Imperial College London

Department of Computing

Interim Report

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

Author:
Constantin Mateescu

Supervisors:
Prof. Abbas Edalat
Dr. Mahdi Cheraghchi

January, 2016

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. From electrical engineering and mechanics, to economics and molecular modelling, all these sciences have taken advantage of the latest developments in the area of mathematical optimisation. The latter is the process by 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 deal with an enormous amount of computation, the problem of precision (i.e. accurate calculation) is still one of great importance 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 applications cannot accept such rounding errors as they may well 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. They 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. Furthermore, the two maps have the additional property of being continuous and piecewise linear, making them suitable to be determined by means of a linear programming algorithm, as presented in [1].

A related problem to the one described above formulated again in [1], [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 called *consistent* if one can find a third map h which is approximated by the first component and whose derivative information is approximated by the second component of the pair. Assuming that f is defined by lower and upper constraints on its value and g is constrained to lie within rectangles (or hyper-rectangles, if referring to higher dimensional spaces) then a *consistency witness* h, which is piecewise linear, is guaranteed to exist. In addition, one can determine the least and greatest piecewise linear maps consistent with the information from f and g via a linear programming algorithm as shown in both [1] and [2].

1.2 Objectives

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

- Firstly, to implement the above optimisation problem in the form of a linear programming algorithm when the derivative constraints are specified as hyper-rectangles and 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.

The final outcome of this project should be in the form of a linear programming framework that implements the aforementioned algorithms and allows a user to determine the consistent piecewise linear map given some input functions f and g (the function and derivative information, respectively).

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. $1.2^34 = 1.2224$):

$a_{100}^{(5)}$	100.0^41
$a_{100}^{(30)}$	$100.0^{29}1$
$a_{100}^{(60)}$	$100.0^{57}997$
$a_{100}^{(100)}$	$100.0^{17}98\ldots$

$a_{100}^{(110)}$	100.0^792
$a_{100}^{(120)}$	$-3.790\ldots$
$a_{100}^{(130)}$	$5.9^78697\ldots$
$a_{100}^{(140)}$	$5.9^787925\ldots$

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 Canonical Forms for LP Problems

As already discussed, there are two ways to define a linear programming problem: either as a minimisation or maximisation problem. The canonical forms for each of those is presented below along with some standard terminology [6].

Definition 2.2.1. A problem is said to be a *canonical maximisation linear programming problem* if it has the following form:

$$\begin{array}{ll} \text{Maximise} & f(x_1, x_2, \ldots, x_n) = c_1x_1 + c_2x_2 + \ldots + c_nx_n - d \\ \text{subject to} & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \qquad\qquad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & x_1, x_2, \ldots x_n \geq 0. \end{array}$$

Definition 2.2.2. A problem is said to be a *canonical minimisation linear programming problem* if it has the following form:

$$\begin{array}{ll} \text{Minimise} & g(x_1, x_2, \ldots, x_n) = c_1x_1 + c_2x_2 + \ldots + c_nx_n - d \\ \text{subject to} & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq b_2 \\ & \qquad\qquad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq b_m \\ & x_1, x_2, \ldots x_n \geq 0. \end{array}$$

In both canonical forms, the first m constraints are said to be *main constraints*, while the second n constraints are called *non-negativity constraints*.

Definition 2.2.3. The linear function f and g in Definition 1.1 and 1.2 are said to be *objective functions*.

Definition 2.2.4. The set of all points $(x_1, x_2, \ldots, x_n)$ satisfying the $m + n$ constraints of a canonical maximisation/minimisation linear programming problem is said to be the *constraint* set of the problem. Any element of the constraint set is said to be a *feasible point* or *feasible solution*.

Definition 2.2.5. Any feasible solution of a canonical maximisation (respectively minimisation) linear programming problem which maximises (respectively minimises) the objective function is said to be an *optimal solution*.

2.2.2 Polyhedral Convex Sets

Let us state some additional results that are essential towards assessing the optimality of feasible solutions in linear programming (see also [6]).

Definition 2.2.6. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$. Then the *line segment* between $\mathbf{x}$ and $\mathbf{y}$ (including the endpoints) is given by:

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

Definition 2.2.7. Let $S \subset \mathbb{R}^n$. The set S is said to be *convex* if for all $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in S$ it holds that:

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

Definition 2.2.8. 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

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

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

Theorem 2.2.1. *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 2.2.9. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. The norm of $\mathbf{x}$, denoted $\|\mathbf{x}\|$, is given by:

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

Definition 2.2.10. Let $r \geq 0$. The set of points $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ such that

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

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

Definition 2.2.11. 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 2.2.12. 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 $\mathbf{x}, \mathbf{y} \in S$ and $\alpha \in (0, 1)$ such that

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

Theorem 2.2.2. *If the constraint set S of a canonical maximisation/minimisation linear programming program is bounded, then the maximum/minimum value of the objective function is attained at an extreme point of S.*

Theorem 2.2.3. *If the constraint set S of a canonical maximisation/minimisation linear programming problem 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.*

2.2.3 A Geometric Approach

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

Example 2.2.3.1. *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:

$$x_1 \geq 0,$$
$$x_2 \geq 0.$$

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:

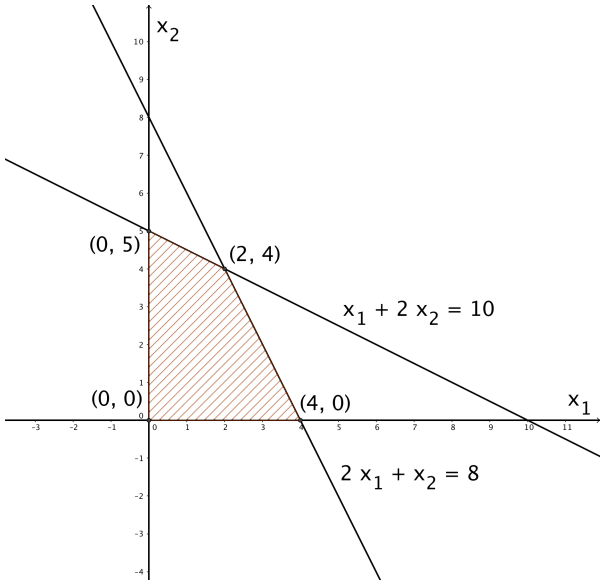

Figure 2.1: Feasible set of points satisfying all constraints

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

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

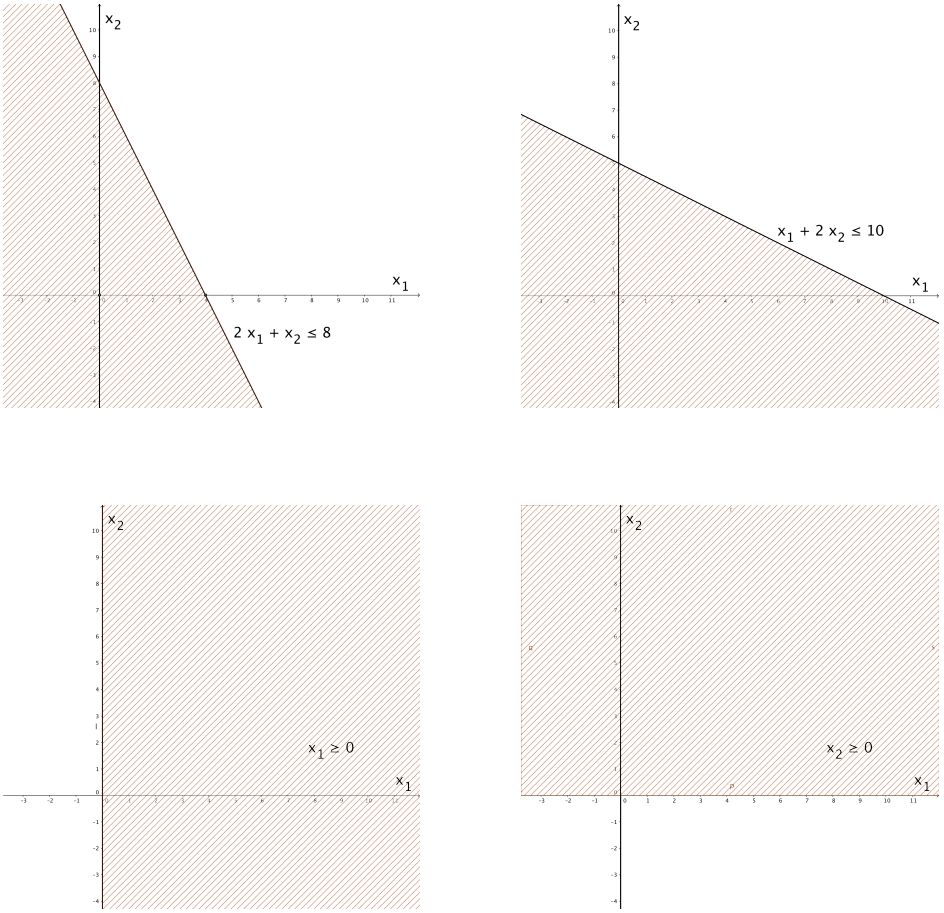

Figure 2.2: All individual constraints

The region in Figure 1 above was obtained by intersecting the four 2-dimensional regions determined by the constraints $2x_1+x_2 \leq 8$, $x_1+2x_2 \leq 10$, $x_1 \geq 0$, 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 2.2.2 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).

2.2.4 Issues 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 2.2.3 we would also need to ensure that the objective function is bounded by above or below, 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.

Finally, we end this section on linear programming noting that several algorithms have been developed to deal with the issues that we have just noted. The famous *simplex algorithm* is able to find optimal solution 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 [6]). Several other optimisations such as branch and bound and cutting plane algorithms can further be used to further enhance the solution finding to LP problems [7].

2.3 CVXOPT Framework

For the purposes of implementing the consistency properties as described in the introduction section, we decided to use an open-source framework named CVXOPT [8]. The main reason for choosing this software package was that it allows straightforward development of linear/convex optimisation problems using the Python programming language. This framework can be used either via an interactive Python interpreter or can be integrated within existing Python modules.

Previous experience with GNU's GLPK package [9], which uses a rather obfuscated syntax with many intricate constructs, made us choose an alternative package which can leverage the strengths of the Python programming language.

2.3.1 Simple Example

We can illustrate the strengths of the CVXOPT framework with a simple example. Given the following minimisation 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 can retrieve the optimal solution vector $\mathbf{x}$ using the following set of commands:

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

     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 Results

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 *cpo* (*complete partial order*) is a poset with a least element denoted $\bot$ such that every chain $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ has a least upper bound, denoted by $\bigsqcup_{i \geq 0} d_i$ or, more conveniently, $\bigsqcup_i d_i$. Hence, $\bigsqcup_i d_i$ gives precisely the total information contained in the chain (and no more).

Definition 2.4.4. 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 2.4.5. A *directed complete partial order* (*dcpo*) or a domain is a partial order in which every directed subset has a least upper bound (lub).

Definition 2.4.6. A dcpo is said to be *pointed* if it has a least element which is denoted by $\perp$ and is called bottom.

Definition 2.4.7. 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 $b \in A$ there exists $c \in A$ with $a \sqsubseteq c$.

Definition 2.4.8. 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 $x = \bigsqcup B_x$.

Definition 2.4.9. A dcpo with a (countable) basis is said to be an *(ω)-continuous* domain.

Definition 2.4.10. 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 2.4.11. 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].

2.5 Consistency of two functions

In this section we focus on the main idea of the proposed project, which involves the idea of *consistency* of two given maps subject to some constraints. We begin by stating the main notations and terminologies used throughout this section. We then present the general results for the n-dimensional case and illustrate with examples for the particular 1D and 2D cases [1], [2].

2.5.1 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 $f(x) \neq \perp$ 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 $\perp$ 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_{χ_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, we introduce the following definition of *interval Lipschitz constant* functions which generalises the well-known *Lipschitz functions*:

Definition 2.5.1. The continuous function $f : U \to \mathbb{R}$ has a *non-empty, convex and compact set-values Lipschitz constant* $b \in \mathbf{C}\mathbb{R}^n$ in an open subset $a \subset U$ if for all $x, y \in a$ we have: $b \cdot (x - y) \sqsubseteq f(x) - f(y)$. The single step tie $\delta(a, b) \subseteq C^0(U)$ of a with 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 b as non-empty convex compact set-values Lipschitz constant in a.

2.5.2 The property of consistency

In simple terms, a pair of functions (f, g), representing function and derivative information respectively, is called *consistent* if one can find a third map h which is approximated by the first component and whose derivative information is approximated by the second component of the pair.

The function information is given by a number of *step functions* whose values are given by a finite set of pairs $(a_i, b_i)_{i \in I}$, 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. The derivative information for the n partial derivatives is given as a finite set of pairs $(a_i, b_i)_{i \in I}$, where a_i are as above, but b_j represent rational compact polyhedra.

In the particular case when the derivative constraints lie within hyper-rectangles with faces parallel to the coordinate planes, there is always a *piecewise linear witness* for consistency. In addition, one can find the least and greatest piecewise linear maps consistent with the information from f and g via a linear programming algorithm as described in both [1] and [2].

We now formalise the *consistency* relation under the definitions and notations used thus far:

Definition 2.5.2. The consistency relation $\mathsf{Cons} \subset D^0(U) \times (U \to \mathbf{C}\mathbb{R}^n)$ is defined by:

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

For a consistent (f, g), we think of f as the *function part* or the *function approximation* and g as the *derivative part* or the *derivative approximation*. It can be shown that the Cons is Scott closed [2]. Also, in the one-dimensional case it holds that (Section 2, [1]):

Theorem 2.5.1. *The following duality property holds:*

$$h \in \int g \iff g \sqsubseteq \frac{dh}{dx}.$$

Furthermore, for a n-dimensional space, $n \geq 2$, we have that (Corrolary 2.8 from [2]):

Theorem 2.5.2. *The following duality property holds:*

$$h \in \int g \iff g \sqsubseteq \mathcal{L}h.$$

Now, according to [1] and [2], we have the least and upper consistency witnesses defined by the following:

Proposition 2.5.1. *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.

Corollary 2.5.2.1. *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.

2.5.3 Consistency for the one-dimensional case

Figure 2.3 shows the piecewise linear maps s and t which provide the least and greatest bounds for the consistency witness, with the given function and derivative approximations (the latter being constrained to rectangles in the Euclidean plane).

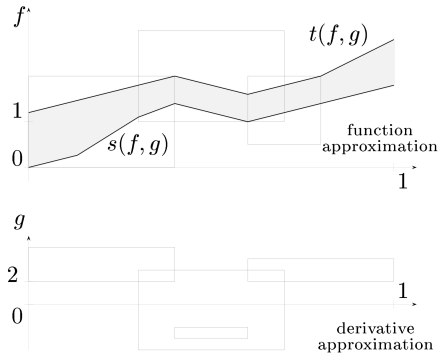

Figure 2.3: Consistency for the 1D case.

A linear algorithm in the number of partitions induced by (f, g) (i.e. the partition of points derived from intersecting all rectangle constraints, within the domain $[0, 1]$) is given in [1], Algorithm 3.3, to obtain the map $s(f, g)$ (a similar one computed the upper bound $t(f, g)$).

Algorithm 2.5.1. *The function updating algorithm consists of an initialisation step and two other main steps (see Figure 2.4). 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* $f_k^-(x) = a_k x + b_k$.

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 $\forall x \in [y_{k-1}, y_k)$
  $s(f, g)(x) := \max\{u(x), s(y_k) + (x - y_{k-1})e_k^+\}$

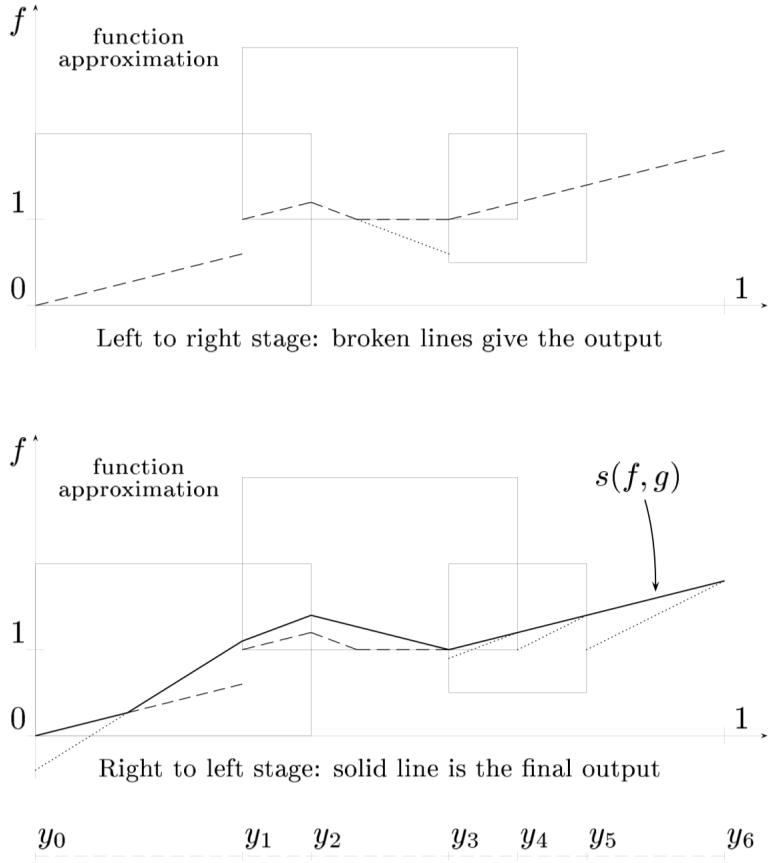

Figure 2.4: The function updating algorithm

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

The framework for studying consistency for an n-dimensional setup ($n \geq 2$) is much more challenging than what we have just seen already. In this section we describe the algorithm from [2] which works for the general case $n \geq 2$. However, for the purposes of presentation, we will only deal with the case when $n = 2$, which is easier to reason about (from a geometric perspective at least). Notice that, as with the 1D case, the derivative constraints are given as hyper-rectangles, to simplify the computational framework.

Figure 2.5 shows two examples of consistent tuples, namely the least and greatest witnesses that enforce the consistency property described so far. In the example from the left hand side, there is one hyper-rectangle for the function approximation, while the derivative approximation in the x and y directions is given my the constant intervals $[n, N]$ and $[m, M]$, where

$n, m > 0$. In the other instance, on the right, there are two intersecting hyper-rectangles for the function approximation and the derivative approximations are the constant intervals $[0, 0]$ and $[m, M]$, where $m > 0$.

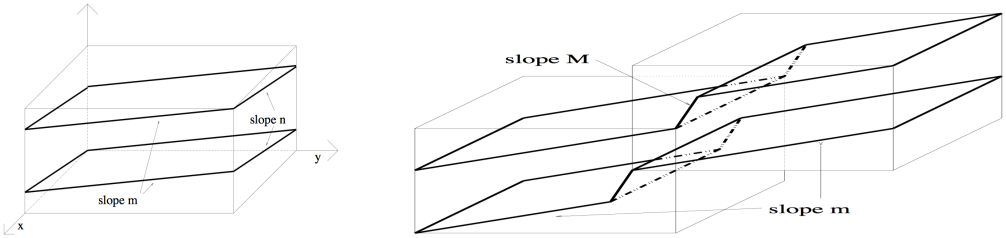

Figure 2.5: Consistency for the 2D case

We now proceed to explaining the framework, as presented in [2], Section 3. Let us introduce the following linear programming algorithm:

Algorithm 2.5.2. *In a similar fashion to the induced partitions for the 1D case, we impose a grid* $(p_0, \dots p_k) \times (q_0, \dots q_l)$ *on the unit square* U *such that the function approximation given by the step function* $f : U \to \mathbf{I}\mathbb{R}$ *and the derivative approximation given by the step function* $g : U \to \mathbf{C}\mathbb{R}^n$ *are constant respectively with values* $c_{ij} \in \mathbf{I}\mathbb{R}$ *and* $b_{ij} \in \mathbf{C}\mathbb{R}^n$ *inside every sub-rectangle* $(p_i, p_{i+1}) \times (q_j, q_{j+1})$*, for* $i = \overline{0, k-1}$ *and* $j = \overline{0, l-1}$*, defined by adjacent grid points. Note that if* $c_{ij} = \perp$ *or* $b_{ij} = \perp$ *then* f *or* g *are undefined in the sub-rectangle and their contribution can be ignored in the following analysis.*

If the pair of step functions $(f, g) \in (U \to \mathbf{I}\mathbb{R}) \times (U \to \mathbf{C}\mathbb{R}^n)$ *is consistent then we can find values* $h_{i,j} \in \mathbb{R}$ *at all the grid points* (p_i, q_j) *for* $i = \overline{0, k}$ *and* $j = \overline{0, l}$*, such that for* $i = \overline{0, k-1}$ *and* $j = \overline{0, l-1}$ *they satisfy the following two conditions:*

1. $c_{ij}^- \leq h_{s,t} \leq c_{ij}^+$*, for* $s = i, i+1$ *and* $t = j, j+1$*;*
2. $c := \left(\dfrac{h_{i+1,j} - h_{i,j}}{p_{i+1} - p_i}, \dfrac{h_{i,j+1} - h_{i,j}}{q_{j+1} - q_j} \right) \in b_{ij}^1 \times b_{ij}^2 = b_{ij}$.

Chapter 3

Project Plan

This section provides information on the next steps of the project, and the order in which the implementation tasks will be handled. Given the reading I have carried out so far, as detailed in the background section above, I expect to complete the project in the following stages:

1. Start with the 1D case for consistency, when derivative constraints lie within rectangles and implement the linear programming algorithm as presented in section 3.3 from [1];

2. Move on towards the nD case for consistency, $n \geq 2$, when the restrictions for the derivative information are given by compact hyper-rectangles with faces parallel to the coordinate planes. The algorithm for this more generic case is presented in [2];

3. Implement the 2D case for consistency, when the constraints lie in convex (hyper)-quadrilaterals, for which a known linear programming algorithm exists;

4. Finally, investigate if the latter case can be generalised to convex polygons/polyhedra constraints and whether the result still holds in higher dimensional spaces.

Below is a table giving the timeline I expect to follow for each of the above-mentioned tasks, until the project deadline. Any incomplete work by the Easter break will need to be completed during the holiday, to ensure the completion of all milestones of the project.

Term & Weeks	**Dates**	**Description**
Term 2, Weeks 4-5	1 FEB - 14 FEB	Starting first task (consistency for the 1D case)
Term 2, Weeks 6-7	15 FEB - 28 FEB	Proceeding to the second task (consistency for the n-dimensional case, $n \geq 2$)
Term 2, Weeks 8-9	29 FEB - 13 MAR	Moving on to a more general case as part of task 3 (consistency for the 2D case with convex quadrilateral constraints for derivatives)
Term 2, Weeks 9-11	14 MAR - 27 MAR	Exams Preparation
Easter Break	28 MAR - 24 APR	Code tweaking, refining and catching up on any implementation that is left incomplete
Term 3, Weeks 1-4	25 APR - 22 MAY	Investigating the most challenging generalisation as described in task 4 (consistency in higher dimensions, where derivative constraints are specified as convex polyhedra, rather than hyperrectangles/quadrilaterals)
Term 3, Weeks 5-7	23 MAY - 12 JUN	Consolidating final report, findings and evaluation
Term 3, Weeks 8-9	13 JUN - 24 JUN	Final report submission and presentation

Chapter 4

Evaluation Plan

In terms of evaluation, the implementation of the linear programming algorithms must be verified against a relevant and thorough test-suite. After each stage of the project, a number of test-suites will be written so as to validate the work and make sure I can proceeded confidently to the next challenges.

The plan is to also validate some of the results presented in [1], [2] in order to have a satisfying product by the end of the project. In addition, it would probably be worthwhile to have some sort of GUI to demonstrate with relevant examples the implementation of the linear programming algorithm. This will be useful, for example, at the end of task 2, when we want to find the minimal and maximal bounding surfaces, once the consistent witness is determined.

In addition, when the linear programming framework will be fully implemented at the end of stage 3, it should become relatively straightforward to extend it in order to assess whether the consistency property holds in more general settings. Assuming the results hold for convex shaped objects and a consistent witness can be determined, we will be able to conjecture or even prove our findings. Should counterexamples be found, we can be certain that the problem of consistency will not hold for those specific cases.

Finally, the framework to be developed should be convenient to use to allow further extensions to be carried out easily in the future. This could include, as mentioned before, some graphical additions to allow the visualisation of the piecewise linear maps. In order to enhance the overall use of our framework, we may also consider ways to add more constraints at runtime or modify input parameters on-the-fly and have the results displayed in real-time.

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

[7] L. R. Foulds. *Optimization Techniques*. Springer Verlag New York, 1981.

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

[9] GLPK (GNU Linear Programming Kit). URL `https://www.gnu.org/software/glpk/`. 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.