we present the algorithms that can decide consistency for maps defined on $\mathbb{R}^n$, $n \geq 1$. In what follows, we consider $n \geq 1$, unless otherwise stated.

3.1 Notations and terminology

We denote by $\mathbb{R}$ the set of real numbers and by $\mathbf{I}\mathbb{R} = \{[a, b] \mid a \leq b \in \mathbb{R}\} \cup \{\mathbb{R}\}$ the interval domain, i.e. the set of compact, nonempty intervals, equipped with a least element $\perp = \mathbb{R}$, ordered by reverse inclusion. It has a canonical basis consisting of all compact intervals with rational end points augmented with $\perp$. We write a non-bottom element $v \in \mathbf{I}\mathbb{R}$ as $v = [v^-, v^+]$ and we identify any real number $x \in \mathbb{R}$ with the singleton $\{x\} \subset \mathbb{R}$.

Also, we denote by $\mathbf{I}\mathbb{R}^n$ the product domain consisting of all non-empty compact hyper-rectangles with faces parallel to the standard coordinate planes ordered with reverse inclusion and augmented with the whole space $\mathbb{R}^n$ as the bottom element. It has a canonical basis consisting of all its rational (compact) hyper-rectangles and the bottom element. We denote the continuous Scott domain of the nonempty, compact and convex subsets of $\mathbb{R}^n$, taken

together with $\mathbb{R}^n$ as the bottom element and ordered by reverse inclusion, by $\mathbf{C}\mathbb{R}^n$. We will use a canonical basis of $\mathbf{C}\mathbb{R}^n$, consisting of rational convex compact polyhedra together with the set $\mathbb{R}^n$ as the bottom element.

For an open subset $U \subset \mathbb{R}^n$, let $C^0(U)$ be the function space of all continuous functions of type $U \to \mathbb{R}$. We will also use domains of function spaces of the form $(U \to D)$ where D is a countably based continuous dcpo, which is either $\mathbf{I}\mathbb{R}$, $\mathbf{I}\mathbb{R}^n$ or $\mathbf{C}\mathbb{R}^n$ in our case. For the sake of convenience, denote $D^0(U) = U \to \mathbf{I}\mathbb{R}$. A function $f \in D^0(U)$ is given by a pair of respectively lower and upper semi-continuous functions $f^-, f^+ : U \to \mathbb{R}$ with $f(x) = [f^-(x), f^+(x)]$ when $f(x) \neq \bot$ for all $x \in U$. Recall that given an open subset $a \subset U$ and an element $b \in D$, the single step function $b\chi_a : X \to D$ is dened as $(b\chi_a)(x) = b$ if $x \in a$ and $\bot$ otherwise. Single-step functions are continuous with respect to the Scott topology. Any finite set of single-step functions that are bounded in the function space $U \to D$ has a least upper bound, called a step function; the set of step functions provides a basis for the continuous Scott domain $U \to D$. This basis in turn gives a countable and canonical basis of rational step functions for $U \to D$, where $D = \mathbf{I}\mathbb{R}$, $\mathbf{I}\mathbb{R}^n$ or $\mathbf{C}\mathbb{R}^n$, generated by single-step functions of the form $b\chi_a$ where a is a rational open hyper-rectangle with faces parallel to the coordinate hyper-planes of $\mathbb{R}^n$ and b is a rational interval for $D = \mathbf{I}\mathbb{R}$, a rational hyper-rectangle for $D = \mathbf{I}\mathbb{R}^n$ and a rational compact convex polyhedron in $\mathbb{R}^n$ for $D = \mathbf{C}\mathbb{R}^n$. Finally, the set of elements above an element c in a domain will be denoted by $\uparrow c$.

Furthermore, we will make use of two operations from interval arithmetic which extend the conventional addition and multiplication of numbers by point-wise application to sets of points. Recall that $\|\boldsymbol{x}\| = \sqrt{\sum_{i=1}^n x_i^2}$ is the standard Euclidean norm of $\boldsymbol{x} = [x_1, x_2, \ldots x_n]^\top \in \mathbb{R}^n$. Then the Euclidean norm is extended point-wise to $b \in \mathbf{C}\mathbb{R}^n$ by $\|\boldsymbol{b}\| = \max\{\|\boldsymbol{x}\| \ : \ x \in \boldsymbol{b}\}$. We will also consider the extension $- \cdot - : \mathbf{C}\mathbb{R}^n \times \mathbb{R}^n \to \mathbf{I}\mathbb{R}$ of the scalar product which is defined point-wise $\boldsymbol{b} \cdot \boldsymbol{x} = \{\boldsymbol{y} \cdot \boldsymbol{x} \ : \ \boldsymbol{y} \in \boldsymbol{b}\}$.

Let us now recall the several well-known definitions from calculus regarding directional derivatives and Lipschitz functions.

Definition 3.1.1. (Partial Derivative) If $f : \mathbb{R}^n \to \mathbb{R}$, then its partial derivative with respect to dimension i is defined as:

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h \ldots x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}$$

Definition 3.1.2. (Gradient) If $f : \mathbb{R}^n \to \mathbb{R}$, then the vector of partial derivatives is called the *gradient* and is defined by:

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \\ \vdots \\ \\ \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}.$$

Definition 3.1.3. (Directional Derivative) Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real valued function, and let $\boldsymbol{d} \in \mathbb{R}^n \backslash \{\mathbf{0}\}$. The directional derivative of f in the direction $\boldsymbol{d}$ is defined as:

$$\frac{\partial f}{\partial \boldsymbol{d}}(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{d}) - f(\boldsymbol{x})}{h}.$$

Definition 3.1.4. (Lipschitz Function) [12] A function $f : S \to \mathbb{R}^m$ is called *Lipschitz continuous* if there is a constant L such that:

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L \cdot \|\boldsymbol{x} - \boldsymbol{y}\|, \ \forall \ \boldsymbol{x}, \boldsymbol{y} \in S.$$

Definition 3.1.5. Let $U \subset \mathbb{R}^n$ be an open subset of a $\mathbb{R}^n$ and let $f : U \to \mathbb{R}$ be Lipschitz near $\boldsymbol{x} \in U$ and $\boldsymbol{d} \in \mathbb{R}^n$. The *generalised directional derivative* [12] of f at $\boldsymbol{x}$ in the direction of $\boldsymbol{d}$ is given by:

$$f^{\circ}(\boldsymbol{x}; \boldsymbol{d}) = \lim_{\boldsymbol{y} \to \boldsymbol{x}} \sup_{h \downarrow 0} \frac{f(\boldsymbol{y} + h\boldsymbol{d}) - f(\boldsymbol{y})}{h}$$

Definition 3.1.6. (Clarke Gradient) The *generalised gradient* of f at $\boldsymbol{x}$, denoted by $\partial f(\boldsymbol{x})$ is the subset of $\mathbb{R}^n$ given by:

$$\{A \in X^{\star} : f^{\circ}(\boldsymbol{x}; \boldsymbol{d}) \geq A(\boldsymbol{d}) \text{ for all } \boldsymbol{v} \in \mathbb{R}^n\}.$$

In addition, $\partial f(\boldsymbol{x})$ is a non-empty, convex and compact subset of $\mathbb{R}^n$ and for $\boldsymbol{d} \in \mathbb{R}^n$ we have:

$$f^{\circ}(\boldsymbol{x}; \boldsymbol{d}) = \max\{A(\boldsymbol{d}) : A \in \partial f(\boldsymbol{x})\}$$

The following definitions, which are given in [2], provide the necessary background on the $\mathcal{L}$-derivative and the domain of Lipschitz maps, specialised to finite dimensions. We begin by introducing the following extension to Lipschitz functions:

Definition 3.1.7. (Set-valued Lipschitz functions) The continuous function $f : U \to \mathbb{R}$ has a *non-empty, convex and compact set-valued Lipschitz constant* $\boldsymbol{b} \in \mathbf{C}\mathbb{R}^n$ in an open subset $a \subset U$ if for all $\boldsymbol{x}, \boldsymbol{y} \in a$ we have:

$$\boldsymbol{b} \cdot (\boldsymbol{x} - \boldsymbol{y}) \sqsubseteq f(\boldsymbol{x}) - f(\boldsymbol{y}).$$

The single step tie $\delta(a, \boldsymbol{b}) \subseteq C^0(U)$ of a with $\boldsymbol{b}$ is the collection of all partial functions f on U with $a \subset \mathrm{dom}(f) \subset U$ in $C^0(U)$ which have $\boldsymbol{b}$ as non-empty convex compact set-values Lipschitz constant in a.

For the rest of this subsection, we will assume that $n \geq 2$. It is possible, as shown in [13] to give an equivalent definition for the Clarke gradient which is expressed only in terms of elementary set notions. In what follows, the equivalent generalised derivative for Lipschitz continuous maps will be referred to as $\mathcal{L}$-derivative. Also, from this points onwards we assume $n \geq 2$ and, in a slight abuse of notation, we will write C^0 instead of $C^0(U)$.

Definition 3.1.8. A *step tie* of C^0 is any finite intersection $\bigcap_{i \in I} \delta(a_i, b_i) \subset C^0$, where I is a finite indexing set. A *tie* of C^0 is any intersection $\Delta = \bigcap_{i \in I} \delta(a_i, b_i) \subset C^0$, for an arbitrary index set I. The *domain* of a non-empty tie Δ is defined as $\mathrm{dom}(\Delta) = \bigcup_{i \in I} \{a_i \mid b_i \neq \bot\}$.

Definition 3.1.9. The *primitive map* $\int : (U \to \mathbf{C}\mathbb{R}^n) \to T^1(U)$ is defined by $\int(g) = \bigcap_{i \in I} \delta(a_i, b_i)$, where $g = \sup_{i \in I} b_i \chi_{a_i}$. We usually write $\int(f)$ as $\int f$ and call it the set of *primitives* of f.

Definition 3.1.10. A map $g : U \to \mathbf{C}\mathbb{R}^n$ is said to be integrable if $\int g \neq \emptyset$.

Given a continuous function $f : U \to \mathbb{R}$, the relation $f\delta(a, \boldsymbol{b})$ gives finitary information about the local differential properties of f. The collection of all such information defines the $\mathcal{L}$-derivative of f, as follows:

Definition 3.1.11. The *derivative* of a continuous function $f : U \to \mathbb{R}$ is defined as:

$$\mathcal{L}f = \bigsqcup_{f \in \delta(a, \boldsymbol{b})} b_{\chi_a} : U \to \mathbf{C}\mathbb{R}^n$$

Theorem 3.1.1. *In the case of arbitrary (possibly infinite) dimension, it holds that (see [13]):*

1. $\mathcal{L}f$ *is well-defined and Scott continuous.*
2. *If* $f \in C^1$ *then* $\mathcal{L}f = f'$.
3. $f \in \delta(a, \boldsymbol{b})$ *if and only if* $b\chi_a \sqsubseteq \mathcal{L}f$.

The following interesting corollary holds, which can be seen as an alternative of the Fundamental Theorem of Calculus for domain-theoretic functions. Moreover, we can state the duality between Clarke gradient and the previously defined $\mathcal{L}$-derivative (see [13]):

Corollary 3.1.1.1. $f \in \int g \iff g \sqsubseteq \mathcal{L}f$.

Theorem 3.1.2. *In finite dimensional Euclidean spaces, the* $\mathcal{L}$*-derivative coincides with the Clarke gradient.*

3.2 The property of consistency

In this section, we will give a formal definition to the concept of consistency which we briefly mentioned at the beginning of this chapter and is the central piece of the present work. We will also present the algorithms that can decide whether consistency holds given constraints on the function and derivative information, respectively.

Informally, a pair of functions (f, g), representing function and derivative information respectively, is called *consistent* if there exists a third function w whose function value is approximated by the first component of the pair and whose derivative information is approximated by the second component.

The function information f is given by a finite set of *step functions* represented as $\{a_i, b_i\}_{i \in I}$, where $a_i \subseteq \mathbb{R}^n$ is a rational hyper-rectangle and $b_i \subseteq \mathbb{R}$ is a compact interval such that b_i and b_j have non empty intersection whenever this is the case for the interiors of a_i and a_j. Equivalently, we can write that $f \in D^0(U)$, i.e. f is interval-valued within each hyper-rectangle defined in its domain. On the other hand, the derivative information g is given for each of the n partial derivatives in the form $\{a_i, b_i\}_{i \in I}$, where a_i are as before, but we allow b_i to be, in the most general setting, rational compact polyhedra. Simply put, $g : U \to \mathbf{C}\mathbb{R}^n$, i.e. the range of is a convex polyhedra for each n-dimensional hyper-rectangle inside the domain.

As we will see, in the particular case when the derivative constraints lie within hyper-rectangles with faces parallel to the coordinate planes, we can decide whether a *consistency witness* w exists or not, via a linear programming algorithm. In addition, when the algorithm indicates that (f, g) is consistent, we can also construct the least and greatest consistent maps – in the sense that the heights of the minimal and maximal witnesses are minimised and maximised, respectively:

$$w_{\min} \leq w \leq w_{\max}.$$

As further explained, the construction of such witnesses will give rise to *piecewise linear* surfaces, because we approximate a witness by linear interpolating between the heights at the corners of any n-dimensional hyper-rectangle defined in the domain. Besides, the resulting $(n + 1)$-dimensional surfaces (hyper-planes) will be kinked when transitioning between adjacent hyper-rectangles in the domain.

It should be noted that choosing rectangular derivative information may result in some loss of information, but it will vastly simplify the LP algorithms and provides a practical framework for implementation. The more challenging problem of convex derivative information will also be analysed in a simple 2-dimensional setting, for which a linear test can also be developed.

Let us now state the consistency relation more formally, as presented in [2], using the terminologies developed so far. Later, we will look at the algorithms that decide consistency, firstly in the rectangular setting for the 1D and nD cases, $n \geq 2$ (as shown in [1] and [2], respectively).

Definition 3.2.1. The consistency relation $\mathsf{Cons} \subset D^0(U) \times (U \to \mathbf{C}\mathbb{R}^n)$ is defined by:

$$(f, g) \in \mathsf{Cons} \text{ if } \uparrow f \cap \int g \neq \emptyset.$$

Also the least and greatest consistency witnesses can be defined through the following:

Proposition 3.2.1. *Let O be a connected component $dom(g)$ and let $R(U)$ be the set of partial maps of U into the extended real line $\mathbb{R} \cup \{-\infty, \infty\}$. Consider the two dcpos $(R(U), \leq)$ and $(R(U), \geq)$ having pointwise ordering inherited from the extended real line. Then the maps $s : D^0(O) \times (U \to \mathbf{C}\mathbb{R}^n) \to (R(U), \leq)$ and $t : D^0(O) \times (U \to \mathbf{C}\mathbb{R}^n) \to (R(U), \geq)$ defined by:*

$$s(f,g) = \inf \left\{ h : dom(g) \to \mathbb{R} \;\middle|\; h \in \int g, h \geq f^- \right\}$$

$$t(f,g) = \sup \left\{ h : dom(g) \to \mathbb{R} \;\middle|\; h \in \int g, h \leq f^+ \right\}$$

represent the least primitive map of g that is greater than the lower part of f and the greatest primitive map of g that is less than the upper part of f, respectively. We also get from here that the following 3 conditions are equivalent:

1. $(f,g) \in \mathsf{Cons}$*;*
2. $s(f,g) \leq t(f,g)$*;*
3. *There exists a locally Lipschitz function $h : dom(g) \to \mathbb{R}$ with $g \sqsubseteq \mathcal{L}h$ and $f \sqsubseteq h$ on $dom(g)$.*

Furthermore, the maps s and t are Scott continuous and the relation Cons *is Scott closed (see [2]).*

Corollary 3.2.0.1. *Let $(f,g) \in \mathsf{Cons}$. Then in each connected component O of the domain of definition of g which intersects the domain of definition of f there exist two locally Lipschitz functions $s(f,g) : O \to \mathbb{R}$ and $t(f,g) : O \to \mathbb{R}$ such that $s(f,g), t(f,g) \in \uparrow f \cap \int g$ and for each $w \in \uparrow f \cap \int g$, it holds that:*

$$s(f,g)(x) \leq w(x) \leq t(f,g)(x),$$

for all $x \in O$.

3.2.1 Consistency for the one-dimensional case

We will now explain the framework for consistency in the case when the domain is unidimensional. For simplicity, we can assume, without loss of generality, that the domain $U = [0,1]$. Otherwise, one can easily convert a closed interval $[a,b]$ into $[0,1]$ upon a simple rescaling of the axis. Notice that the range of the derivative map g will be $\mathbf{I}\mathbb{R}$, as we are only dealing with the derivative concerning a single variable.

Let us consider a partition $0 = y_0 < y_1 < \ldots < y_n = 1$ of the domain $[0,1]$ in which interval-valued functions $f, g : [0,1] \to \mathbf{I}\mathbb{R}$, representing approximations for function and derivative information, respectively, are given. Then the following algorithm (which appears in Section 3 from [1]) with linear

complexity in the number of partitions induced by (f, g) can be developed as a test for consistency and also for determining the least map $s(f, g) \equiv w_{\min}$ which is also a witness to consistency. A analogous algorithm computes the greatest witness, i.e. $t(f, g) \equiv w_{\max}$.

Algorithm 3.2.1. *The function updating algorithm consists of an initialisation step and two other main steps (see Figure 3.1). The initialisation process determines the common partition points* $\{y_0, \ldots y_n\}$ *of* (f, g)*. On each interval* (y_{k-1}, y_k)*, the functions* g^- *and* g^+ *are constant, with* $g^-|_{(y_{k-1},y_k)} = \lambda t.e_k^-$ *and* $g^+|_{(y_{k-1},y_k)} = \lambda t.e_k^+$*, where* $e_k^-, e_k^+ \in \mathbb{R}$*. Furthermore, on each interval* (y_{k-1}, y_k)*, the map* f^- *has a constant slope,* a_k *say, i.e.* $f^-|_{(y_{k-1},y_k)} = f_k^-$*, with* $f_k^-(x) = a_k x + b_k$*.*

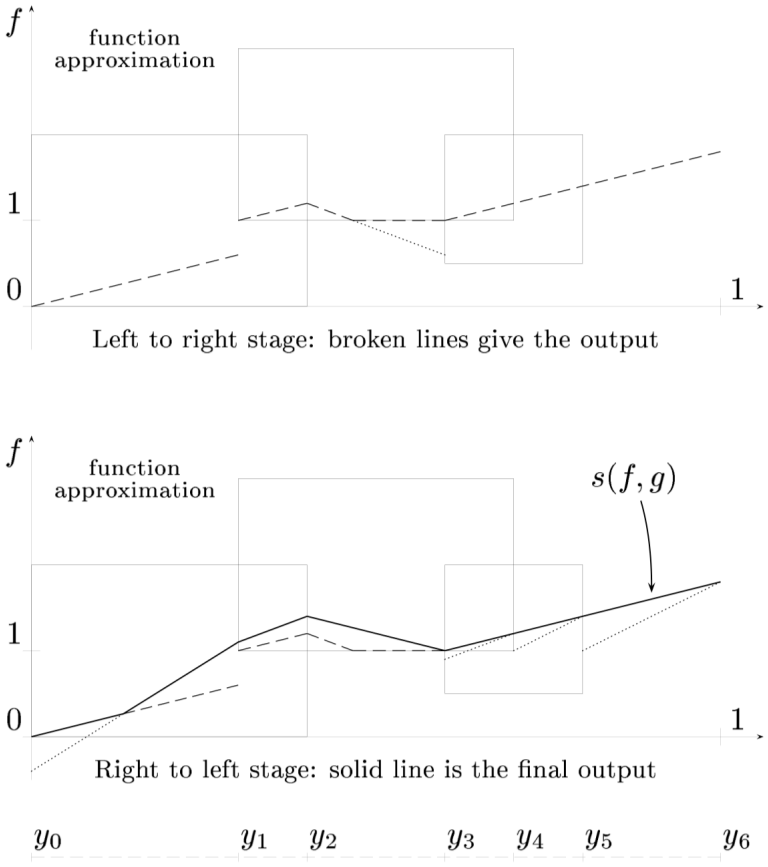


Figure 3.1: The function updating algorithm

Input: $f, g : [0,1] \to \mathbf{I}\mathbb{R}$, where f is a linear step function and g is a step function.

Output: Continuous function $s(f,g) : [0,1] \to \mathbf{I}\mathbb{R}$ which represents the least function consistent with the information from f and g.

Initialisation:
$\{y_0, \ldots y_n\}$: `induced-partition-of` (f,g)

Part 1:
$u(y_0) := f^-(y_0^+)$
`for` $k = 1 \ldots n$ and $\forall x \in [y_{k-1}, y_k)$
$u(x) := \max\{f^-(x), u(y_{k-1}) + (x - y_{k-1})e_k^-\}$
$u(y_k) := \max\{\overline{\lim} f^-(y_k), u(y_{k-1}) + (y_k - y_{k-1})e_k^-\}$
Part 2:
$s(y_n) := u(y_n)$
`for` $k = n \ldots 1$ and $\forall x \in [y_{k-1}, y_k)$
$s(f,g)(x) := \max\{u(x), s(y_k) + (x - y_{k-1})e_k^+\}$

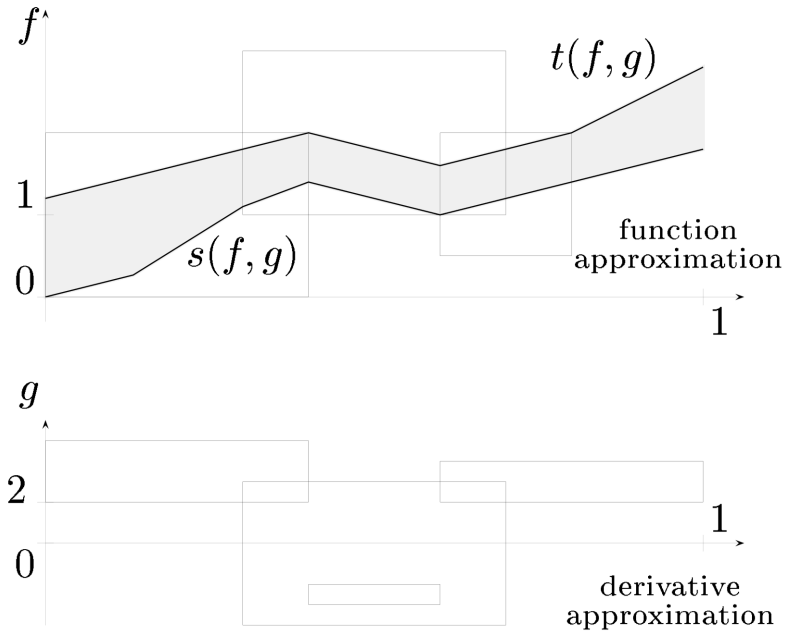


Figure 3.2: Consistency for the 1D case. The least witness s and the greatest witness to consistency t are shown (they are both made up of piecewise-linear segments). Note that any other witness can vary within the shaded region.

3.2.2 Consistency for the n-dimensional case, $n \geq 2$

After getting a flavour of the algorithm in the one-dimensional setup, we can now proceed to the central piece of the present work by explaining the general framework that decides consistency in higher dimensional spaces. As with the previous case, we will assume, for convenience, that the derivative information is rectangular, that is, contained within hyper-rectangles (as we are referring to higher dimensional spaces).

We will begin our analysis with the case $n = 2$, as it provides a simpler way to reason about the geometric structure of the problem. The algorithm for this case is described in Section 3 from [2], but we will follow a much simpler geometric approach to derive the linear programming algorithm (without using theory of cones). Afterwards, it will become straightforward to generalise the algorithm for any arbitrary $n \geq 2$.

Let us therefore consider consider the two-dimensional case. We will assume, as in the uni-dimensional setting, that each variable is constrained to lie within the closed interval $[0, 1]$ and as such we can consider $U = [0, 1]^2$ as the domain of definition. Now, both the function approximation f and derivative approximation g will be defined for rational rectangles inside U, so we can therefore enforce a grid $(p_0, p_1, \ldots, p_{k-1}) \times (q_0, q_1, \ldots, q_{l-1})$ within the unit square, where all points p_i and q_j lie on the x and y axis, respectively, $i = \overline{0, k-1}$, $j = \overline{0, l-1}$. By this construction, we have: $p_0 = q_0 = 0$ and $p_{k-1} = q_{l-1} = 1$. Then, for every sub-rectangle $R_{ij} = (p_i, p_{i+1}) \times (q_j, q_{j+1})$ formed by adjacent grid points, the functions $f : U \to \mathbf{I}\mathbb{R}$ and $g : U \to \mathbf{I}\mathbb{R}^2$ are given as follows:

$$f|_{R_{ij}} = \left[c_{ij}^-, c_{ij}^+\right] = c_{ij} \in \mathbf{I}\mathbb{R} \tag{3.1}$$

$$g|_{R_{ij}} = b_{ij}^1 \times b_{ij}^2 = b_{ij} \in \mathbf{I}\mathbb{R}^2, \tag{3.2}$$

for all $i = \overline{0, k-2}$, $j = \overline{0, l-2}$. Note that each b_{ij}^k, $k \in \{1, 2\}$ is also an interval, as we restrict the value of each partial derivative to lie within a closed and compact interval, so: $b_{ij}^k = \left[b_{ij}^{k-}, b_{ij}^{k+}\right]$ for every k.

Therefore, the problem of checking consistency for the pair of step functions (f, g) in the two-dimensional case reduces to determining the existence of heights $h_{ij} \equiv h((p_i, q_j))$ at all the grid points (p_i, q_j), $i = \overline{0, k-1}$, $j = \overline{0, l-1}$, subject to the given constraints (3.1) and (3.2).

Let us first tackle the constraints for function information. By (3.1) we see that all 4 heights at the corners of any of the $k \times l$ sub-rectangles R_{ij} must lie within $c_{ij} \in \mathbf{I}\mathbb{R}$ and hence:

$$c_{ij}^{-} \leq h_{st} \leq c_{ij}^{+}, \text{ for } s \in \{i, i+1\}, t \in \{j, j+1\} \tag{3.3}$$

where $i = \overline{0, k-2}$, $j = \overline{0, l-2}$. An observant reader would now notice that most of the inequalities (3.3) have the potential to overlap, due to the fact that a height h_{ij} can de defined for at most 4 sub-rectangles at a time (e.g. h_{11} is constrained by c_{00}, c_{10}, c_{01} as well as c_{11}). Therefore, we need to account for 3 types of grid points (p_i, q_j): corner, border and interior (see Figure 3.3 below), so that, upon intersecting all possible constraints, we can derive the tightest bounds for each height h_{ij}.

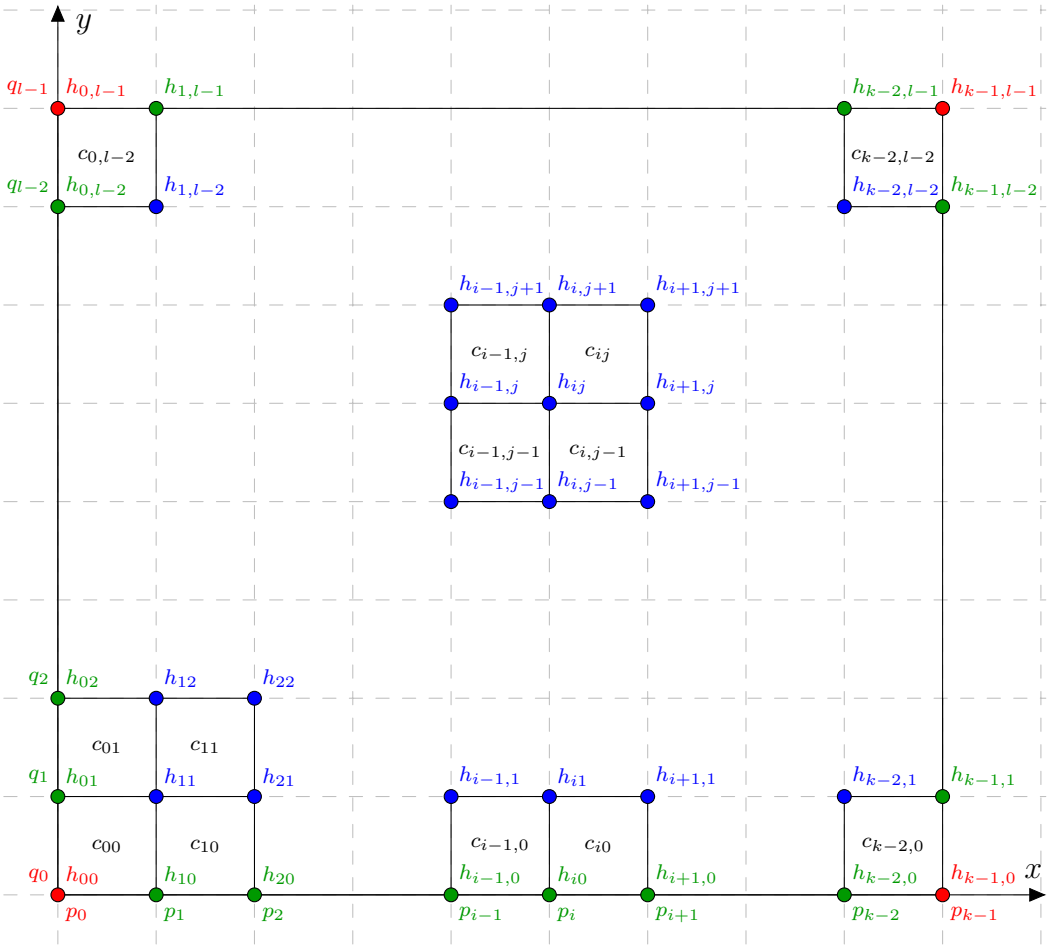


Figure 3.3: Function information for each sub-rectangle of the grid.

We now consider each case in turn:

1. If (p_i, q_j) is a corner point of the grid, then h_{ij} will only use the constraint within its own sub-rectangle, namely:

$$\begin{aligned} c^-_{00} &\leq h_{00} \leq c^+_{00} \\ c^-_{k-1,0} &\leq h_{k-1,0} \leq c^+_{k-1,0} \\ c^-_{0,l-1} &\leq h_{0,l-1} \leq c^+_{0,l-1} \\ c^-_{k-1,l-1} &\leq h_{k-1,l-1} \leq c^+_{k-1,l-1}. \end{aligned} \tag{3.4}$$

2. Now, if (p_i, q_j) is a border point of the grid, then, apart from the constraint defined for its sub-rectangle, h_{ij} will also require the constraint from the adjacent sub-rectangle. For example, consider the grid point (p_i, q_0) along the x axis. Then the constraint for h_{i0} will be derived by intersecting the constraints for the sub-rectangles $R_{i-1,0}$ and $R_{i,0}$. We can proceed similarly for all the other border points along the sides of the unit-square to get the following inequalities:

$$\begin{aligned} \max\left\{c^-_{i-1,0}, c^-_{i0}\right\} &\leq h_{i0} \leq \min\left\{c^+_{i-1,0}, c^+_{i0}\right\} \\ \max\left\{c^-_{i-1,l-1}, c^-_{i,l-1}\right\} &\leq h_{i,l-1} \leq \min\left\{c^+_{i-1,l-1}, c^+_{i,l-1}\right\} \\ \max\left\{c^-_{0,j-1}, c^-_{0j}\right\} &\leq h_{0j} \leq \min\left\{c^+_{0,j-1}, c^+_{0j}\right\} \\ \max\left\{c^-_{k-1,j-1}, c^-_{k-1,j}\right\} &\leq h_{k-1,j} \leq \min\left\{c^+_{k-1,j-1}, c^+_{k-1,j}\right\}, \end{aligned} \tag{3.5}$$

   where $i = \overline{1, k-2}$ and $j = \overline{1, l-2}$.

3. Finally, if (p_i, q_j) is an interior point of the grid, then in order to derive the best bounds for h_{ij} we will need to account for all four adjacent sub-rectangles with common vertex (p_i, q_j). As such, we can optimise the constraints in this case by taking the intersection of the intervals $c_{i-1,j-1}$, $c_{i,j-1}$, $c_{i-1,j}$ and c_{ij}, respectively, so we can thus write:

$$\max\left\{c^-_{i-1,j-1}, c^-_{i,j-1}, c^-_{i-1,j}, c^-_{ij}\right\} \leq h_{ij} \leq \min\left\{c^+_{i-1,j-1}, c^+_{i,j-1}, c^+_{i-1,j}, c^+_{ij}\right\},$$

   for all $i = \overline{1, k-2}$ and $j = \overline{1, l-2}$. (3.6)

In the analysis developed so far, we made the tacit assumption that all adjacent intervals c_{ij} have non-empty intersection. This is an absolute requirement for consistency to hold, because otherwise the rectangular parallelepipeds defined by adjacent sub-rectangles in the grid would not intersect and will naturally give rise to discontinuities. If that is the case, then the function information will be classified as being inconsistent. In equivalent terms, inconsistent function information will translate to at least one of the inequalities from above being false (in such a scenario the greatest lower bound for some h_{ij} will be strictly greater than its lowest upper bound, contradiction).

Having broken down the function information constraints into inequalities of the type: `lower_bound` $\leq h_{ij} \leq$ `upper_bound`, for all suitable indices i and j, let us now focus our attention on the derivative constraints given by function g.

Consider an arbitrary sub-rectangle with lower left corner starting at (p_i, q_j) in the square U, where $1 \leq i \leq k-2$, $1 \leq j \leq l-2$. The corresponding heights at each of the 4 corners of the sub-rectangle R_{ij} are, respectively, h_{ij}, $h_{i+1,j}$, $h_{i,j+1}$, $h_{i+1,j+1}$, all bounded within the rectangular box with height $c_{ij}^{+} - c_{ij}^{-}$ (see the diagram from Figure 4.2). Since a consistent witness h needs to pass through all the 4 heights h_{st}, where $s \in \{i, i+1\}$, $t \in \{j, j+1\}$, it means that the resulting surface contained inside the rectangular parallelepiped will be piecewise-linear and can be obtained by interpolating the heights as follows:

- In the triangle with vertices (p_i, q_j), (p_i, q_{j+1}) and (p_{i+1}, q_j), the map h linearly interpolates between the values h_{ij}, $h_{i,j+1}$ and $h_{i+1,j}$ at these vertices respectively.

- In the triangle with vertices (p_{i+1}, q_{j+1}), (p_i, q_{j+1}) and (p_{i+1}, q_j), the map h linearly interpolates between the values $h_{i+1,j+1}$, $h_{i,j+1}$ and $h_{i+1,j}$ at these vertices respectively.

This construction is depicted in 4.2. Alternatively, the piecewise linear surface can be constructed by interpolating the values given at the vertices of the other two triangles, that is, one with vertices (p_i, q_j), (p_i, q_{j+1}) and (p_{i+1}, q_{j+1}), and the other with vertices (p_i, q_j), (p_{i+1}, q_j) and (p_{i+1}, q_{j+1}) – from a geometric perspective, this means that the slanted line of intersection between the two triangular surfaces would join h_{ij} and $h_{i+1,j+1}$ as opposed to $h_{i,j+1}$ and $h_{i+1,j}$.

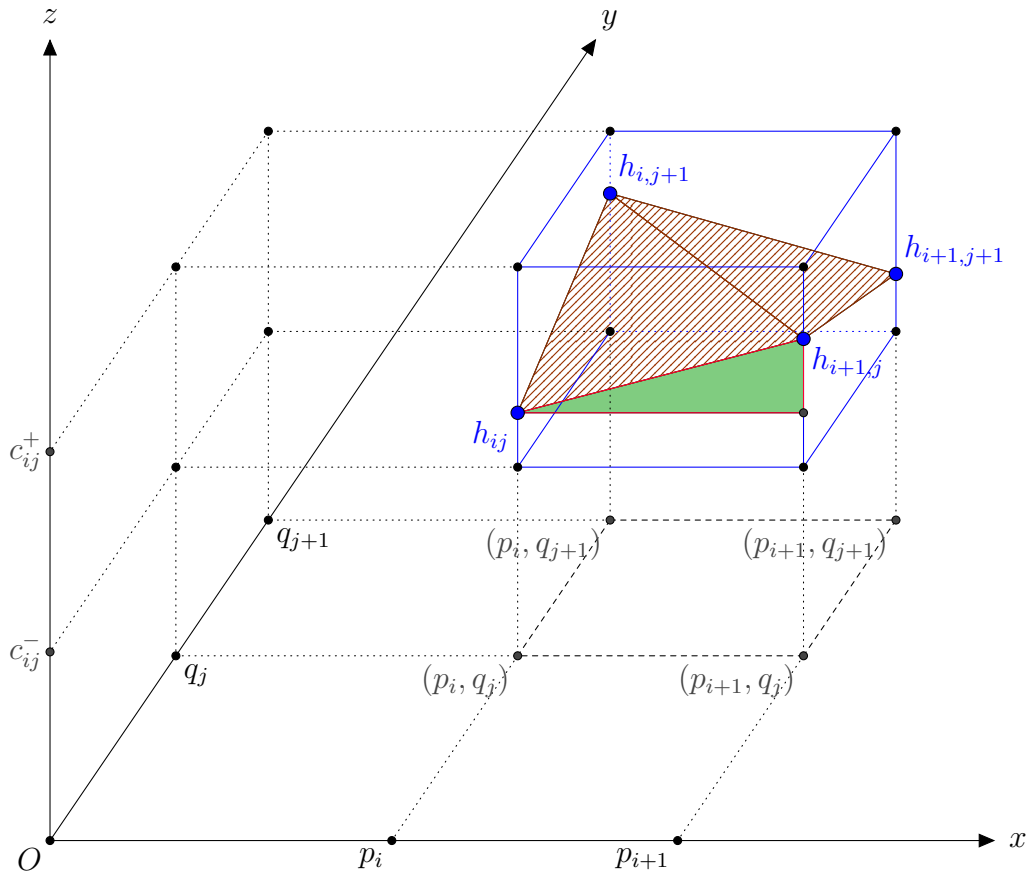


Figure 3.4: Piecewise-linear surfaces obtained by interpolating along the heights of a consistent witness.

Hence, the piecewise linear map constructed above (in either scenario) satisfies the derivative constraints if and only if the slopes of each of the 4 lines obtained by joining adjacent heights lie within the appropriate part of $b_{ij} = b_{ij}^1 \times b_{ij}^2 \in \mathbf{I}\mathbb{R}^2$. For example, the slope of the line passing through the points $((p_i, q_j), h_{ij})$ and $((p_{i+1}, q_j), h_{i+1,j})$ must be included in the interval b_{ij}^1 corresponding to the partial derivative with respect to x (see the green triangle from Figure 4.2). Similarly, the slope of the line joining $((p_i, q_{j+1}), h_{i,j+1})$ and $((p_{i+1}, q_{j+1}), h_{i+1,j+1})$ must belong to b_{ij}^1, so we can thus write:

$$b_{ij}^{1-} \leq \frac{h_{i+1,j} - h_{ij}}{p_{i+1} - p_i} \leq b_{ij}^{1+} \qquad b_{ij}^{1-} \leq \frac{h_{i+1,j+1} - h_{i,j+1}}{p_{i+1} - p_i} \leq b_{ij}^{1+}, \tag{3.7}$$

where $i = \overline{0, k-2}$ and $j = \overline{0, l-1}$ (but in the second case we require that $j = \overline{0, l-2}$).

In a similar fashion, the constraints for the partial derivative with respect to y translate to:

$$b_{ij}^{2-} \leq \frac{h_{i,j+1} - h_{ij}}{q_{i+1} - q_i} \leq b_{ij}^{2+} \qquad b_{ij}^{2-} \leq \frac{h_{i+1,j+1} - h_{i+1,j}}{q_{i+1} - q_i} \leq b_{ij}^{2+}, \tag{3.8}$$

where $i = \overline{0, k-1}$ and $j = \overline{0, l-2}$ (but $i = \overline{0, k-2}$ in the second case).

We can already notice that the inequalities given by (3.7) and (3.8) feature the same kind of overlapping structure as with the inequalities (3.3) defined within each sub-rectangle R_{ij}. As a result, there is a potential for optimising these constraints in the same manner as we did earlier for the function approximation (see Figure 3.5 below).

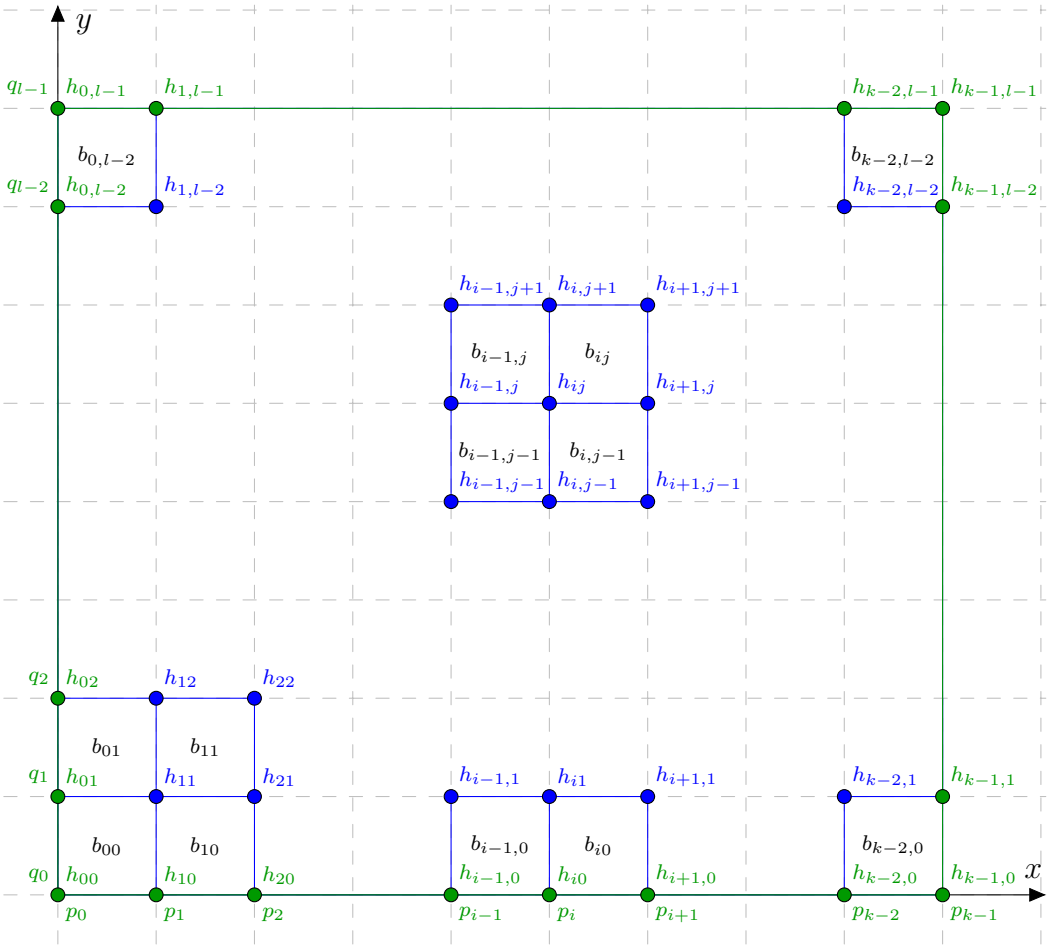


Figure 3.5: Derivative information for each sub-rectangle of the grid.

The difference here is that we only need to account for interior sides of sub-rectangles, because the constraint coming from an adjacent sub-rectangle may potentially contribute towards improving the bounds of a slope of type $(h_{i+1,j} - h_{ij})/(p_{i+1} - p_i)$. On the other hand, the slopes defined along the border edges of the grid will only use the constraint within their corresponding sub-rectangle.

By writing the inequalities in full and taking great care of indices, we get in each case that:

1. The slopes along the border sides of sub-rectangles are simply:

$$\begin{aligned} b_{i0}^{1-} &\leq \frac{h_{i+1,0} - h_{i0}}{p_{i+1} - p_i} \leq b_{i0}^{1+} \\ b_{i,l-2}^{1-} &\leq \frac{h_{i+1,l-1} - h_{i,l-1}}{p_{i+1} - p_i} \leq b_{i,l-2}^{1+} \\ b_{0j}^{2-} &\leq \frac{h_{0,j+1} - h_{0j}}{q_{j+1} - q_j} \leq b_{0j}^{2+} \\ b_{k-2,j}^{2-} &\leq \frac{h_{k-1,j+1} - h_{k-1,j}}{q_{j+1} - q_j} \leq b_{k-2,j}^{2+}, \end{aligned} \tag{3.9}$$

   where $i = \overline{0, k-2}$ and $j = \overline{0, l-2}$.

2. The slopes along the interior sides of sub-rectangles need also to take into account that the same slope occurs in an adjacent sub-rectangle and hence we get the following inequalities:

$$\begin{aligned} \max\left\{b_{i,j-1}^{1-}, b_{ij}^{1-}\right\} &\leq \frac{h_{i+1,j} - h_{ij}}{p_{i+1} - p_i} \leq \min\left\{b_{i,j-1}^{1+}, b_{ij}^{1+}\right\} \\ \max\left\{b_{i-1,j}^{2-}, b_{ij}^{2-}\right\} &\leq \frac{h_{i,j+1} - h_{ij}}{q_{j+1} - q_j} \leq \min\left\{b_{i-1,j}^{2+}, b_{ij}^{2+}\right\}, \end{aligned} \tag{3.10}$$

   where $i = \overline{0, k-2}$ and $j = \overline{1, l-2}$ for the first chain of inequalities, while for the second we impose $i = \overline{1, k-2}$ and $j = \overline{0, l-2}$.

From the discussion developed so far, we observe that all the inequalities given by (3.4), (3.5), (3.6), (3.9) and (3.10) represent, in fact, necessary and sufficient conditions for deciding whether a pair of functions (f, g) is consistent or not. We can therefore introduce the following:

Theorem 3.2.1. *(Decidability of consistency) A pair* $(f, g) \in (U \to \mathbf{I}\mathbb{R}) \times (U \to \mathbf{I}\mathbb{R}^2)$*, representing function and derivative information respectively, is consistent if and only if we can find values* $h_{ij} \in \mathbb{R}$ *at the grid points* (p_i, q_j) *such that the inequalities* (3.4)*,* (3.5)*,* (3.6)*,* (3.9) *and* (3.10) *are simultaneously satisfied.*

Since f and g are given in terms of rational numbers, the question of consistency boils down to solving a finite set of inequalities with rational coefficients for the $k \times l$ unknowns h_{ij}, which is decidable. In fact, it represents the set of constraints for a linear programming problem in which only a feasible solution is required to be found in order to guarantee the existence of a consistent witness.

Finally, we will show that the construction of the minimal and maximal surfaces which are also witnesses to consistency can be derived upon minimising or maximising $\sum_{\substack{0 \le i \le k-1 \\ 0 \le j \le l-1}} h_{ij}$, respectively. Let us therefore present:

Algorithm 3.2.2. *(Minimal consistent witness) For a consistent pair* $(f, g) \in (U \to \mathbf{I}\mathbb{R}) \times (U \to \mathbf{I}\mathbb{R}^2)$*, representing function and derivative information respectively, consider the following linear programming problem:*

$$\begin{aligned} &\text{Minimise} && \sum_{\substack{0 \le i \le k-1 \\ 0 \le j \le l-1}} h_{ij} \\ &\text{subject to} && \text{Constraints (3.4), (3.5), (3.6), (3.9) and (3.10),} \end{aligned} \tag{3.11}$$

which minimises the sum of heights at each of the given grid points (p_i, q_j) *in the unit square. Then the linear programming problem* (3.11) *has an optimal solution given by:*

$$w_{\min} = \left\{ h^\star_{ij} \mid 0 \le i \le k-1, 0 \le j \le l-1 \right\}$$

and furthermore $w_{\min} \le w$*, for any other consistent witness* w*.*

Proof. Let W be the set of all witnesses consistent with the pair (f, g). By Theorem 3.2.1 we clearly have that $W \neq \emptyset$. According to Corollary 6.4 from [2], the minimal and maximal surfaces are guaranteed to exist for a consistent pair (f, g). Thus, $w^\star = \inf W$ exists and is a witness. In particular, the heights $h^\star_{ij}$ of $w^\star$ are optimal and will therefore minimise the objective function. Now, if $w_{\min} \le w^\star$ then there would be some $h_{ij} < h^\star_{ij}$, thus contradicting the optimality of $h^\star_{ij}$ and also the optimality of the objective function. Consequently, $w^\star = w_{\min}$, as desired. □

A similar recipe applies for deriving the algorithm that will determine the greatest consistent witness $w_{\max}$, as we only need to convert (3.11) to become a maximisation problem:

Algorithm 3.2.3. *(Maximal consistent witness) For a consistent pair* $(f, g) \in (U \to \mathbf{I}\mathbb{R}) \times (U \to \mathbf{I}\mathbb{R}^2)$*, representing function and derivative information respectively, consider the following linear programming problem:*

$$\begin{aligned} &\text{Maximise} && \sum_{\substack{0 \le i \le k-1 \\ 0 \le j \le l-1}} h_{ij} \\ &\text{subject to} && \text{Constraints (3.4), (3.5), (3.6), (3.9) and (3.10),} \end{aligned} \tag{3.12}$$

which maximises the sum of heights at each of the given grid points (p_i, q_j) *in the unit square. Then the linear programming problem* (3.11) *has an optimal solution given by:*

$$w_{\max} = \left\{ h^{\star}_{ij} \mid 0 \le i \le k-1, 0 \le j \le l-1 \right\}$$

and furthermore $w \le w_{\max}$*, for any other consistent witness* w*.*

The above algorithms (3.11) and (3.12), together with the Theorem of decidability of consistency 3.2.1 can now be easily extended to an n-dimensional setting. We consider the unit cube $U \subset \mathbb{R}^n$ and a number of points along each of the n edges intersecting at the origin of the cube so that $f \in \mathbf{I}\mathbb{R}$ and $g \in \mathbf{I}\mathbb{R}^n$ are constant in each of the resulting sub-hyper-rectangles. We then derive similar constraints to (3.4), (3.5) and (3.6) from the function approximation by intersecting all the overlapping intervals at grid-points contained in adjacent sub-hyper-rectangles. Similar to the inequalities (3.9) and (3.10), we need to account for each of the n partial derivatives when constructing the derivative constraints – the slope between heights at adjacent grid-points must lie within the intersection of all the intervals at sub-hyper-rectangles that contain the edges defined by neighbouring grid-points.

Finally, the algorithms for minimising and maximising the least and greatest consistent witnesses are identical to the ones developed for the 2D case: we require that sum of heights for all grid-points to be either minimised or maximised, respectively, subject to the constraints derived from the pair $(f, g) \in (U \to \mathbf{I}\mathbb{R}) \times (U \to \mathbf{I}\mathbb{R}^n)$.

3.2.3 Consistency for a triangle with convex derivative information in the two-dimensional case

Let us now examine a more general framework for deciding consistency, which is presented in [14]. Suppose we have a non-empty convex and compact polygon B of the plane as the derivative information. Assume three distinct points $v_i = (x_i, y_i)$ with $i \in \{1, 2, 3\}$ and T_{123}, or simply T, is the closed region defined by these three points. Suppose that we have $h_i \in \mathbb{R}$ for $i = \{1, 2, 3\}$. We aim to establish if there is a Lipschitz witness $z = w(x, y)$ with $w : T \to \mathbb{R}$, that goes through the three points (v_i, h_i), for $i \in \{1, 2, 3\}$, whose derivative everywhere is contained in B. Let $z = P(x, y) = \alpha x + \beta y + \gamma$ with $P : T \to \mathbb{R}$ be the plane that goes through three points (v_i, h_i), for $i \in \{1, 2, 3\}$ with $\nabla P = (\alpha, \beta) = b$. Then, w satisfies our requirements if and only if $w - P$ goes through $(v_i, 0)$, for $i = \{1, 2, 3\}$, with its derivative consistent with $B - b$. Thus, we can equivalently consider the latter problem of finding a Lipschitz maps that goes through $(x_i, y_i, 0)$, for $i \in \{1, 2, 3\}$, with its derivative contained everywhere in $B' \equiv B - b$.

Consider $e_{ij} \equiv v_j - v_i$, with ij in the cyclic order $1, 2, 3$ and let $e_{ij}^{\perp}$ be unit vector orthogonal to e_{ij} in the direction into the triangle T. Now, by the mean value theorem (MVT) for Lipschitz maps, for a witness to exist, it is necessary that:

$$0 \in B' \cdot e_{ij}$$

for all distinct pairs $i, j \in \{1, 2, 3\}$. Assume therefore that these conditions, called the MVT conditions, hold. Thus, there exist $b_{ij} \in B'$ for cyclic ordered pairs ij such that $b_{ij} \cdot e_{ij} = 0$, i.e., $b_{ij} = k e_{ij}^{\perp}$ for some $k \in \mathbb{R}$. Let $[b_{ij}^{-}, b_{ij}^{+}]$ be the interval along the direction $e_{ij}^{\perp}$ that is contained in B. Let P_{ij}^{-} be the plane with $\nabla P_{ij}^{-} = b_{ij}^{-}$ that contains e_{ij} for each cyclic order ij.

Proposition 3.2.2. *[14] Suppose the MVT conditions hold. Then, there is Lipschitz map $w^{\star} : T \to \mathbb{R}$ that goes through the three points (x_i, y_i, h_i), with $i = 1, 2, 3$ and whose derivative is contained in B, such that for every other witness w with these properties we have:*

$$\min w \leq w^{\star} \leq \max w.$$

Proof. If $b \in B$ (i.e. $0 \in B'$) then the plane $w^{\star} = P$ satisfies our conditions; see the thick dashed triangle in the figure. Suppose, therefore, that $b \notin B$. By considering witnesses of the form $w - P$ we can equivalently consider the reduced problem with $h_i = 0$ and $B' = B - b$. Then $0 \notin B'$. In particular, the convex hull of the three segments $[b_{ij}^{-}, b_{ij}^{+}]$ along $e_{ij}^{\perp}$ does not contain 0.

This means that there exist two pairs i_1i_2 and i_3i_1 such that precisely one of the following two conditions hold:

- Both $b^-_{i_1i_2}$ and $b^-_{i_3i_1}$ have positive components along $e^\perp_{i_1i_2}$ and $e^\perp_{i_3i_1}$ respectively while $b^+_{i_2i_3}$ has negative component on $e^+_{i_2i_3}$.
- Both $b^-_{i_1i_2}$ and $b^-_{i_3i_1}$ have negative components along $e^\perp_{i_1i_2}$ and $e^\perp_{i_3i_1}$ respectively while $b^+_{i_2i_3}$ has positive component on $e^+_{i_2i_3}$.

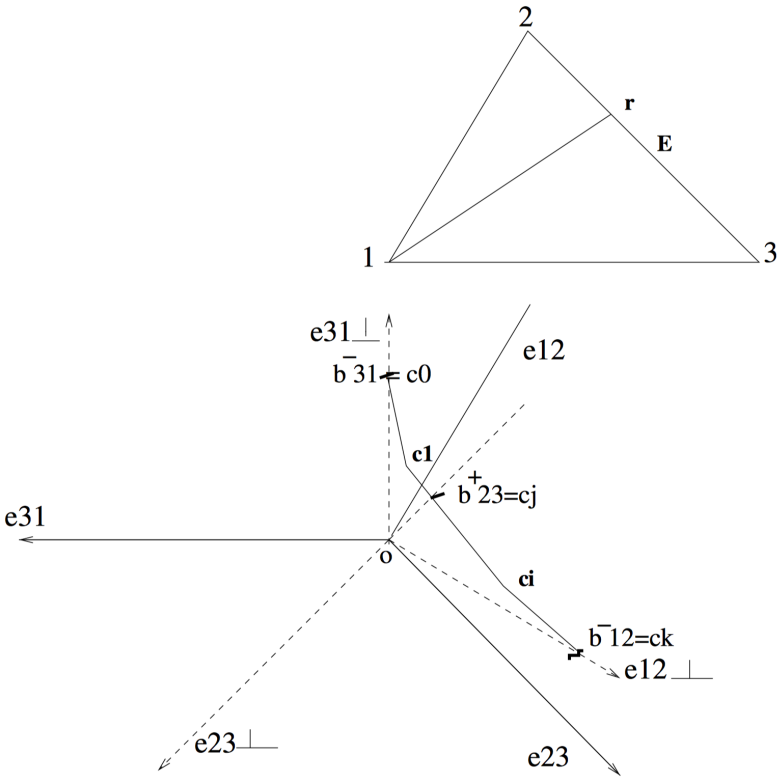


Figure 3.6: Consistency for triangle

We consider the first case depicted in Figure 3.6 as the second is similar. Therefore, assume that there exist two pairs i_1i_2 and i_3i_1 such that $b^-_{i_1i_2}$ and $b^-_{i_3i_1}$ have positive components along $e^\perp_{i_1i_2}$ and $e^\perp_{i_3i_1}$ respectively while $b^+_{i_2i_3}$ has negative component on $e^\perp_{i_2i_3}$.

Let $c_0 \equiv b^-_{i_3i_1}, c_1, \ldots, c_{k-1}, c_k \equiv b^-_{i_1i_2}$ be the vertices of B' from $b^-_{i_3i_1}$ to $b^-_{i_1i_2}$ on the same side of the origin with respect to the line l that goes through $b^-_{i_3i_1}$ and $b^-_{i_1i_2}$. For any $c \in \mathbb{R}^2$ let $z = P_c(x, y) = c_1x + c_2y + \gamma_c$ be the plane

through (v_i, h_i) with $\nabla P_c = (c_1, c_2) = c$. Therefore, using our previous notation, we have $P_{c_0} = P_{i_3 i_1}$ and $P_{c_k} = P_{i_1 i_2}$. Let $w^\star \equiv \min\left\{P^-_{c_j} : 0 \leq j \leq k\right\}$.

Now let w be any Lipschitz map through the three points $v_i = (x_i, y_i, 0)$ for $i \in \{1, 2, 3\}$ and consistent with B'. Then, w is differentiable almost everywhere, and is equal to the integral of its derivative. Consider any path $p : [0, 1] \to T$, given by $p(t) = v_{i_1} + t \cdot (r - v_{i_1})$ from vertex v_{i_1} to a point r on the opposite edge E of T. Then $\inf B' \cdot (r - v_{i_1}) = c_j \cdot (r - v_{i_1})$ for some j with $0 \leq j \leq k$ and $w^\star|_{[v_{i_1}, r]} = P_{c_j}$, where $[v_{i_1}, r]$ is the line segment from v_{i_1} to r in T. Thus we immediately obtain that:

$$\begin{aligned} w(r) - w(v_{i_1}) &= \int_0^1 w'(v_{i_1} + t(r - v_{i_1})) \cdot (r - v_{i_1}) \, \mathrm{d}t \\ &\geq \int_0^1 c_j \cdot (r - v_{i_1}) \, \mathrm{d}t \\ &= \int_0^1 w^\star(v_{i_1} + t(r - v_{i_1})) \cdot (r - v_{i1}) \, \mathrm{d}t \\ &= w^\star(r) - w^\star(v_{i_1}) \end{aligned}$$

On the other hand, $w(v_{i_1}) = w^\star(v_{i_1}) = h_{i_1} = 0$. But $\max w^\star = \max_{r \in E} w^\star(r)$ and hence:

$$\max w \geq \max_{r \in E} w(r) \geq \max_{r \in E} w^\star(r) = \max w^\star$$

Consequently, $\max w \geq \max w^\star$ and since $\min w \leq \min\{h_i \mid i \in \{1, 2, 3\}\} = \min w^\star$, the result follows. □

Now recall the original problem with three points (x_i, y_i, h_i), for $i \in \{1, 2, 3\}$, and derivative information B. Suppose lower and upper limits $c^- \leq c^+$ are given. Let $P : T \to \mathbb{R}$ be the plane that goes through three points (x_i, y_i, h_i), for $i \in \{1, 2, 3\}$ with $\nabla P = b$ and put $B' = B - b$. Consider $w^\star$ constructed above.

Theorem 3.2.2. *[14] There is a witness to consistency if and only if the following two conditions hold:*

- *For all distinct pairs $i, j \in \{1, 2, 3\}$, we have: $0 \in B' \cdot e_{ij}$.*
- $c^- \leq w^\star + P \leq c^+$.

This shows that consistency is semi-decidable, that is, for any given $h = (h_1, h_2, h_3) \in \mathbb{R}^3$ we can decide if there is a witness for consistency with heights h_i at vertex v_i for $i \in \{1, 2, 3\}$. Returning to the original problem we can now state the following:

Theorem 3.2.3. *[14] Let rational points v_i for $i \in \{1, 2, 3\}$, forming a triangle in the plane, a rational convex polygon B and rational numbers $c^- \leq c^+$ be given. Then it is decidable that there exist heights h_i for which there exists a Lipschitz witness going through (v_i, h_i) consistent with B and the bound c^- and c^+ in the closed region bounded by the triangle, where $i \in \{1, 2, 3\}$.*

Proof. We first check if there exists $h \in \mathbb{R}^3$ such that the plane $z = P(x, y) = \alpha x + \beta y + \gamma$ going through the three points (v_i, h_i) for $i \in \{1, 2, 3\}$ has gradient $(\alpha, \beta) \in B$. If this condition holds then P is clearly a witness and we are done. Otherwise, we know from the construction presented in this section that there is a witness w if and only if $w^\star + P$ is a witness, where $w^\star$ is the piecewise linear surface constructed above using $B' = B - \nabla P$ and $z = P(x, y)$ is the plane passing through the three points (v_i, h_i), with $h_i = w(v_i)$ where $i \in \{1, 2, 3\}$. Let us now show that for each consecutive pair of vertices c_j and c_{j+1} in the construction of $w^\star$ above, the *slanted* line of intersection of the two planes P_{c_j} and $P_{c_{j+1}}$ – equivalently the line of intersection of the two planes $P'_j \equiv P_{c_j} + P$ and $P'_{j+1} \equiv P_{c_{j+1}} + P$ – is perpendicular to the line segment $c_j c_{j+1}$. Recall that all planes P_{c_j} pass through the point $(v, 0)$ with $v \equiv v_{i_1}$. This allows us to write that:

$$P_j(v) = P_{j+1}(v) = 0.$$

Now, for some $u = (x, y) \in T$ we can successively write that:

$$P'_j(u) = P'_{j+1}(u) \iff P_{c_j}(u) = P_{c_{j+1}}(u)$$

$$\iff P_{c_j}(u) - P_{c_j}(v) = P_{c_{j+1}}(u) - P_{j+1}(v)$$

$$\iff c_j \cdot (u - v) = c_{j+1} \cdot (u - v)$$

$$\iff (u - v) \cdot (c_j - c_{j+1}) = 0,$$

as claimed. It follows that the piecewise linear witness w will linearly interpolate in each of the triangles with a vertex at v_{i_1} and sides with common vertex v_{i_1} perpendicular to the faces $c_j c_{j+1}$ for $j = \overline{0, k}$. This gives a simple triangulation of T. Let the vertices of the all the triangles in the triangulation be denoted by $u_0, u_1, \ldots, u_N$ which includes the vertices v_1, v_2 and v_3 and suppose $t_j \equiv w(u_j)$ for $j = \overline{1, N}$. □

Chapter 4

Implementation

In this chapter we will present a detailed overview of the implementation and design decisions behind the linear test that decides consistency for the rectangular case in n-dimensions, as explained by Theorem 3.2.1. In the case when the test indicates consistency for a pair of functions (f, g), representing function and derivative information, respectively, we make use of the linear programming algorithms 3.11 and 3.12 to determine the minimal and maximal bounding surfaces. We also implemented a simple GUI that allows the user of this program to visualise the 3D piecewise-linear surfaces whenever the algorithm reports consistency for the two-dimensional case.

4.1 Tools

The entire codebase of the project has been written in the Python programming language. As mentioned in Section 2.3, this choice was motivated by the existence of a robust and developer-friendly linear programming framework called CVXOPT, which is implemented in Python. The language features both functional and object oriented programming styles that are expressed using a lightweight and intuitive syntax.

In addition to being a cross-platform language, Python is a very popular choice among programmers due to the plethora of readily-available packages which can be easily integrated in projects of any scale and complexity. Python also enjoys tremendous support from the academical community, which has boosted the development of a great deal of libraries. Apart from CVXOPT, we used a couple of other dependencies including:

- `numpy` [15]: scientific package which offers a variety of convenient features such as powerful n-dimensional arrays with flexibility for arbitrary data types, efficient numerical computation and random number capabilities.

- `sympy` [16]: written entirely in Python, this module enables effective *symbolic computation* for mathematical expressions. Moreover, it has excellent support for polynomials with an arbitrary number of variables

e.g. computing the value of a polynomial at a specific point, or the value of a derivative at any given point.

- `matplotlib` [17]: a library that is rather destined for 2D plots, but also features decent support for basic 3D graphs via the `mplot3d` toolkit. This package allowed very fast development of the front-end component of our framework, which will be detailed later in this chapter.

4.2 Back-end

We will now explain the two main components which make up the core implementation of the framework that decides consistency when the derivative approximation is constrained by hyper-rectangles. Despite having dealt with the 2D case in the previous chapter, the implementation below is described as much as possible for the general setting where the domain is n-dimensional. For the sake of clarity, we may occasionally illustrate with examples in the two-dimensional case whenever the general instance becomes notationally heavy or harder to reason about.

4.2.1 Input Format and Data Generation

In order to formulate the constraints for the linear programming problems given by Algorithms 3.11 and 3.12, we first need to specify three essential elements to our program: grid information, function information and derivative information respectively. For simplicity, these are all provided in a single file that will be loaded when the final program is run. We impose the following format for each element:

1. **Grid information**: this is given as n separate lines, corresponding to the n axis of the domain. Each line is represented as a sequence of strictly increasing rational numbers which specify the divisions within $[0, 1]$ (including the endpoints) along the i^{th} axis, $1 \leq i \leq n$. Thus, each line is assumed to contain at least 2 values.

   For presentation purposes, let us denote by $p_i \geq 2$ the number of points along the i^{th} axis so that any point of the grid can be written as $G_{k_1 k_2 \dots k_n}$, where $0 \leq k_i \leq p_i - 1$, for every $1 \leq i \leq n$ (e.g. $G_{00\dots0}$ will correspond to the origin O of $\mathbb{R}^n$, while $G_{p_1-1,\dots,p_n-1}$ will be mapped to $(1, 1, \dots, 1) \in \mathbb{R}^n$; more specifically, the i^{th} coordinate of such a point is given by the k_i^{th} rational value located on the i^{th} line above).

2. **Function information**: this is specified as an n-dimensional array of intervals, whose size $p_1 \times p_2 \times \ldots \times p_n$ is determined by the grid information provided earlier. An entry at an array index $(k_1, k_2, \ldots k_n)$, where $0 \leq k_i \leq p_i - 2$ is given as a pair of floating point numbers:

$$\left(c^{-}_{k_1 k_2 \ldots k_n}, c^{+}_{k_1 k_2 \ldots k_n}\right)$$

with $c^{-}_{k_1 k_2 \ldots k_n} \leq c^{+}_{k_1 k_2 \ldots k_n}$, such that it represents the closed and compact interval constraint inside the sub-hyper-rectangle with 2^n vertices $G_{k_1+b_1,\ldots,k_n+b_n}$, where $b_i \in \{0, 1\}$, for $1 \leq i \leq n$.

It is now crucial to observe that a point $G_{k_1 k_2 \ldots k_n}$ for which at least some $k_i = p_i - 1$, where $1 \leq i \leq n$, then the function information is already provided within one or more sub-hyper-rectangles that lie on the border of the unit-square $U = [0, 1]^n$. We will refer to such points as being *border* grid points (not to be confused with the interpretation given in Section 3.2.2 from Chapter 2, which is fundamentally different). For simplicity, we will provide a default pair of $(0, 0)$ at each of these $p_1 \times p_2 \times \ldots \times p_n - (p_1 - 1) \times (p_2 - 1) \times \ldots \times (p_n - 1)$ border grid-points so that the n-dimensional array is fully initialised.

Finally, in order to store the n-dimensional array within the input file, we use a convenient 2D matrix layout with $p_1 \times p_2 \times \ldots \times p_{n-1}$ rows and p_n columns.

3. **Derivative information**: this is also constructed as an n-dimensional array of size $p_1 \times p_2 \times \ldots \times p_n$ similar to the one presented above for the function information. The only distinction here is that each entry will be given as a tuple of n pairs, where each pair will represent a closed and compact interval constraint for each partial derivative within the corresponding sub-hyper-rectangle. Specifically, an entry belonging to the derivative information array at index $(k_1, k_2, \ldots k_n)$ features the following form:

$$\left(\left(b^{1-}_{k_1 k_2 \ldots k_n}, b^{1+}_{k_1 k_2 \ldots k_n}\right), \left(b^{2-}_{k_1 k_2 \ldots k_n}, b^{2+}_{k_1 k_2 \ldots k_n}\right), \ldots, \left(b^{n-}_{k_1 k_2 \ldots k_n}, b^{n+}_{k_1 k_2 \ldots k_n}\right)\right),$$

where $0 \leq k_i \leq p_i - 2$ and $b^{i-}_{k_1 k_2 \ldots k_n} \leq b^{i+}_{k_1 k_2 \ldots k_n}$ are given rational values for each $1 \leq i \leq n$. Similarly, for the indices corresponding to border grid points in the unit-square, we need to initialise the n-dimensional array with a default value – e.g. we can consider each element of the n-dimensional tuple to be the interval $(0, 0)$.

```
# Grid information:
0.0 0.6 1.0
0.0 1.0

# Function information:
# Array shape: (3, 2)
(2.89, 29.64) (0.0, 0.0)
(2.89, 29.73) (0.0, 0.0)
( 0.0,   0.0) (0.0, 0.0)

# Derivative information:
# Array shape: (3, 2)
((-21.16,  5.97), (-13.55, 10.76)) ((0.0, 0.0), (0.0, 0.0))
(( -5.55, 18.32), (-15.08,  6.44)) ((0.0, 0.0), (0.0, 0.0))
((   0.0,   0.0), (   0.0,   0.0)) ((0.0, 0.0), (0.0, 0.0))
```

Listing 4.1: Sample input file

A very minimalistic example is provided in Listing 4.1 where we considered three points along the first axis, and only the endpoints 0 and 1 along the second axis. Observe that for the border points previously defined as having at least one coordinate equal to 1, the default values of `(0.0, 0.0)` and `((0.0, 0.0), (0.0, 0.0))` are used for function and derivative information, respectively.

Although the format developed above for specifying grid, function and derivation information can be scaled to an arbitrary dimension $n \geq 2$, it would become extremely cumbersome if we were to input such data manually into a file. Also, apart from very simple cases like the one shown above, we would not have any guarantees as to whether the input is consistent or not. This will immediately pose a significant problem, especially when dealing with a fine-grained structure of the n-dimensional grid contained inside the unit-square $U = [0, 1]^n$.

In order to address these issues, we had to come up with a robust method of generating data automatically in the above format. Also, for the purposes of evaluation, we had to ensure that we can actually generate both consistent and inconsistent input so that the correctness of our implementation can then be thoroughly validated. We will present the input generation mechanisms in very great detail in the next chapter dedicated to Evaluation.

4.2.2 Linear Programming Algorithms

Given that we devised a scalable method for representing all the necessary input required for the linear programming algorithms 3.11 and 3.12, we can now dive into their implementation.

It is first important to notice that the Theorem 3.2.1 allows us to test for whether a pair of functions (f, g) is consistent or not by constructing an LP problem which will only check for the feasibility of a solution, rather than for optimising a linear objective function. However, either algorithms 3.11 or 3.12 can verify the existence of a solution given the provided constraint set while at the same time trying to derive the least and greatest surfaces that are witnesses to consistency.

Therefore, it makes complete sense to use one or the other optimisation problems given by 3.11 and 3.12 for assessing the consistency of a given pair (f, g). Without loss of generality, we will first ask the LP solver to deal with the minimisation problem 3.11. In case consistent input has been detected – and implicitly the minimal witness constructed – we will then proceed to solving the converse maximisation problem. According to Theorem 2.2.1, the latter will be nothing else than solving the minimisation problem again but flipping the sign of the objective function upon multiplication with -1, as the constraint sets are otherwise identical for both problems.

On the other hand, given our reliance on the CVXOPT library which we covered in Section 2.3, we had to ensure that the Algorithm 3.11 (or the corresponding result in higher dimensions) is mapped to the following matrix form 2.6 expected by the LP solver (note that we stripped off the equality constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ which are not relevant in this case):

$$\begin{array}{ll} \text{Minimise} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}. \end{array} \tag{4.1}$$

Firstly, it should be clear than all the $p_1 \times p_2 \times \ldots \times p_n$ heights in the unit-square U will be the decision variables corresponding to the vector $\boldsymbol{x}$ in 4.1 above (not to be confused with $\boldsymbol{h}$, which is the column vector for the inequality constraints that will be discussed further). As we are trying to minimise the sum of all these heights, it immediately follows that all the $p_1 \times p_2 \times \ldots \times p_n$ components of $\boldsymbol{c}$ will have to be equal to 1.

Before moving on to the inequality constraints of the LP problem, we need to make a crucial assumption that will be used throughout the construction

of the coefficient matrix $\boldsymbol{G}$ and the column vector of right hand sides $\boldsymbol{h}$. As pointed out earlier, the vector $\boldsymbol{x}$ of decision variables will include the heights $h_{k_1k_2\ldots k_n}$ at each grid point $G_{k_1k_2\ldots k_n}$ in the unit-square U, where $0 \leq k_i \leq p_i - 1$, $1 \leq i \leq n$. Since the product between the coefficient matrix $\boldsymbol{G}$ and the column vector $\boldsymbol{x}$ will involve all the dot products between each row of the matrix and the vector of decision variables, we need to have a precise ordering of the entries $h_{k_1k_2\ldots k_n}$ within $\boldsymbol{x}$. This is equivalent to imposing a specific ordering on the indices $k_1k_2\ldots k_n$, where, as before, $0 \leq k_i \leq p_i - 1$ and $1 \leq i \leq n$. The most convenient arrangement is to consider the lexicographical ordering for these indices, that is:

$$(k_1, k_2, \ldots, k_n) \leq_l (k'_1, k'_2, \ldots, k'_n)$$

holds if and only if:

$$\exists\, m > 0.\ \forall\, i < m.\ (k_i = k'_i \wedge k_m < k'_m)\,.$$

With this assumption in mind, we are now ready to tackle the inequality constraints from Algorithm 3.11, which need to be written in the matrix form $\boldsymbol{Gx} \leq \boldsymbol{h}$. We should now observe that there are only two types of inequality constraints that can arise from the LP Algorithm 3.11 (written below for the arbitrary case $n \geq 2$):

1. From the function information, the heights corresponding to each grid-point in the unit-square will be constrained by lower and upper bounds of the form:
$$C^{-}_{k_1k_2\ldots k_n} \leq h_{k_1k_2\ldots k_n} \leq C^{+}_{k_1k_2\ldots k_n}, \tag{4.2}$$
where $0 \leq k_i \leq p_i - 1$, $1 \leq i \leq n$.

2. From the derivative information, the differences between adjacent heights at a sub-hyper-rectangle will satisfy inequalities of the form:
$$B^{k-}_{k_1k_2\ldots k_n} \leq h_{k'_1k'_2\ldots k'_n} - h_{k_1k_2\ldots k_n} \leq B^{k+}_{k_1k_2\ldots k_n}, \tag{4.3}$$
where $k_1k_2\ldots k_n$ and $k'_1k'_2\ldots k'_n$ are appropriately chosen indices along the k^{th} axis within the sub-hyper-rectangle with vertices $G_{k_1+b_1,\ldots,k_n+b_n}$, $b_i \in \{0, 1\}$, for $0 \leq k_i \leq p_i - 2$, $1 \leq i \leq n$.

However, it is important to observe that the representation 4.1 enforced by CVXOPT requires that any linear combination of decision variables to be bounded above by a corresponding upper bound from the column vector $\boldsymbol{h}$.

Thus, we will need to flip the lower bounds from the above constraints and write them as follows:

$$-h_{k_1k_2\ldots k_n} \leq -C^-_{k_1k_2\ldots k_n} \qquad h_{k_1k_2\ldots k_n} - h_{k'_1k'_2\ldots k'_n} \leq -B^{k-}_{k_1k_2\ldots k_n}. \tag{4.4}$$

As a result, it suffices to create the coefficient matrix $\boldsymbol{G}$ for the right hand sides of (4.2) and (4.3), because then $\boldsymbol{G'} = -\boldsymbol{G}$ will be the corresponding coefficient matrix for the lower bounds written in the form 4.4. We can actually think of the matrix $\boldsymbol{G}$ as being split into sub-blocks $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$, which represent the coefficient matrices for the upper bounds of (4.2) and the upper bounds of (4.3), respectively. Corresponding to these $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ we will have some column vectors of right hand sides $\boldsymbol{h}_1^+$ and $\boldsymbol{h}_2^+$, respectively, whose entries will be nothing else than the values $C^+_{k_1k_2\ldots k_n}$ and $B^{k+}_{k_1k_2\ldots k_n}$, respectively. Similarly, for the inverted lower bounds given by the inequalities (4.4), we will have coefficient sub-matrices $-\boldsymbol{G}_1$ and $-\boldsymbol{G}_2$ with corresponding upper bounds given by $-C^-_{k_1k_2\ldots k_n}$ and $-B^{k-}_{k_1k_2\ldots k_n}$, that will be gathered in some column vectors $\boldsymbol{h}_1^-$ and $\boldsymbol{h}_2^-$, respectively.

In summary, by putting together all the constraints from (4.2) and (4.3) in the form required by (4.1), we get that:

$$\begin{aligned} \boldsymbol{G}_1\boldsymbol{x} &\leq \boldsymbol{h}_1^+ \\ -\boldsymbol{G}_1\boldsymbol{x} &\leq \boldsymbol{h}_1^- \\ \boldsymbol{G}_2\boldsymbol{x} &\leq \boldsymbol{h}_2^+ \\ -\boldsymbol{G}_2\boldsymbol{x} &\leq \boldsymbol{h}_2^- \end{aligned} \tag{4.5}$$

Note that the construction of the column vectors $\boldsymbol{h}_k^-$ and $\boldsymbol{h}_k^+$, $k \in \{1, 2\}$ is equivalent to finding the the best bounds $C^-_{k_1k_2\ldots k_n}$, $C^+_{k_1k_2\ldots k_n}$, $B^{k-}_{k_1k_2\ldots k_n}$, $B^{k+}_{k_1k_2\ldots k_n}$, where $1 \leq k \leq n$, which one can express in terms of the given $c^-_{k_1k_2\ldots k_n}$, $c^+_{k_1k_2\ldots k_n}$, $b^{k-}_{k_1k_2\ldots k_n}$, $b^{k+}_{k_1k_2\ldots k_n}$ as well as the grid information – the latter being needed only when computing the gradients as a difference of appropriately chosen consecutive heights. We have described this procedure for the two-dimensional case in Chapter 3, Section 3.2.2. This is not more technically or conceptually involved in the n-dimensional setting, as we follow the same procedure described in the particular case: we consider all the sub-hyper-rectangles in the unit-grid and iteratively optimise the bounds for each height and for each particular difference of heights corresponding to all the n partial derivatives.

Thus, it remains to address the construction of the coefficient matrices $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$, respectively. We will also briefly discuss them in relation to the

column vectors of right hand sides $\boldsymbol{h}_1^+$ as well as $\boldsymbol{h}_2^-$ (note that the case for the inverted lower bounds is similar). Let us take each of them in turn:

Construction of $\boldsymbol{G}_1$. According to the upper bound of (4.2), $\boldsymbol{G}_1$ will simply be the *identity matrix* of size $p_1 \times p_2 \times \ldots \times p_n$. This is easy to see because relation (4.2) has $p_1 \times p_2 \times \ldots \times p_n$ independent inequalities, each corresponding to a distinct height defined at grid point $G_{k_1 k_2 \ldots k_n}$. It is also obvious that, since we imposed the lexicographical ordering within the vector $\boldsymbol{x}$ of heights, then the entries of type $C^+_{k_1 k_2 \ldots k_n}$ in the column vector of right hand sides $\boldsymbol{h}_1^+$ will follow the same lexicographical ordering for the subscripts $k_1 k_2 \ldots k_n$.

Construction of $\boldsymbol{G}_2$. Finally, for the construction of $\boldsymbol{G}_2$ we need the following important observation. From the right hand side of (4.3), we see that it is convenient to consider the inequalities for each i^{th} partial derivative in turn. Thus, matrix $\boldsymbol{G}_2$ can striped horizontally in n sub-matrices $\boldsymbol{G}_2^i$, each corresponding to the i^{th} partial derivative, where $1 \leq i \leq n$.

Let us now focus on such a sub-matrix $\boldsymbol{G}_2^d$, for some fixed d, $1 \leq d \leq n$. Note that the upper bounds for the inequalities (4.3) require all the differences between heights at adjacent grid points for *all* the sub-hyper-rectangles within the unit-square $[0,1]^n$. Since we examine the constraints along the d^{th} partial derivative, we can observe that there are exactly:

$$N_d \equiv (p_d - 1) \cdot \prod_{i \neq d} p_i$$

such inequalities, given that those differences are only undefined when we have reached a *border* grid point $G_{k_1 k_2 \ldots k_n}$ for which $k_d = p_d - 1$ (since we cannot step outside the unit-grid to obtain an additional difference of two heights).

To be more specific, for this d^{th} partial derivative, we can rewrite the upper bounds of (4.3) as follows:

$$h_{k_1,\ldots,k_{d-1},k_d+1,k_{d+1},\ldots,k_n} - h_{k_1,\ldots,k_{d-1},k_d,k_{d+1},\ldots,k_n} \leq B^{d+}_{k_1 k_2 \ldots k_n}, \tag{4.6}$$

where $0 \leq k_i \leq p_i - 1$ for $1 \leq i \leq n$, $i \neq d$ and $0 \leq k_d \leq p_d - 2$.

Finally, we can construct the upper coefficient matrix $\boldsymbol{G}_2^+$ since we now have the mathematical expression of all the N_d inequalities given by (4.6). Hence, for each of the N_d rows of the sub-matrix in question, we need to determine

the column indexes, running from 0 to $p_1 \times p_2 \times \ldots \times p_n$, at which the values -1 and 1 will be placed, so that taking the dot product with $\boldsymbol{x}$ will yield all the inequalities (4.6). Recall that $\boldsymbol{x}$ contains all the decision variables whose indices are ordered lexicographically. Thus, in order to determine the appropriate column indices previously mentioned, we simply need to find the sequencing/row number of each height $h_{k_1 k_2 \ldots k_n}$ within the column vector $\boldsymbol{x}$. But this is now trivial, as given any index $k_1 k_2 \ldots k_n$, we can compute its 0-indexed position within the vector $\boldsymbol{x}$ as follows:

$$k_1 \cdot \prod_{i \geq 1} p_i + k_2 \cdot \prod_{i \geq 2} p_i + \ldots + k_{n-1} \cdot p_n + k_n. \tag{4.7}$$

Needless to say, the vector of right hand sides $\boldsymbol{h}_2^+$ will be similarly row striped in d sub-vectors and the position within such a stripe at which $B^{d+}_{k_1 k_2 \ldots k_n}$ will be inserted can be computed using the same (4.7) derived earlier.

To better illustrate this rather convoluted discussion, we will show in full the sparse matrix $\boldsymbol{G}_2$, together with the the column vector $\boldsymbol{h}_2^+$ for a simple example in the two-dimensional case. We will consider, for the sake of presentation, that we have $p_1 = 3$ and $p_2 = 4$ points along the x and y axis, respectively. Thus, there are $3 \times 4 = 12$ decision variable in total and the system $\boldsymbol{G}_2 \boldsymbol{x} \leq \boldsymbol{h}_2^+$ is given by:

$$\left[\begin{array}{cccc|cccc|cccc}
-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\
\hline\hline
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1
\end{array}\right]
\left[\begin{array}{c}
h_{00} \\ h_{01} \\ h_{02} \\ h_{03} \\ \hline
h_{10} \\ h_{11} \\ h_{12} \\ h_{13} \\ \hline
h_{20} \\ h_{21} \\ h_{22} \\ h_{23}
\end{array}\right]
\leq
\left[\begin{array}{c}
B^+_{00} \\ B^+_{01} \\ B^+_{02} \\ B^+_{03} \\ \hline
B^+_{10} \\ B^+_{11} \\ B^+_{12} \\ B^+_{13} \\ \hline
B^+_{20} \\ B^+_{21} \\ B^+_{22} \\ B^+_{23}
\end{array}\right]$$

Putting together the algorithm. After the thorough analysis developed so far, we now possess all the necessary information to write a procedure that, given n-dimensional function and derivative constraints for each of the sub-hyper-rectangles in the unit square U, determines whether consistency holds or not.

The pseudocode Algorithm 4.2.1 defines a procedure SolveLPAlgorithm which takes as input interval-valued function information f and rectangular derivative-information g within an n-dimensional *grid*. The program builds the constraints in the matrix form 4.5 and then solves the minimisation LP problem to decide consistency. If found, the procedure also solves the converse optimisation problem by inverting the objective function. In such a case, the output will be a pair of n-dimensional arrays representing the minimal and maximal bounding surfaces which are witnesses to consistency. Otherwise, an empty pair is returned signalling that the input given by the pair (f, g) is inconsistent.

4.3 Front-end

Lastly, we will cover some details regarding the graphical user interface that we implemented alongside the LP algorithms described above. Given that for an n-dimensional domain the resulting piecewise linear surfaces are in dimension $(n + 1)$, we could only provide 3D visualisation of the geometric objects for a consistent pair defined in the two-dimensional unit square.

In order to achieve this, we leveraged the `mplot3d` toolkit from the `matplotlib`[1] package, which provides rendering of tri-surface plots via the following convenient API: [2]

```
Axes3D.plot_tri_surf(X, Y, Z, color, cmap, norm, vmin, vmax,
                     shade, triangles),
```

where `X`, `Y` and `Z` are the data values represented as flat, one-dimensional arrays. All the next parameters offer enough flexibility for customising the look and feel of the resulting plot. We will dedicate a special attention to the last argument, i.e. `triangles`, which can be used to specify a particular arrangement of the triangular patches that are finally rendered in the 3D scene.

[1] `http://matplotlib.org/mpl_toolkits/mplot3d/`

[2] `http://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html#mpl_toolkits.mplot3d.Axes3D.plot_trisurf`

Algorithm 4.2.1 The Final Linear Programming Algorithm

procedure SolveLPAlgorithm(f, g, $grid$)
    $result \leftarrow (\emptyset, \emptyset)$

    ▷ *Coefficient matrices for function and derivative constraints.*
    $\boldsymbol{G}_1 \leftarrow$ BuildFunctionCoefMatrix(f, $grid$)
    $\boldsymbol{G}_2 \leftarrow$ BuildDerivativeCoefMatrix(g, $grid$)

    ▷ *Column vectors of optimised lower and upper bound constraints. For function information, this works by taking the* max *of all the lower bounds as well as the* min *of all the upper bounds for each height that is defined in adjacent sub-hyper-rectangles within the grid. A similar reasoning applies for optimising the overlapping differences of heights using the derivative constraints.*
    $(C^-_{\text{vec}}, C^+_{\text{vec}}) \leftarrow$ BuildFunctionConstraints(f, $grid$)
    $(B^-_{\text{vec}}, B^+_{\text{vec}}) \leftarrow$ BuildDerivativeConstraints(g, $grid$)

    ▷ *Right hand sides corresponding to 4.5.*
    $(\boldsymbol{h}^+_1, \boldsymbol{h}^-_1) \leftarrow (C^+_{\text{vec}}, -C^-_{\text{vec}})$
    $(\boldsymbol{h}^+_2, \boldsymbol{h}^-_2) \leftarrow (B^+_{\text{vec}}, -B^-_{\text{vec}})$

    ▷ *Solve the LP minimisation problem by combining constraints 4.5 into single matrix form* $\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}$.
    $\boldsymbol{c} \leftarrow$ Vector(1, $grid_{\text{noPoints}}$)
    $\boldsymbol{G} \leftarrow$ JoinMatrices($\boldsymbol{G}_1$, $-\boldsymbol{G}_1$, $\boldsymbol{G}_2$, $-\boldsymbol{G}_2$)
    $\boldsymbol{h} \leftarrow$ JoinVectors($\boldsymbol{h}^+_1$, $\boldsymbol{h}^-_1$, $\boldsymbol{h}^+_2$, $\boldsymbol{h}^-_2$)
    $\boldsymbol{x}_{\min} \leftarrow$ SolveLP($\boldsymbol{c}$, $\boldsymbol{G}$, $\boldsymbol{h}$)

    ▷ *If consistency is found, then solve the LP maximisation problem.*
    if $\boldsymbol{x}_{\min} \neq \emptyset$ **then**
        $\boldsymbol{x}_{\max} \leftarrow$ SolveLP($-\boldsymbol{c}$, $\boldsymbol{G}$, $\boldsymbol{h}$)
        $result \leftarrow (\boldsymbol{x}_{\min}, \boldsymbol{x}_{\max})$
    end if

    return $result$
end procedure

4.3.1 Triangulation of Sub-rectangles

Recall that in Section 3.2.2 from Chapter 3, when discussing the construction of the piecewise linear surfaces, we chose to divide each 2D sub-rectangle of the grid along the diagonal joining the lower-right and upper-left corners of any such sub-rectangle. This allows us to linearly interpolate the z-values at the vertices of all of these triangles in a consistent manner.

However, without the last parameter, the `Axes3D.plot_tri_surf` function will employ Delaunay triangulation [18] which performs the partitioning of sub-rectangles in such a way so that it maximises the minimal angle of the resulting 3D piecewise linear triangles. As a consequence, this technique allows for more aesthetically pleasing surfaces because it avoids skinny or close to degenerate triangles. Despite the fact that this triangulation also yields a consistent piecewise linear witness, we nonetheless wanted to follow our own construction that we previously described. To do this, one can override the default Delaunay triangulation by providing a list of vertex-labels upon which the triangulation of any rectangular quadrilateral will be uniquely determined. We illustrate how this labelling works using a simple example. Consider a 3×4 grid within the square $[0, 1] \times [0, 1]$ as follows:

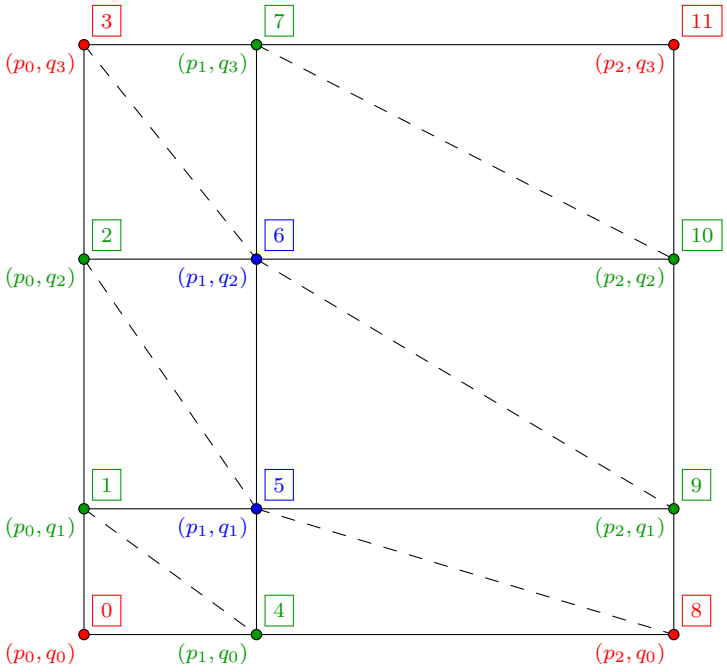


Figure 4.1: Labelling for a 3×4 grid.

We will now assume that the z-values for the heights h_{ij} at each grid-point (p_i, q_j), $i = \overline{0,2}$, $j = \overline{0,3}$, have been arranged as a one-dimensional array such that the collection of indices ij are ordered lexicographically. If this is the case, then each two-dimensional grid point (p_i, q_j) will be assigned a label corresponding to the index at which the height h_{ij} is located in the flat array of values. In the above Figure 4.1, the labels are shown in the box next to each grid-point.

Finally, the triangulation expected by `Axes3D.plot_tri_surf` will simply require a two-dimensional list of all the triangle labels defined using the above procedure. In this example, the `triangles` parameter would therefore be set to:

```
[[0, 4, 1], [4, 5, 1], [1, 5, 2], [5, 6, 2], [2, 6, 3],
 [6, 7, 3], [4, 8, 5], [8, 9, 5], [5, 9, 6], [9, 10, 6],
 [6, 10, 7], [10, 11, 7]].
```

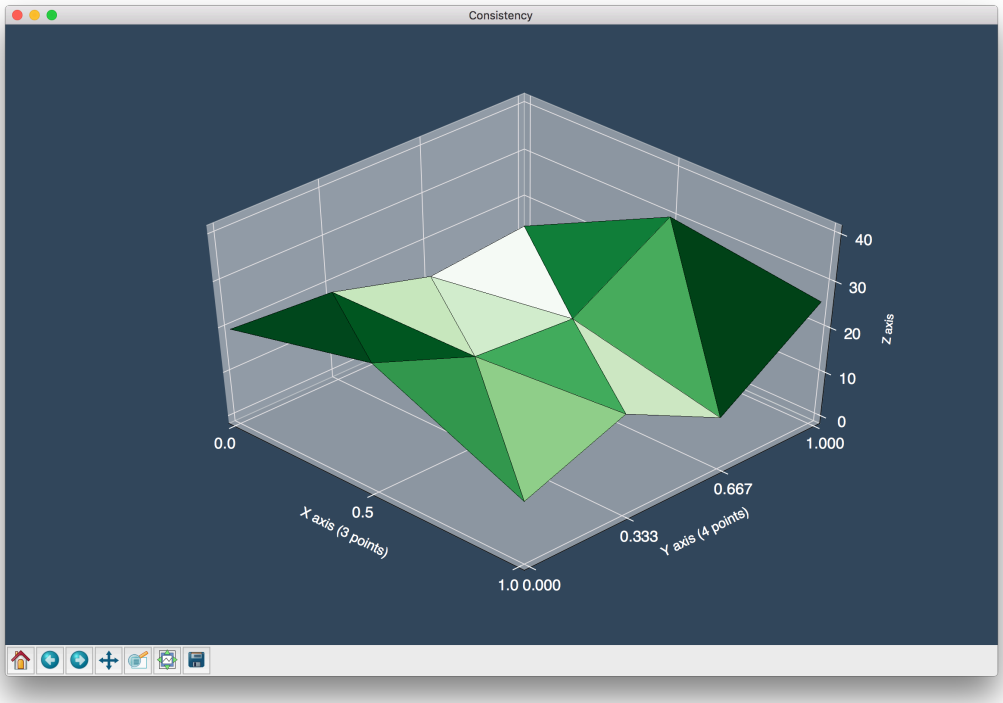


Figure 4.2: Triangulation for a consistent piecewise linear witness

Figure 4.2 shows an example of this triangulation for a piecewise linear surface within a similar 3×4 grid. Note that, for presentation purposes, we chose equally spaced points along the $[0, 1]$ segments. In addition, the 3D scene in the running application can be zoomed, panned or even rotated so that one can analyse the piecewise linear surfaces at a greater level of detail.

4.3.2 Minimal and Maximal Surfaces

We can now easily integrate the visualisation of the least and greatest witnesses to consistency in our framework. Assuming consistent input has been supplied, we retrieve both the minimal and maximal heights via Algorithm 4.2.1 and we then make two separate calls to `Axes3D.plot_tri_surf` by using the same `x` and `y` parameters, but different `z` arguments.

Figure 4.3 shows an example of the least and greatest bounding surfaces that are witnesses to consistency within a 3×4 grid as before:

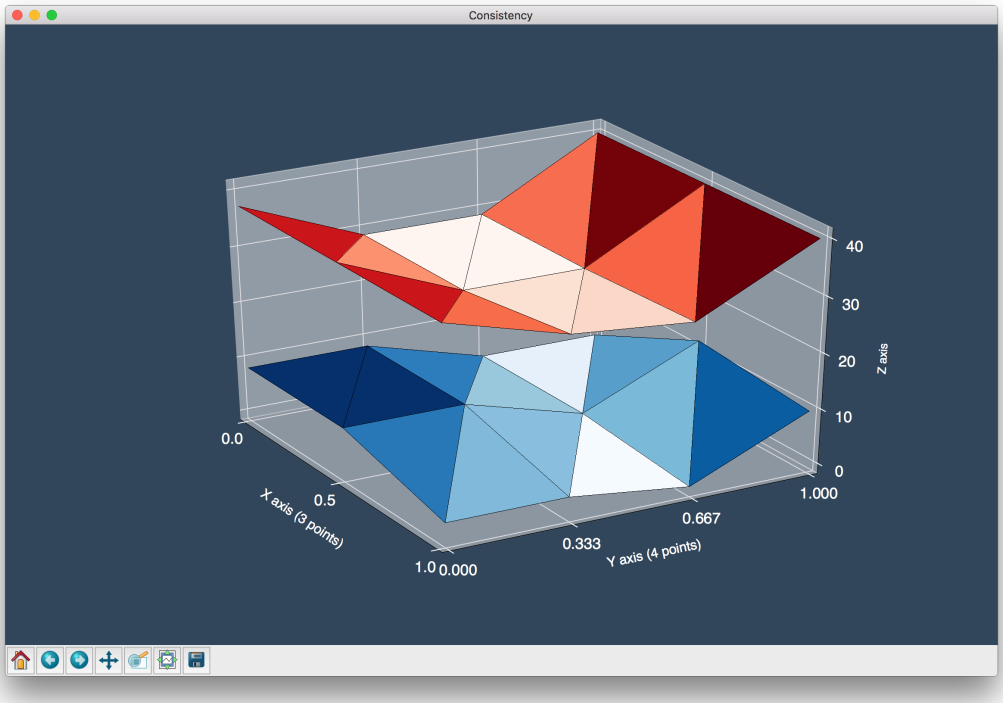


Figure 4.3: Example of minimal and maximal piecewise linear surfaces

Chapter 5

Evaluation

In this chapter we will examine the techniques used to ensure that the implementation of the linear programming algorithm 4.2.1 that decides consistency is correct. We begin by focusing on the methodology of generating consistent/inconsistent input for both function and derivative information, which will lead to an effective method of performing boundary testing [19]. Also, in the particular case when the domain is two-dimensional, we will use the visualisation of the 3D piecewise linear surfaces to further validate our implementation subject to varying some parameters. We then conclude with the challenges and limitations associated with this project.

5.1 Input Generation

Recall that at the end of Section 4.2.1 from the previous chapter we established that it is rather impractical to manually input data in the form of closed and compact intervals for function and derivation information, respectively. Even in the case of a two-dimensional domain, the task of writing values into a file is certainly cumbersome, let alone in higher dimensional spaces.

The problem becomes even more serious when we would like to distinguish between consistent and inconsistent input for the purposes of verification. This is because we want to guarantee that the implementation of the LP algorithm is correctly identifying consistent input as being consistent and, conversely, that inconsistent input is classified as being inconsistent. Thus, the absence of any false alarms when testing on a large number of randomly-generated inputs will provide a strong justification towards the correctness of our implementation.

As a result, the task of generating input is of paramount importance to a successful evaluation of this project. We start by looking at how we can automatically produce consistent input, based either on randomly chosen values or randomly-generated polynomials. We will then see how we can create inconsistent input by making a minimal and non-trivial modification to any consistent pair of function and derivative approximation.

Throughout this analysis, we will make extensive use of the notations developed in Section 4.2.1 from the previous chapter. If needed, the reader is strongly encouraged to refer back to that section, as the following discussion will be rather notationally-heavy.

5.1.1 Generation of consistent input

The problem of generating consistent function and derivative information reduces to reverse engineering the constraints of the linear programming algorithms 3.11 and 3.12 from Chapter 3. For this reason, we start by randomly generating heights $h_{k_1k_2\ldots k_n}$ at each of the of the grid points $G_{k_1k_2\ldots k_n}$, where $0 \leq k_i \leq p_i - 1$ and $1 \leq i \leq n$. We implemented two distinct ways for generating such arbitrary heights:

- Either by drawing $p_1 \times p_2 \times \ldots \times p_n$ values from a uniform distribution over a fixed interval;
- Or, more interestingly, by constructing a random polynomial in n variables from which we calculate the values at each of the $p_1 \times p_2 \times \ldots \times p_n$ grid points $G_{k_1k_2\ldots k_n}$.

Consistent Function Information. After having the heights in place for all the $p_1 \times p_2 \times \ldots \times p_n$ grid points, we firstly need to ensure that we construct large enough intervals such that all the heights within a sub-hyper-rectangle satisfy:

$$c^-_{k_1k_2\ldots k_n} \leq h_{k_1+b_1,\ldots,k_n+b_n} \leq c^+_{k_1k_2\ldots k_n},$$

where $0 \leq k_i \leq k_i - 2$ and $b_i \in \{0, 1\}$, for $1 \leq i \leq n$. It is now easy to see that we can guarantee consistent function information by choosing the tightest closed and compact intervals

$$\left[c^-_{k_1k_2\ldots k_n}, c^+_{k_1k_2\ldots k_n}\right],$$

where $0 \leq k_1 \leq p_i - 2$ and $1 \leq i \leq n$. This is trivial because for each of the $(p_1 - 1) \times (p_2 - 1) \times \ldots \times (p_n - 1)$ sub-hyper-rectangles inside the unit-square U, the corresponding heights will be contained within the most rigid $(n + 1)$-dimensional box.

However, for added flexibility, we can introduce a parameter ε to allow for more loose bounding boxes at each of the sub-hyper-rectangles within the grid, so we will define intervals of the form:

$$\left[c^{-}_{k_1 k_2 \ldots k_n} - \varepsilon, c^{+}_{k_1 k_2 \ldots k_n} + \varepsilon\right],$$

for all $0 \leq k_i \leq p_i - 2$, $1 \leq i \leq n$.

A simple implementation of the ideas discussed thus far is given by the pseudocode procedure 5.1.1. This function returns an n-dimensional array of intervals representing consistent function information within each sub-hyper-rectangle of U, given random heights h_{rand}, grid information $grid$ and tolerance parameter ε.

Algorithm 5.1.1 Generating Consistent Function Information

```
1: procedure GenerateFunctionInfo(h_rand, grid, ε)
2:     f ← ∅
3:     for all index_curr ∈ GetGridIndices(grid) do
4:         interval ← (0, 0)
5:         if index_curr ∉ BorderIndexSet(grid) then
6:             (h_min, h_max) ← (∞, −∞)
7:             for all index_next ∈ NextGridIndices(index_curr) do
8:                 h ← h_rand[index_next]
9:                 (h_min, h_max) ← (Min(h_min, h), Max(h_max, h))
10:            end for
11:            interval ← (h_min − ε, h_max + ε)
12:        end if
13:        f[index_curr] ← interval
14:    end for
15:    return f
16: end procedure
```

Consistent Derivative Information. Finally, we also need to deal with the construction of the n-dimensional array that will provide the derivative information. Given that we derived the minimal bounding boxes that guarantee consistent function information, we could very easily compute the minimal and maximal slopes along n intersecting hyper-rectangular faces which are pairwise orthogonal. However, this approach may result in some loss of information since we specifically started from random heights at the grid points and most of these heights will be strictly included in the minimal bounding boxes computed previously.

As a result, we will take into account all the randomly generated heights in our strategy for deriving the tightest closed and compact intervals along each of the n axis and for each of the $(p_1 - 1) \times (p_2 - 1) \times \ldots \times (p_n - 1)$ sub-hyper-rectangles in the unit-grid. For convenience, we will explain this method for the case when $n = 2$, as it is similar for higher dimensions.

Recall Figure 3.2 in which we had a single sub-rectangle defined by the lower-left corner at $(p_i, q_j) \in \mathbb{R}^2$ and 4 heights h_{st}, $s \in \{i, i+1\}$, $t \in \{t, t+1\}$. As demonstrated by 3.7 in Section 3.2.2, the gradients along the x axis are given by the following slopes:

$$\frac{h_{i+1,j} - h_{ij}}{p_{i+1} - p_i} \qquad \frac{h_{i+1,j+1} - h_{i,j+1}}{p_{i+1} - p_i}. \tag{5.1}$$

Algorithm 5.1.2 Generating Consistent Derivative Information

procedure GENERATEDERIVATIVEINFO(h_{rand}, $grid$, ε)
  $d \leftarrow \emptyset$
  for all $index_{\text{curr}} \in$ GETGRIDINDICES($grid$) **do**
    ▷ n-dimensional tuple of pairs
    $intervals \leftarrow ((0,0), \ldots (0,0))$
    if $index_{\text{curr}} \notin$ BORDERINDEXSET($grid$) **then**
      for $k \leftarrow 1, n$ **do**
        ▷ Calculate the edge-length between $index_{\text{curr}}$ and the next index along the k^{th} axis for the current sub-hyper-rectangle in the grid
        $edge_{\text{len}} \leftarrow$ GETEDGELENGTH($grid$, $index_{\text{curr}}$, k)
        $(s_{\min}, s_{\max}) \leftarrow (\infty, -\infty)$
        $I_{\text{edges}} \leftarrow$ INDICESFORPARALLELEDGES($grid$, $index_{\text{curr}}$, k)
        for all $(index_{\text{left}}, index_{\text{right}}) \in I_{\text{edges}}$ **do**
          $(h_{\text{left}}, h_{\text{right}}) \leftarrow (h_{\text{rand}}[index_{\text{left}}], h_{\text{rand}}[index_{\text{right}}])$
          $s \leftarrow (h_{\text{right}} - h_{\text{left}})/edge_{\text{len}}$
          $(s_{\min}, s_{\max}) \leftarrow (\text{MIN}(s_{\min}, s), \text{MAX}(s_{\max}, s))$
        end for
        $intervals[i-1] \leftarrow (s_{\min} - \varepsilon, s_{\max} + \varepsilon)$
      end for
    end if
    $d[index_{\text{curr}}] \leftarrow intervals$
  end for
  return d
end procedure

Notice that both ratios have in common the distance given by $p_{i+1} - p_i$, which represents the edge-length of the sub-rectangle in the x direction. Also, it is obvious that in this simple two-dimensional case we would just construct a consistent interval from the values of these 2 ratios.

With these details in mind, we can explain the most challenging parts in the pseudocode procedure 5.1.2. We use the same input parameters as before and we output an n-dimensional array of tuples, with each such tuple consisting of n pairs. Now, on line 9 we effectively determine the common edge-length for the current sub-hyper-rectangle in the direction of the k^{th} derivative. Secondly, on line 11 we collect all 2^{n-1} pairs of grid indices for which the corresponding heights will need to be subtracted, in the style shown by (5.1) for the 2D case. In the particular instance that we just analyzed, the IndicesForParallelEdges procedure would return the set with 2 elements:

$$\Big\{\Big((i,j),(i+1,j)\Big),\Big((i,j+1),(i+1,j+1)\Big)\Big\}$$

since $index_{\text{curr}} = (i,j)$ and $k = 1$ (i.e. the partial derivative along the x axis). Lastly, one can specify the tolerance parameter ε to accommodate wider intervals for each partial derivative, in the same manner as we did when extending the minimal bounding boxes for function information.

5.1.2 Generation of inconsistent input

Having seen a rather involved procedure for constructing consistent input, one may think that generating inconsistent function or derivative information would be much easier. This is indeed the case, and we have two ways for tackling this problem:

- We can either break the function information by making two adjacent bounding boxes to have empty intersection – thus, no continuous witness will exist in such a case;
- Or, we can make a minimal change to break the derivative information as follows: we consider to have generated the most constrained intervals for both function and derivative information, as shown in Algorithms 5.1.1 and 5.1.2 when $\varepsilon = 0$ is supplied to both procedures. This means that we have the most rigid $(n+1)$-dimensional hyper-rectangles and also the tightest intervals across all partial derivatives. Now we can pick a random sub-hyper-rectangle in the unit-square U at grid index

$G_{r_1 r_2 \ldots r_n}$, as well as a random axis r_a, for which we have the following interval:

$$\left[b^{r_a^-}_{r_1 r_2 \ldots r_n}, b^{r_a^+}_{r_1 r_2 \ldots r_n}\right] \tag{5.2}$$

where $0 \leq r_i \leq p_i - 2$, $0 \leq i \leq n$ and $1 \leq r \leq n$. Now, since the corresponding minimal bounding box is given in terms of the closed and compact interval:

$$\left[c^-_{r_1 r_2 \ldots r_n}, c^+_{r_1 r_2 \ldots r_n}\right],$$

it means that the greatest consistent interval for the r_a^{th} partial derivative is $S^{r_a} \equiv \Big[-s^{r_a}_{\max}, s^{r_a}_{\max}\Big]$, where

$$s^{r_a}_{\max} \equiv \frac{c^+_{r_1 r_2 \ldots r_n} - c^-_{r_1 r_2 \ldots r_n}}{G^{r_a}_{r_1 \ldots (r_a+1) \ldots r_n} - G^{r_a}_{r_1 \ldots r_a \ldots r_n}}$$

represents the largest achievable slope within the corresponding sub-hyper-rectangle along this randomly chosen derivative. Notice that we use $G^k_{k_1 k_2 \ldots k_n}$ to denote the k^{th} coordinate of the grid point $G_{k_1 k_2 \ldots k_n}$, where $0 \leq k_i \leq p_i - 1$, $1 \leq i \leq n$, $1 \leq k \leq n$. As a matter of fact, the denominator of $s^{r_a}_{\max}$ would be the output of the virtual procedure GetEdgeLength within GenerateDerivativeInfo, whose pseudocode was given as part of Algorithm 5.1.2.

At last, coming back to the single modification that is needed in order to obtain inconsistent derivative information, we need to see that the interval from (5.2) will always be included within S^{r_a}, i.e.

$$\left[b^{r_a^-}_{r_1 r_2 \ldots r_n}, b^{r_a^+}_{r_1 r_2 \ldots r_n}\right] \subseteq \Big[-s^{r_a}_{\max}, s^{r_a}_{\max}\Big].$$

Hence, by choosing any interval that is strictly outside of S^{r_a} it will be sufficient to guarantee inconsistent input. This is because the slope along the r_a^{th} partial derivative within the minimal $(n+1)$-dimensional bounding box under consideration will not have the necessary freedom to oscillate beyond the limits of S^{r_a}.

For the purposes of verifying our implementation described in Algorithm 4.2.1, we decided to choose for second alternative when generating inconsistent data. This is easily motivated because it enables us to perform effective boundary testing [19], as we are able to create inconsistent input by making the minimal non-trivial change to a consistent pair of function and derivative information, respectively, upon altering the latter.

5.2 Results

Having developed robust methods for automatically generating both consistent and inconsistent input, it was now possible to assess the correctness of our implementation of the Algorithm 4.2.1.

We checked the implementation against **hundreds** of randomly generated input files for dimensions $n \in \{2, 3, 4, 5, 6\}$ and considering at most 12 points along each axis. Furthermore, we allowed the parameter $\varepsilon \geq 0$ to disturb both the function and derivative information whenever generating consistent input – which was either coming from random polynomials or from arbitrary values drawn from the interval $[10, 30]$. We chose $\varepsilon \in \{0.0, 0.1, 0.5, 1.0, 10.0, 50.0\}$ in order to cover a wider spectrum of test cases, although it is only when $\varepsilon \to 0$ that the generated test cases are more relevant.

The results are indeed very encouraging. After a couple of weeks spent to address various issues in the implementation outlined in Algorithm 4.2.1, we managed to successfully classify each type of input when running a substantial suite of tests for all dimensions considered under analysis (≈ 1000 tests in total). As such, in the absence of any counterexamples, we have the confidence that we delivered a solid implementation of the linear programming algorithm that decides consistency in the rectangular case.

We do not, however, exclude the possibility of any bugs being missed throughout this process: despite having generated as many various tests as possible, there might still be rather unusual situations which may very well uncover bugs in our framework, e.g. precision errors due to floating-point arithmetic, bugs in the CVXOPT solver etc.

5.2.1 Examples of 3D Piecewise Linear Surfaces

As a final method of evaluation, let us now visually inspect and comment on some of the piecewise linear maps that we have briefly seen as part of the Front-End Section in the previous chapter. We will start by looking at the simplest example in the two-dimensional domain, that is, when the grid is given by the corners of the unit-square $U = [0, 1]^2$. We will then gradually increase the granularity of the grid, together with varying the ε parameter. In all the plots that follow, the randomly-generated surfaces from which consistent data has been constructed are coloured with green. Blue and red are reserved for the minimal and maximal consistent surfaces, respectively.

Figures 5.1 and 5.2 show two examples of consistent input when the grid coincides with the unit-square. The randomly generated surface in the middle is in fact the same in both cases, but the constraints for function and derivative approximation are altered by means of $\epsilon \in \{0, 1\}$.

Note that when $\varepsilon = 0$, the least and greatest piecewise linear witnesses will just touch the original surface at its global minimum and maximum, respectively. This is indeed expected as these extreme values determine the most rigid rectangular parallelepiped in which any witness will lie. In addition, we should also observe that the minimal and maximal surfaces are slanted towards the interior of this tightest bounding box, because the intervals for derivative constraints did not contain 0 for sure – as otherwise these witnesses would have been parallel to the xy-plane.

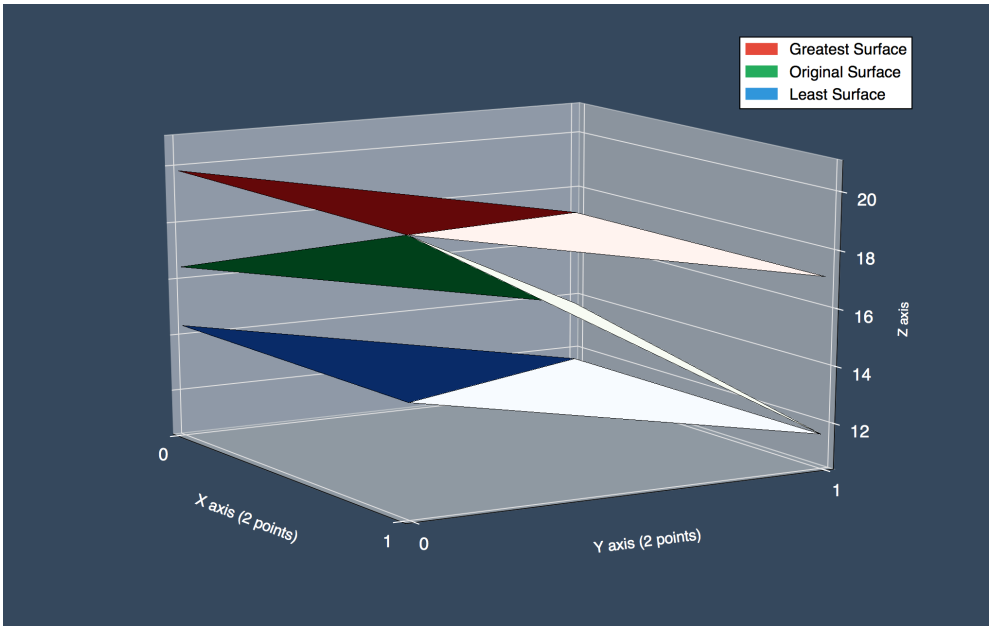


Figure 5.1: 2×2 grid and $\varepsilon = 0$.

Now, when $\varepsilon = 1$ in Figure 5.2, we get to see that the minimal and maximal surfaces are separated from the initial one by exactly this value of ε at the extreme points of the original witness. Moreover, the global bounding surfaces are flatter this time compared to the ones in Figure 5.1 where $\varepsilon = 0$. This is because an $\varepsilon > 0$ will result in wider intervals for the derivative constraints, thus allowing the gradients to have more freedom.

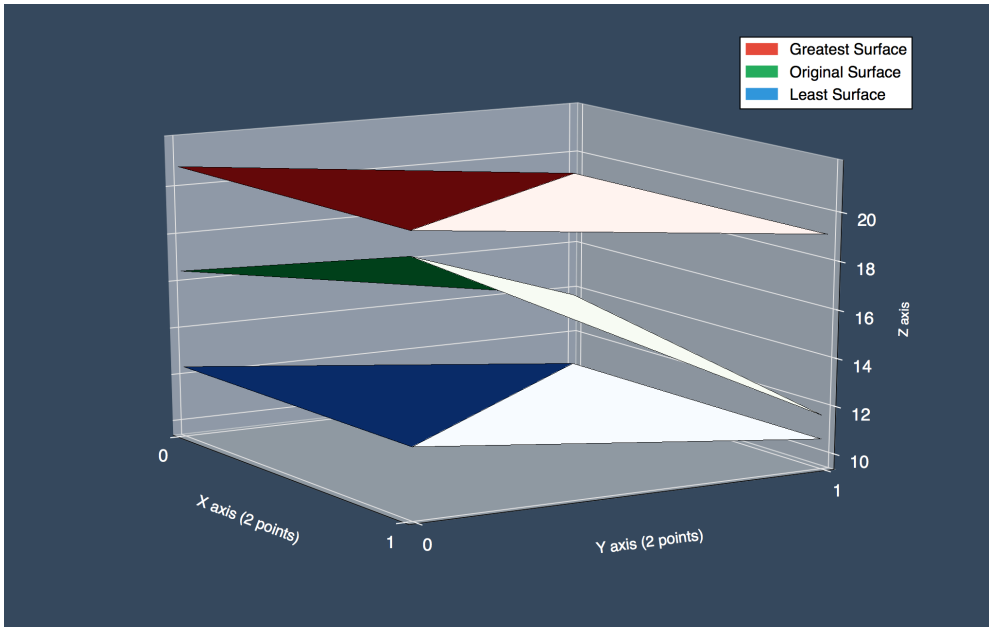


Figure 5.2: 2×2 grid and $\varepsilon = 1$

Next, Figures 5.3, 5.4 and 5.5 show the case of a 3×3 grid with equally spaced points along each axis. In all of these scenarios we end up with much richer surfaces given the more granular partitioning of the unit-square.

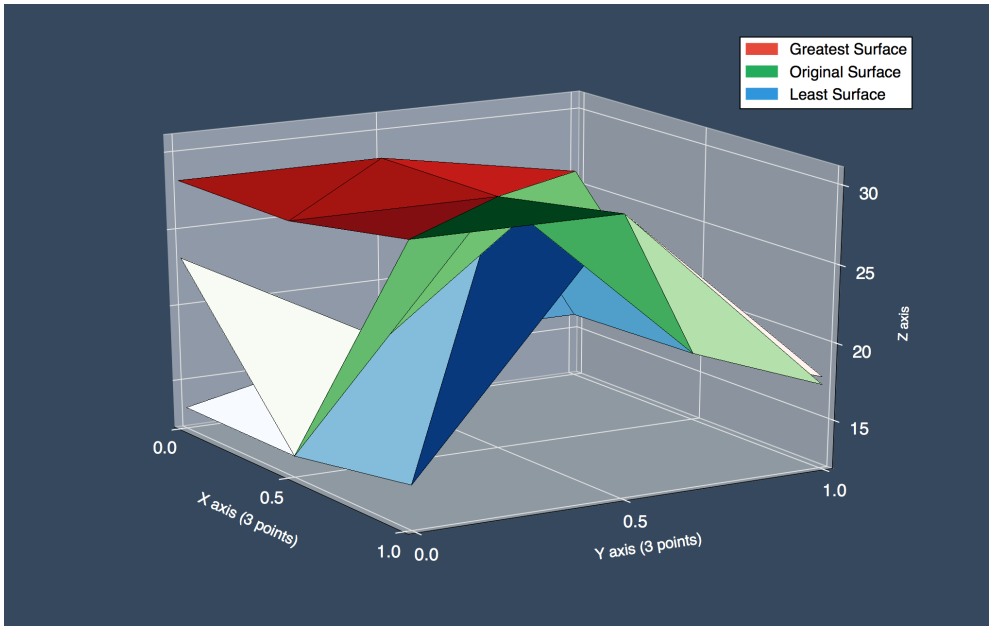


Figure 5.3: 3×3 grid and $\varepsilon = 0$

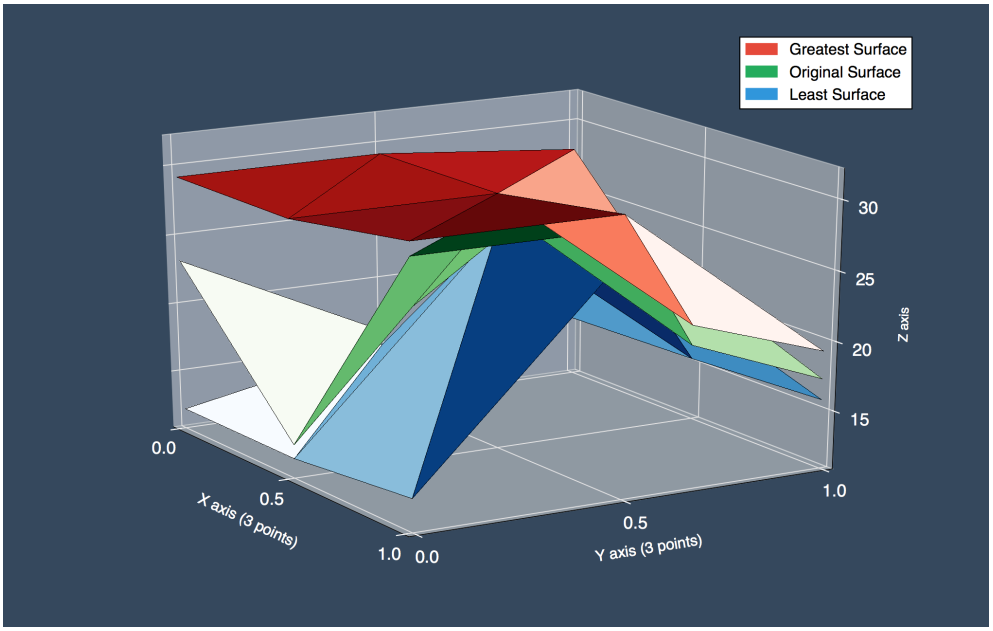


Figure 5.4: 3×3 grid and $\varepsilon = 1$

As in the case of a 2×2 grid, the witnesses are more apart from each other as the ε increases. Similarly, the maximal and minimal surfaces are flatter, due to more permissive constraints on the derivative information.

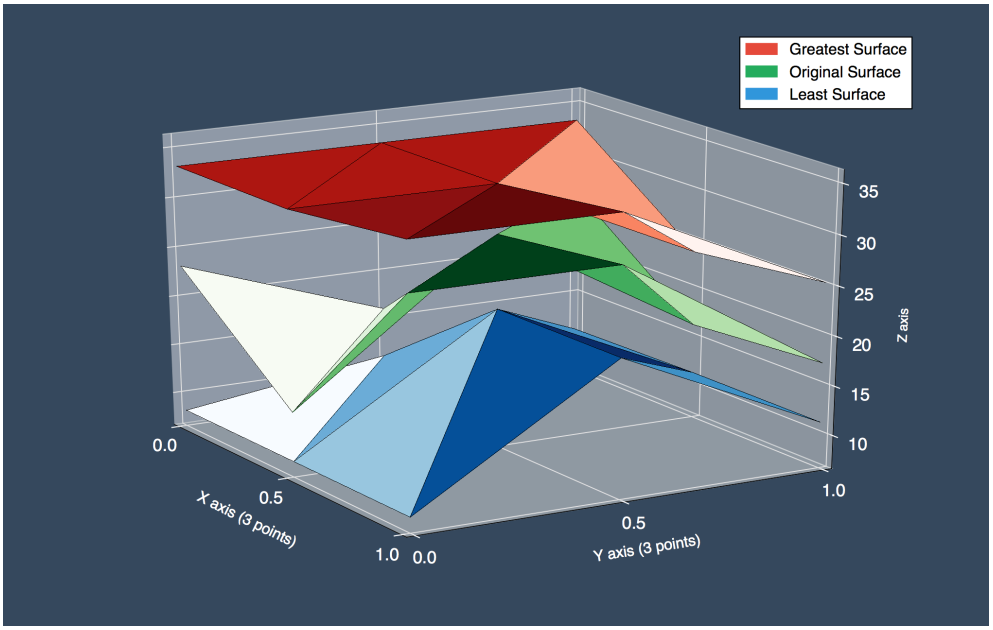


Figure 5.5: 3×3 grid and $\varepsilon = 5$

Apart from randomly generating values within an interval, we also have the ability to generate consistent input from random polynomials. In the two-dimensional case, these polynomials have the following general form:

$$P(x_1, x_2) \equiv \sum_{i=1}^{10} \alpha_i x_1^{e_i^1} x_2^{e_i^2},$$

where the coefficients $\alpha_i \in [-40, 40]$ and exponents $e_i^1, e_i^2 \in [0, 10]$ are randomly chosen, for all $1 \leq i \leq 10$. We opted for a rather small number of terms when generating polynomials, because otherwise evaluating P for each point in the unit grid would require a great deal of unnecessary arithmetic operations.

As we can see in Figures 5.6, 5.7 and 5.8, the resulting surfaces feature a much smoother geometry than before. Note that $\varepsilon = 0$ for the first two plots. Since there is no additional offset introduced into the constraints, the original green surface is now "sandwiched" between the least and greatest consistent witnesses, for all the sub-rectangles in the unit square $[0, 1]^2$. Finally, in Figure 5.8, a larger offset $\varepsilon = 30$ produces separate minimal and maximal surfaces which is what we anticipated.

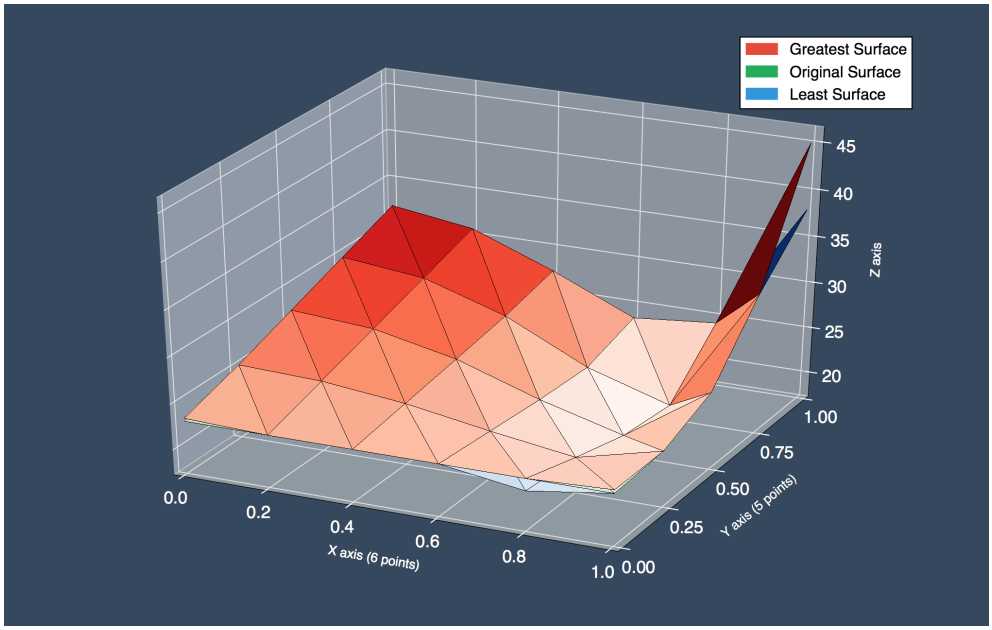


Figure 5.6: 6×5 grid and $\varepsilon = 0$

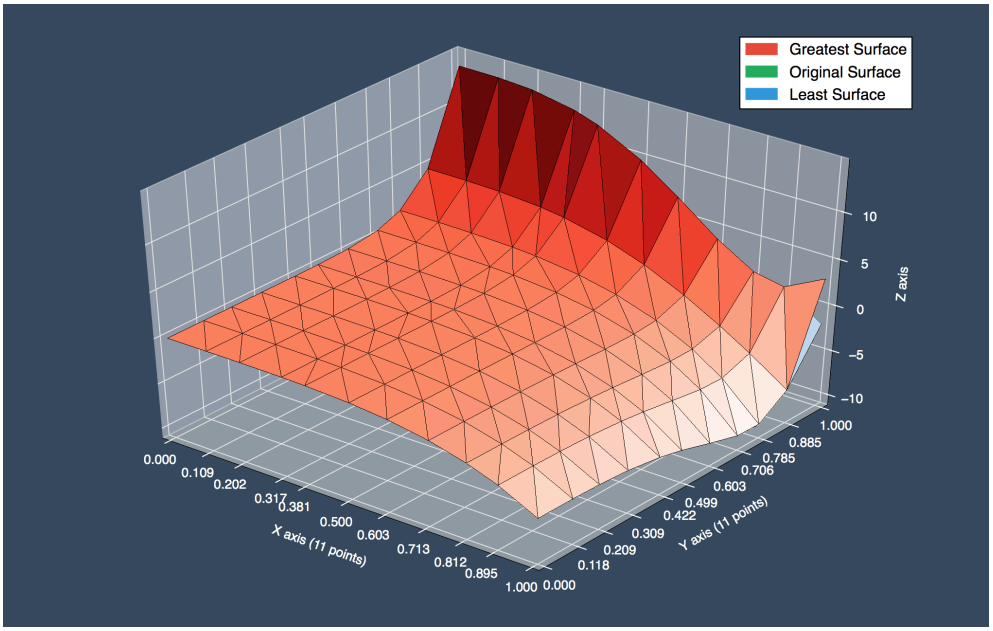


Figure 5.7: 11×11 grid and $\varepsilon = 0$

In addition, we also implemented random generation of grid points, as they do not necessarily need to be equally spaced along the axis of the grid. This is depicted in Figures 5.7 and 5.8.

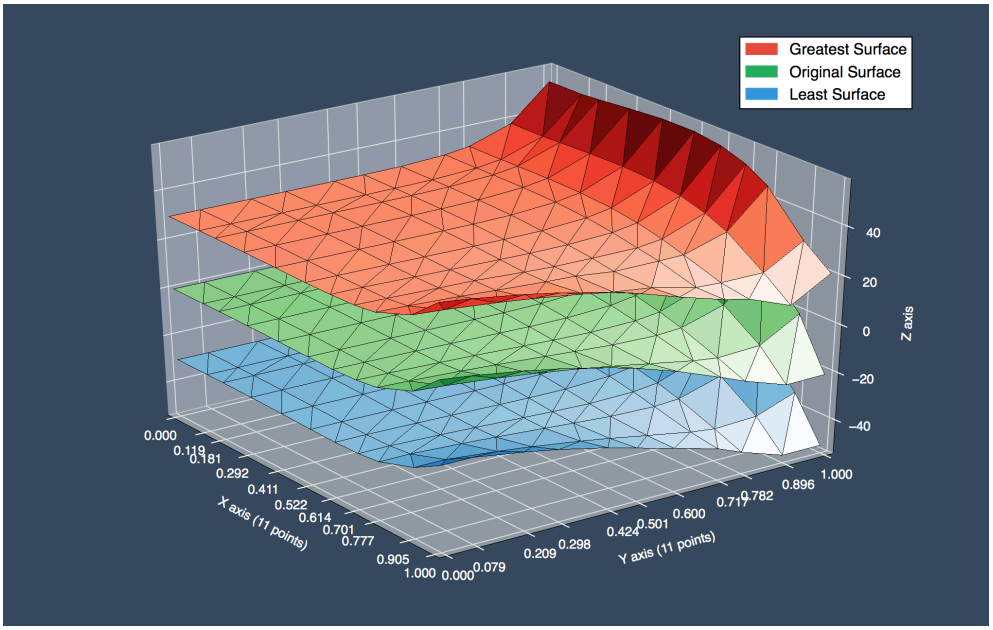


Figure 5.8: 11×11 grid and $\varepsilon = 30$

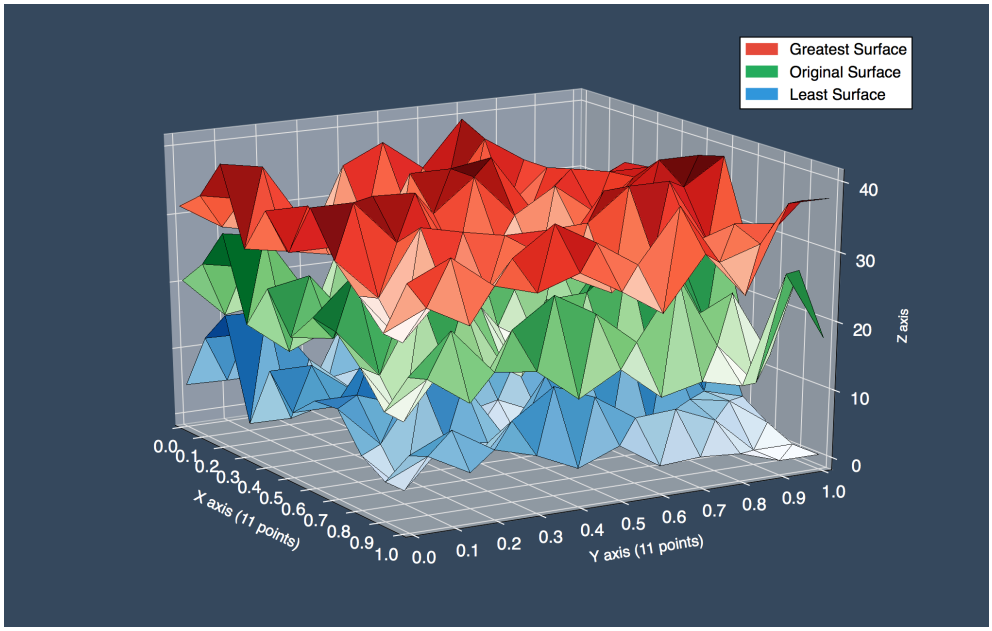


Figure 5.9: 11×11 grid and $\varepsilon = 10$

For the remaining plots, only arbitrary values are used to create the input. Figures 5.9, 5.10 show an 11×11 grid in which $\varepsilon \in \{10, 50\}$. As expected, the surfaces are much closer together for the smaller value of the ε.

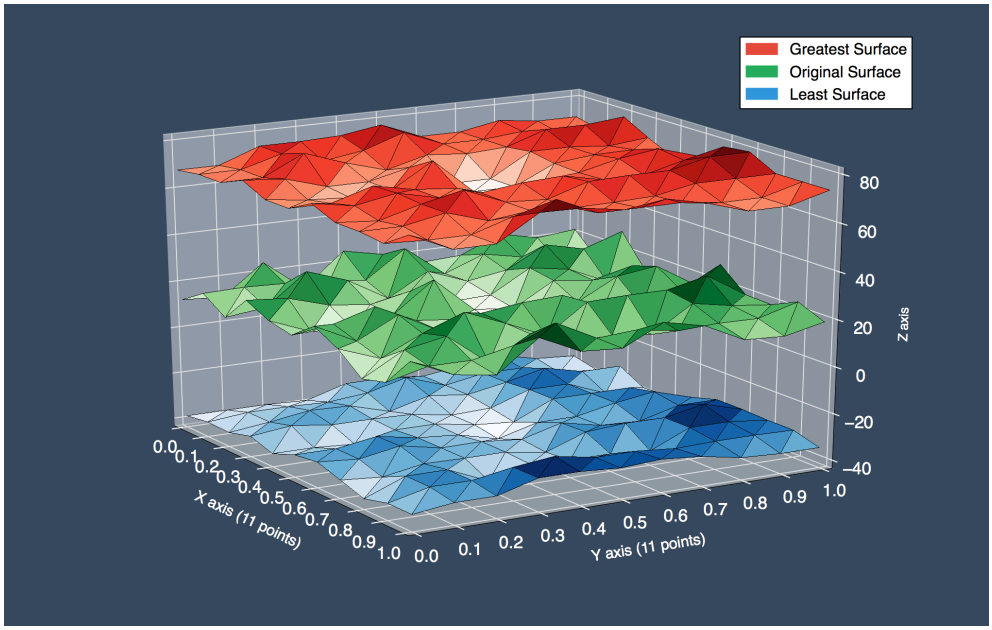


Figure 5.10: 11×11 grid and $\varepsilon = 50$

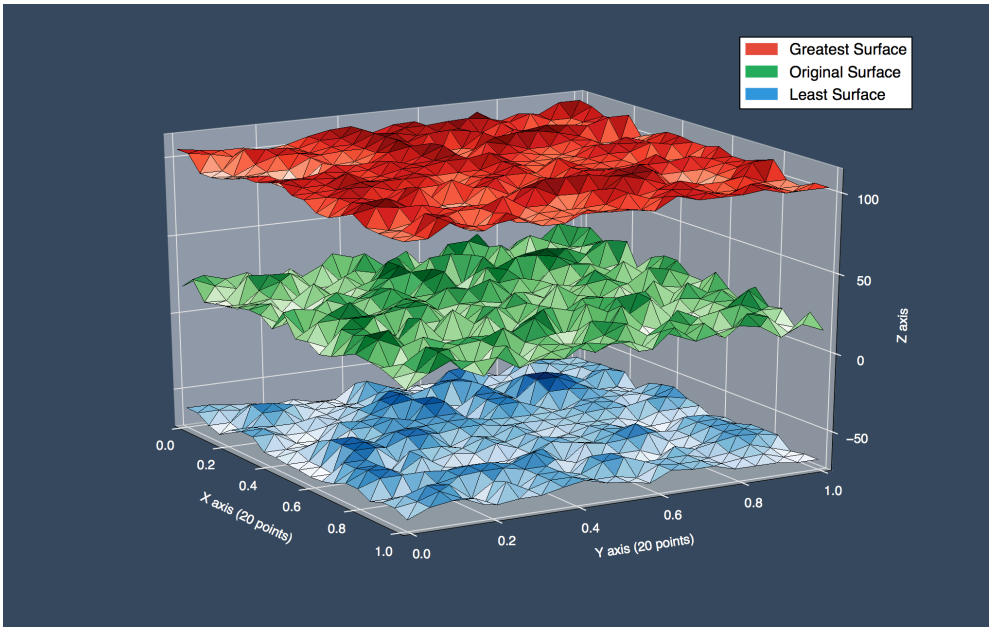


Figure 5.11: 20×20 grid and $\varepsilon = 80$

Upon increasing the grain size of the grid, the piecewise linear objects present a much richer geometrical structure, as we can examine in Figures 5.11 and 5.12, respectively.

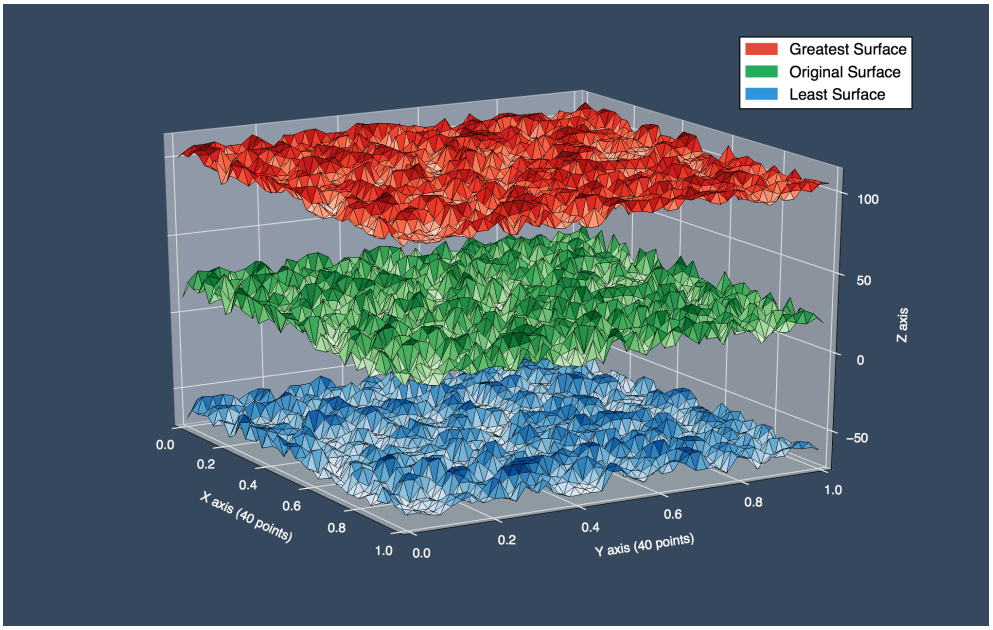


Figure 5.12: 40×40 grid and $\varepsilon = 80$

5.3 Challenges & Limitations

In this section, we will outline some of the limitations that arise in the implementation of our framework that decides consistency in the rectangular case. The major bottleneck concerns the size of the linear programming problem which involves no less than $p_1 \times p_2 \times \ldots \times p_n$ number of decision variables, where p_i is the number of points along the i^{th} axis, $1 \leq i \leq n$.

To show why this presents an enormous problem, we carried out a very simple performance analysis to evaluate the time spent in each of the core components of our implementation. We run 10 separate tests for each dimension ranging from $n = 2$ to $n = 9$, where we chose the hyper-rectangle $[0, 1]^n$ as the grid information – hence each LP problem involves 2^n decision variables. The averaged results across all of these experiments are shown in Figure 5.13 below:

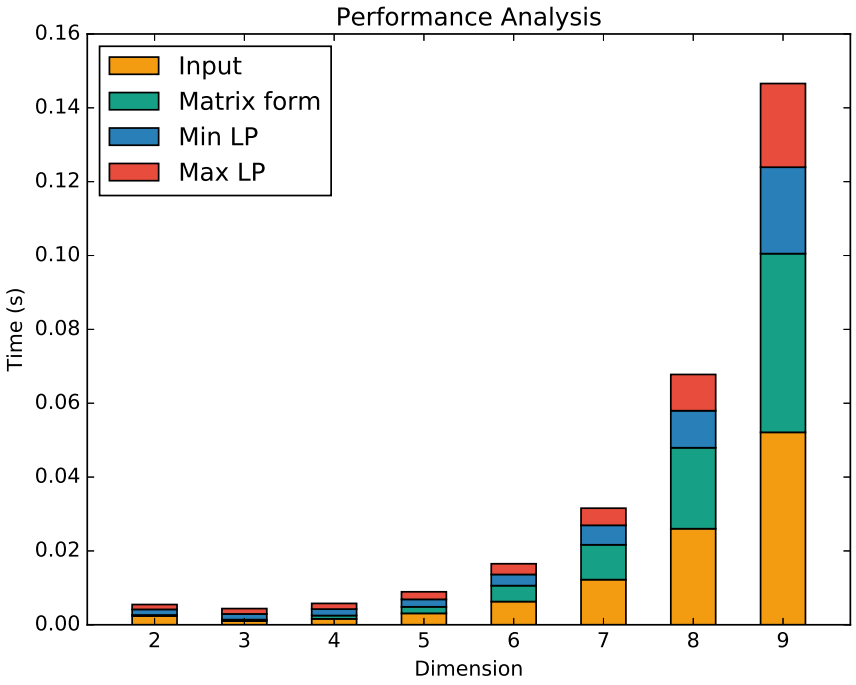


Figure 5.13: Performance analysis for dimensions $n = \overline{2, 9}$, where the grid coincides with the n-dimensional unit square. Measurements are averaged across 10 separate runs using randomly generated input.

Even in this simplest scenario where we have the least number of points along each axis, i.e. $p_i = 2$, for $1 \leq i \leq n$, we can notice that the total time spent within each of the main components (input generation, construction of matrix form $\boldsymbol{Gx} \leq \boldsymbol{b}$ and both LP algorithms) nearly doubles with every increment of the domain dimension n. This is clearly expected, as we need to build an exponentially bigger input which will also reflect in the size of the resulting linear programming problem.

However, as it obvious from the previous plot, the time spent in the components prior to invoking the CVXOPT solver dominate more than half of the total running time. Thus, there is a scope for improvement in the areas of input generation and construction of final matrix form, which would involve careful parallelisation of the current codebase. The main challenge here would be to decompose the overlapping structure of the function and derivative approximation into disjoint components, as most of the heights use information not only from their designated sub-hyper-rectangle, but also from the adjacent ones.

Chapter 6

Conclusion

For the most part, we consider this project to have been a success. Using very rigorous mathematical tools, we showed that the problem of consistency for a pair of interval-valued function and rectangular derivative approximation, defined on an n-dimensional domain, $n \geq 2$, reduces to whether a finite set of inequalities are simultaneously satisfied. In addition, we also proved that the minimal and maximal surfaces that are witnessing consistency can be derived by means of a linear programming algorithm, which provides a practical setting for implementation.

By leveraging the Python programming language, we successfully developed a framework which implements the above linear test for consistency, given constraints for both function and derivative information within a partitioning of the n-dimensional unit hyper-rectangle. Moreover, the least and greatest surfaces are also determined whenever the provided input is found to be consistent. In order to provide a better insight into the problem, we implemented a simple graphical user interface which can depict the 3D piecewise linear surfaces whenever there is a consistent pair defined in the unit square $[0, 1] \times [0, 1]$.

Finally, we spent a great deal of time developing an automatic testing framework that was crucial for assessing the correctness of our implementation. At the same time, we do not rule out the possibility of our implementation being completely error-free; however, we believe that the absence of any counterexamples in the sustained testing performed up to 6^{th} dimension is convincing enough to guarantee a solid and reliable end-product.

6.1 Future Work

As an open-ended project, we suggest a list of possible ideas in which the present work can be further extended:

- A faster method of input generation and translation of the linear programming problem into matrix form would be highly desirable. This may allow further test cases to be more quickly generated and verified.

- Extend the current framework to implement the algorithms that decide consistency in the case of a triangle/convex quadrilateral subject to convex derivative constraints.
- Also, examine if the latter algorithms in the convex setting can be extended to higher dimensions, similar to the rectangular case.

Bibliography

[1] A. Edalat, M. Krznarić, and A. Lieutier. Domain-theoretic solution of differential equations (scalar fields). In *Proceedings of MFPS XIX*, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2003. `www.entcs.org/files/mfps19/mfps19.html`, full paper in `www.doc.ic.ac.uk/~ae/papers/scalar.ps`.

[2] A. Edalat, A. Lieutier, and D. Pattison. A computational model for multi-variable differential calculus. *Information and Computation*, 224: 23–45, 2013.

[3] C. K. Yap and T. Dubé. The exact computation paradigm. *D.-Z. Du, F.K. Hwang (Eds.), Computing in Euclidean Geometry, World Scientific Press*, pages 452–486, 1995.

[4] C. K. Yap. Towards exact geometric computation. *Computational Geometry: Theory and Applications*, 7(1-2):3–23, 1997.

[5] Abbas Edalat and Reinhold Heckmann. Computing with real numbers - i. the lft approach to real number computation - ii. a domain framework for computational geometry. In *PROC APPSEM SUMMER SCHOOL IN PORTUGAL*, pages 193–267. Springer Verlag, 2002.

[6] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer Verlag, 3 edition.

[7] James. K. Strayer. *Linear Programming and Its Applications*. Springer Science+Business Media New York, 1989.

[8] GLPK (GNU Linear Programming Kit). URL `https://www.gnu.org/software/glpk/`. Accessed: 2016-01-20.

[9] CVXOPT (Python Software for Convex Optimization). URL `http://cvxopt.org`. Accessed: 2016-01-20.

[10] Abbas Edalat. Domain theory and fractals, 1999. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.1725&rep=rep1&type=pdf`. Accessed: 2015-11-4.

[11] Samson Abramski and Achim Jung. Domain theory. URL `http://www.cs.bham.ac.uk/~axj/pub/papers/handy1.pdf`. Accessed: 2015-10-28.

[12] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, 1983.

[13] Abbas Edalat. A continuous derivative for real-valued functions. In *S. B. Cooper, B. L öwe, and A. Sorbi, editors, New Computational Paradigms, Changing Conceptions of What is Computable*, pages 493–519. Springer, 2008.

[14] Abbas Edalat. Decidability of consistency for a triangle and convex quadrilateral. Unpublished note.

[15] Numpy. URL `http://www.numpy.org`. Accessed: 2016-05-08.

[16] Sympy. URL `http://www.sympy.org/en/index.html`. Accessed: 2016-05-08.

[17] Matplotlib. URL `http://matplotlib.org`. Accessed: 2016-05-08.

[18] Delaunay triangulation. URL `http://uk.mathworks.com/help/matlab/math/delaunay-triangulation.html`. Accessed: 2016-05-12.

[19] T. Murnane, K. Reed, and R. Hall. On the learnability of two representations of equivalence partitioning and boundary value analysis. In *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, pages 274–283, April 2007. ISSN 1530-0803.