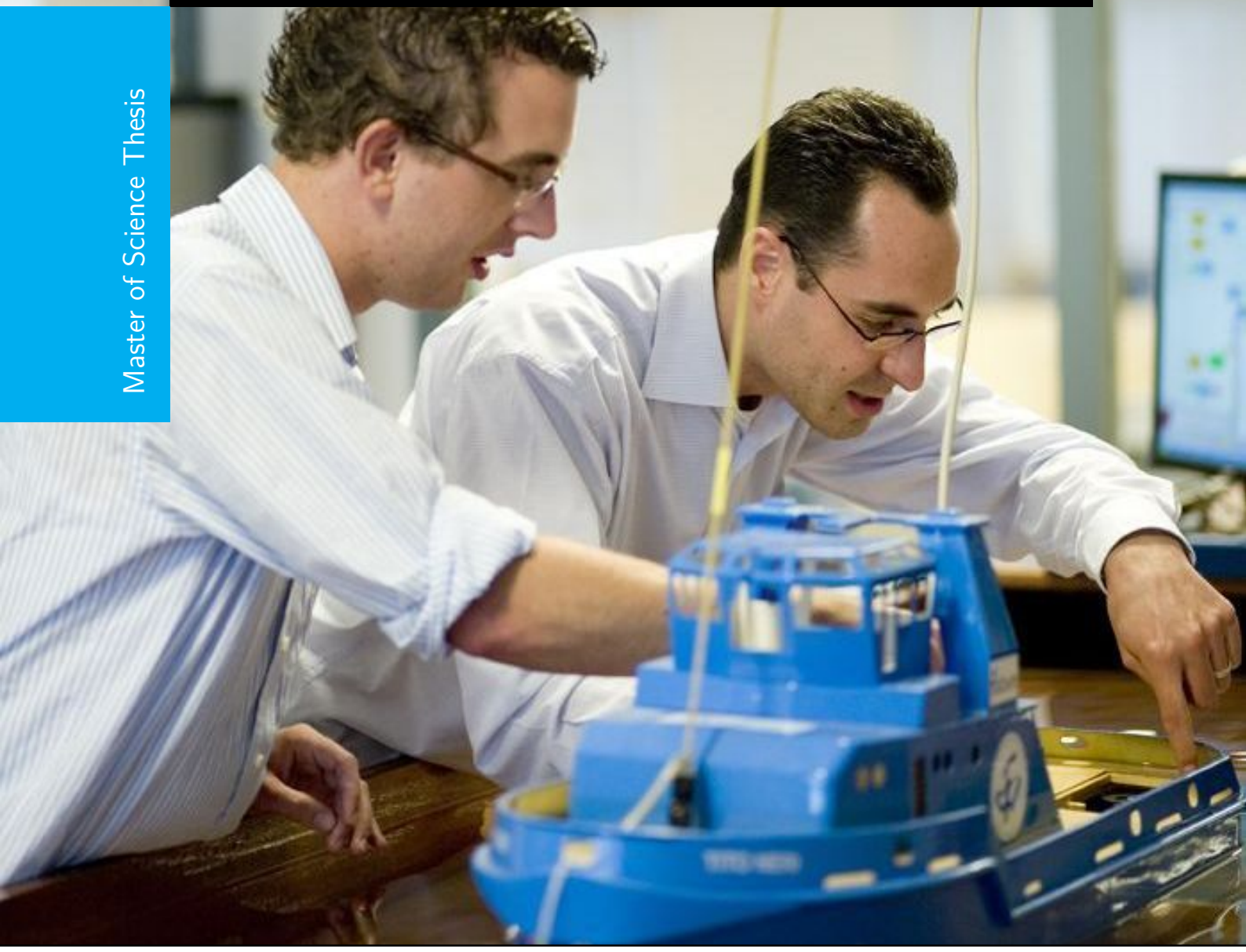
Canonical MMPS realisations using finest-base-region partitioning

F.T. Gallagher

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Canonical MMPS realisations using finest-base-region partitioning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

F.T. Gallagher

February 22, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Canonical MMPS realisations using finest-base-region partitioning

by

F.T. Gallagher

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: February 22, 2023

Supervisor(s): _____
dr.ir. A.J.J van den Boom

Reader(s): _____

Abstract

In this thesis a novel method for the realisation of Max-Min-Plus-Scaling (MMPS) functions is presented. It has previously been shown that continuous piecewise affine (PWA) functions, conjunctive MMPS functions (also lattices- or min-max functions) and kripfganz MMPS functions (difference between two convex functions) can each describe the same function. Each form has its own specific use cases and benefits and thus it may be desired to rewrite a specific function described in one form, into another. Current techniques are input dependent and not available for each combination, while some techniques blow the number of parameters. The technique presented in this thesis however is input and output independent and rigid in its construction. It fills up the gaps where some realisation were not possible yet, it generates a rigid and predictable output.

The necessary and sufficient conditions for the existence of each form are proposed. Additionally, it is explained how redundant terms may be removed in order to ensure a minimal representation. An algorithm is given for the decomposition and for the construction of each canonical form and supported with some worked examples. The decomposition provides the tools to efficiently map between different descriptions.

Table of Contents

Chapter 1

Introduction

1-1 Background

Piecewise Affine (PWA) function is a nonlinear function with affine components defined on polyhedral regions. Continuous PWA functions can be found in a variety of fields and applications such as control systems, optimization, robotics and machine learning to name a few. It is a very intuitive method for approximating nonlinear functions (and systems), but is also an excellent tool to model hybrid systems. Generally the steps to solve the continuous PWA function is to identify which polyhedral region is active and then use the corresponding affine function. Interestingly, although continuous PWA functions are defined as multiple individual affine sections, there exist analytical expressions for continuous PWA functions. These functions are called Max-Min-Plus-Scaling (MMPS) functions.

MMPS functions follow a recursive grammar where they are constructed using the operations maximization, minimization, addition and scalar multiplication. The MMPS framework an extremely useful in various fields and problems such as explicit Model Predictive Control (MPC), dynamic programming, modelling of Discrete Event Systems (DES), DC programming, neural networks, parametric linear programs [14, 26, 32, 7, 22] to just name a few. The MMPS framework can be used to model discrete phenomena such as concurrency, synchronisation and event triggered systems.

The equivalence between MMPS functions and continuous PWA functions has been exploited in multiple fields, mostly motivated by the fact that the solution of an MMPS function is much easier to obtain than it is for a continuous PWA function. For example the control law of an implicit MPC is usually obtained as a continuous PWA controller. MPC is a powerful framework, but generally constrained by the online computational complexity. The control law can then be written as an MMPS expression creating is much more feasible control law [28]. In the field of hybrid systems the equivalence between continuous PWA and MMPS classes, including three other classes (Mixed Logical Dynamical (MLD) systems, Extended Linear Complementarity (ELC) systems and Linear Complementarity (LC) systems) was established in [13]. It was postulated that the equivalences can be used to transfer established knowledge between the classes, thus gaining knowledge of each individual class.

There are multiple methods available in literature that can transform a continuous PWA function into a MMPS function. Either the continuous PWA function is decomposed into two maximization (convex) functions, or as the minimization of multiple maximization terms. However the existing methods, do not cover all possible transformations and usually the methods are for specific transformation. The goal of this thesis is to bridge all transformations using a universal method.

1-2 Motivation and major work

Motivated by the importance of being able to transform between the continuous PWA and MMPS classes, the topic of this thesis are the methods for these transformations. The main research goals of this thesis are:

- How can we transform any MMPS function into a continuous PWA function and vice versa?
- How would we implement such a method in an algorithm?

In other words, can we find a "universal method" that can transform any type of expression into another? And how would this be implemented in an algorithm? The major work of this thesis answer both of these questions. A partitioning method based on hyperplane arrangements lies at the heart of this method. Three algorithms are presented that can compute this partitioning. The structure of this thesis is given in the next section.

1-3 Organization

The first two chapters of this thesis are a review of existing research on continuous PWA functions and MMPS functions. In Chapter 2 an introduction to continuous PWA functions and MMPS functions is given. The core concepts, use cases and subclasses are discussed. Emphasis is put on the section of canonical MMPS functions as this is the main topic of this thesis. In Chapter 3 the techniques for transforming continuous PWA functions and MMPS functions into one another that can currently be found in existing literature is presented and discussed.

The main contribution of this thesis can be found in Chapters 4 and 5, where a novel method for transforming between continuous PWA functions and MMPS functions is presented. In Chapter 4, a domain partitioning method based hyper-plane arrangements is introduced, which forms the basis of the method. In Chapter 5 three algorithms for the realisation of the partitioning is discussed. The three algorithms are analysed for their performance and compared to each other. Examples are given for the partitioning method and each algorithm.

In Chapter 6 the new partition and algorithm are discussed. In Chapter 7 conclusions are drawn and a final section for possible future work related to MMPS and this thesis is presented.

A Matlab based MMPS toolbox has been created in the making of this thesis. Documentation can be found in Appendix C. The reader is referred to the appendix for MMPS algebraic rules in Appendix A, Polyhedral Theory in Appendix B and Matlab source code for the algorithms and toolbox in Appendix C.

Chapter 2

MMPS functions and Continuous PWA functions

In this first chapter we will review the concepts of continuous Piecewise Affine (PWA) functions and Max-Min-Plus-Scaling (MMPS) functions. This includes their definition, canonical forms (MMPS), the equivalence between the functions and their applications. We will start with continuous PWA functions as these are the most intuitive. From there we will bridge over to MMPS functions and the equivalences.

2-1 Continuous Piecewise Affine functions

A piecewise affine (PWA) function is a function that is defined by multiple affine segments, each with its own slope and intercept. The function can be thought of as being constructed by gluing together these linear segments at certain points.

Generally Piecewise affine functions are a composition of affine functions on a polyhedral partition given by

$$f(x) = f_i(x) := \alpha_i(x) + \beta_i, \quad x \in \Omega_i \tag{2-1}$$

where $f_i : \Omega_i \to \mathbb{R}$ with $\Omega_i \subseteq \mathbb{R}^n$ is called the *local function* on the convex polyhedral region Ω_i. The regions form a partition of the convex polyhedral set $\Omega := \text{dom}(f)$, i.e. $\text{int}(\Omega_i) \cap \text{int}(\Omega_j) = \emptyset$ for all $i \neq j$ and $\bigcup_{i=1}^{n_r} \Omega_i = \Omega$, where n_r is the total number of regions.

An important class of PWA functions are those that are continuous on their domain. A continuous PWA function is a PWA function in which the different affine segments are joined together in such a way that there are no jumps or discontinuities at the transitions. In other words, the function is defined and continuous over the entire domain. Continuous PWA functions are defined as:

Definition 2-1.1. [6] *Continuous piecewise-affine function.* A function $f : \Omega \to \mathbb{R}$, with $\Omega \subseteq \mathbb{R}^n$, is said to be a continuous piecewise-affine (PWA) function if and only if the following conditions hold:

1. The domain space Ω can be divided into a finite number of regions which are expressed as convex polyhedra as, $\Omega = \bigcup_{i=1}^{N} \Omega_i, \quad \Omega_i \neq \emptyset$, where N is the total number of regions. The polyhedra are closed and have non-overlapping interiors, i.e. $\text{int}(\Omega_i) \cap \text{int}(\Omega_j) = \emptyset, \quad \forall i, j \in \{1, ..., N\}, i \neq j$. The boundries of each polyhedra region are of $(n-1)-$dimensional hyperplane.

2. For each region Ω_i, f can be expressed as:

$$f(x) = \ell_{\text{loc}(i)}(x) := \alpha_{(i)}^{\text{T}} x + \beta_{(i)}, \quad \forall x \in \Omega_i \tag{2-2}$$

   where we refer to $\ell_{\text{loc}(i)} : \Omega_i \to \mathbb{R}$ as the *local function*, with $\alpha_{(i)} \in \mathbb{R}^n$ and $\beta_{(i)} \in \mathbb{R}$

3. f is continuous on any boundary between two adjacent regions

Note that the local affine function $\ell_{\text{loc}(i)}$ may be the same for different regions, i.e.

$$\ell_{\text{loc}(i)}(x) = \ell_{\text{loc}(j)}(x), \quad \forall x \in \Omega \tag{2-3}$$

or simply

$$\text{loc}(i) = \text{loc}(j) \tag{2-4}$$

Suppose there are M distinct affine functions, i.e., $\text{loc}(i) \in \{1, ..., M\}$, $\forall i \in \{1, ..., N\}$, we have $M \leq N$.

Example 1. Let us demonstrate a continuous PWA function with an example. Consider a 1-dimensional continuous PWA function with 5 local affine functions, defined over the interval $[0, 5]$:

$$f_{\text{pwa}}(x) = \begin{cases} \ell_1(x) = 0.5x + 0.5 & x \in \Omega_1 \\ \ell_2(x) = 2x - 1 & x \in \Omega_2 \\ \ell_3(x) = 2 & x \in \Omega_3 \\ \ell_4(x) = -2x + 9 & x \in \Omega_4 \\ \ell_5(x) = -0.5x + 3 & x \in \Omega_5 \end{cases} \tag{2-5}$$

with the regions Ω_i:

$$\Omega_1 = [0.0, 1.0], \Omega_2 = [1.0, 1.5], \Omega_3 = [1.5, 3.5], \Omega_4 = [3.5, 4.0], \Omega_5 = [4.0, 5.0]$$

The function produces the following plot:

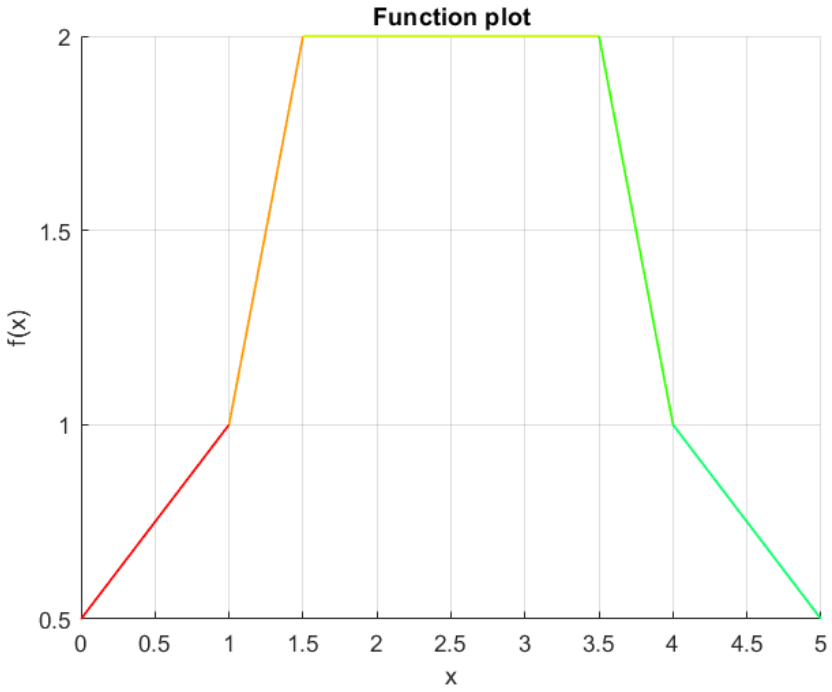


Figure 2-1: Plot of example 1, a 1-dimensional continuous PWA function. The function is defined over the interval [0,5]. There are 5 regions where a specific affine function is active.

2-1-1 Working with continuous PWA functions

Continuous PWA functions are defined by a set of affine functions and a convex polytope over which a certain affine function is active. Thus we need to store three types of information: the unique affine functions, the regions and which affine function is active over which region.

For a continuous PWA function in $\mathcal{R}^n$ with N_Ω regions described in H-representation the storage requirements for the regions are N_Ω number of matrices of size $N_{H_i} \times n$ and where N_{H_i} are the number of hyperplanes that defines the region N_{Ω_i}. For the unique affine functions N_{aff} in $\mathbb{R}^n$ can be stored into a $N_{\text{aff}} \times n$ matrix. There are different techniques to keep track which function is active over which region. For example one can store the active function inside a cell along side the region, or an index pointing to the matrix with unique affine functions can instead be stored with the region.

The MPT-toolbox [16] uses (nested) cells to store vectors and matrices. For the purpose of this paper, the MPT-toolbox is used to store and work with continuous PWA functions.

2-1-2 Applications of PWA functions

PWA functions are very intuitive and conceptually easy to understand. PWA functions can therefor be a powerful tool to approximate nonlinear functions. The key advantage of PWA functions is that they are relatively simple to work with and can provide a good approximation of more complex nonlinear functions [12, 14, 33, 28, 3]. One of the uses of PWA functions

is in the field of control systems. PWA models can be used to approximate the behavior of nonlinear systems, making them an excellent choice for control design [12, 15, 3]. This is done by partitioning the state space of the system into a number of regions, and then approximating the dynamics of the system within each region by a linear function. The resulting model is then a piecewise affine function, with each linear function corresponding to a different region of the state space.

Another application of PWA functions is in the field of machine learning, specifically in the context of neural networks [20]. It's known that neural networks are universal approximators, meaning they can approximate any function given enough hidden neurons, but when it comes to large-scale datasets and high-dimensional inputs, it's computationally infeasible[20].

In general, PWA functions are useful in any context where the goal is to approximate a complex nonlinear function with a simpler, piecewise affine function. They can provide a good trade-off between complexity and accuracy and are widely used in many fields such as optimization, machine learning, control systems, computer vision and more [20, 12, 3, 2].

2-2 Max-min-plus-scaling functions

With the introduction of PWA functions a jump is made to another type of functions, namely *max-min-plus-scaling functions*. As will be shown, continuous PWA functions and MMPS functions are canonically identical, i.e. each continuous PWA function has a MMPS function with the same output behaviour. More specifically, MMPS functions can be seen as an analytical expression of a continuous PWA function.

The *max-min-plus-scaling* framework may be viewed as a general description for functions using the operations maximization, minimization, addition and scaling. The term was first introduced in [8]. First define the following $\top = \infty$, $\varepsilon = -\infty$, $\mathbb{R}_\top = \mathbb{R} \cup \{\infty\}$, $\mathbb{R}_\varepsilon = \mathbb{R} \cup \{-\infty\}$ and $\mathbb{R}_c = \mathbb{R} \cup \{\infty\} \cup \{-\infty\}$. Furthermore $0 \cdot \varepsilon = 0$ and $0 \cdot \top = 0$ and $\top + \varepsilon = 0$. Then let the set $\mathcal{R}$ be any of three sets $\mathbb{R}_\top$, $\mathbb{R}_c$ or $\mathbb{R}_\varepsilon$. Then a MMPS function is defined as follows:

Definition 2-2.1. [8] (Max-min-plus-scaling function): A max-min-plus-scaling (MMPS) function $f : \mathcal{R}^n \to \mathcal{R}$ is defined by the recursive grammar

$$f(x) := x_i | \alpha | \max\Big(f_k(x), f_l(x)\Big) | \min\Big(f_k(x), f_l(x)\Big) | f_k(x) + f_l(x) | \beta f_k(x) \tag{2-6}$$

With $i \in \{1, 2, ..., n\}$, $\alpha \in \mathcal{R}, \beta \in \mathbb{R}$ and where f_k and f_l are again MMPS functions over the set $\mathcal{R}$. The symbol | stands for "*or*" and max and min are performed entry-wise.

For vector-valued MMPS functions $f : \mathcal{R}^n \to \mathcal{R}^m$ the above statements hold componentwise.

A MMPS function as defined above in Definition 2-2.1 can be considered in *general form* when the recursion does appear in a specific order. There are however two forms with specific recursion that can be considered *canonical*, i.e. any MMPS function using a specific recursion can be rewritten into these canonical form. The form are called:

- **Conjunctive & disjunctive MMPS functions** - This canonical form uses a single minimization and single maximization (and vice versa) in consecutive order, over a set of affine functions.

- **Convex & concave Kripfganz MMPS functions** - This canonical form consists of the difference of two convex or maximization functions (or two concave or minimization functions) of a set of affine functions.

2-2-1 Conjunctive & disjunctive MMPS functions

Conjunctive and disjunctive MMPS functions are functions that perform the operations minimization and maximization (or maximization and minimization) in consecutive order, of a set of affine functions.

Definition 2-2.2. [7](Conjunctive & disjunctive MMPS function) A scalar-valued MMPS function $f : \mathcal{R}^n \to \mathcal{R}$ is called a conjunctive MMPS function if it can be written in the form:

$$f(x) = \min_{i=1,...,K} \max_{j=1,...,n_i} (\alpha_{(i,j)}^{\mathrm{T}} x + \beta_{(i,j)}) \tag{2-7}$$

or into the disjunctive canonical form, for $f : \mathcal{R}^n \to \mathcal{R}$, $f(x) = \max_{i=1,...,L} \min_{j=1,...,m_i} (\gamma_{(i.j)}^{\mathrm{T}} x + \delta_{(i.j)})$ for some integers K, L, $n_1, ..., n_K$, $m_1, ..., m_L$, vectors $\alpha_{(i,j)}$, $\gamma_{(i,j)}$ and real numbers $\beta_{(i,j)}$, $\delta_{(i,j)}$. For vector-valued MMPS functions the above statements hold componentwise. (Or alternatively, $\alpha_{i,j}$ are matrices, and $\beta_{i,j}$ are vectors).

In literature these type of functions can also be found under the name *min-max* and *max-min* functions or *lattice representation*. The latter is usually used in the context of *continuous piecewise affine functions* which will be discussed in the forth coming section. For the rest of this thesis, the use of *conjunctive MMPS functions* is used unless specifically clarified.

It is shown that any MMPS function from Definition 2-2.1 can be written into the conjunctive or disjunctive MMPS form:

Theorem 2-2.1. [7] Any MMPS function $f : \mathcal{R}^n \to \mathcal{R}$ can be rewritten into the conjunctive or disjunctive MMPS form.

Proof. [7, 29] Consider the two affine functions f_k and f_l, then the functions that result from applying the basic constructors of an MMPS function (a list with properties of the max, min, + and scaling operations in the MMPS framework are provided in Appendix A) are in min-max MMPS form. Then a recursive argument can be used that consists in showing that if the basic constructors of an MMPS function are applied to two (or more) MMPS functions in min-max MMPS form, then the result can again be transformed into min-max MMPS form.

Let two MMPS functions f and g be in min-max canonical form : $f = \min(\max(f_1, f_2) \max(f_3, f_4))$ and $g = \min(\max(g_1, g_2) \max(g_3, g_4))$. Now it can be shown that $\max(f, g)$, $\min(f, g)$, $f + g$ and βf with $\beta \in \mathbb{R}$ can again be written in min-max MMPS form:

$$\begin{aligned}
\max(f, g) &= \max[\min(\max(f_1, f_2), \max(f_3, f_4)), \min(\max(g_1, g_2), \max(g_3, g_4))] \\
&= \max[\max(\min(f_1, f_3), \min(f_1, f_4), \min(f_2, f_3), \min(f_2, f_4), \\
&\qquad \max(\min(g_1, g_3), \min(g_1, g_4), \min(g_2, g_3), \min(g_2, g_4)] \\
&= \max(\min(f_1, f_3), \min(f_1, f_4), \min(f_2, f_3), \min(f_2, f_4), \\
&\qquad \min(g_1, g_3), \min(g_1, g_4), \min(g_2, g_3), \min(g_2, g_4)) \\
&= \min(\max(f_1, f_1, f_2, f_2, g_1, g_1, g_2, g_2), \max(f_1, f_1, f_2, f_2, g_1, g_1, g_2, g_4), ... \\
&\qquad \max(f_3, f_4, f_3, f_4, g_3, g_4, g_3, g_4))
\end{aligned}$$

$$\begin{aligned}\min(f,g) &= \min[\min(\max(f_1,f_2),\max(f_3,f_4)),\min(\max(g_1,g_2),\max(g_3,g_4))]\\ &= \min(\max(f_1,f_2),\max(f_3,f_4),\max(g_1,g_2),\max(g_3,g_4))\end{aligned}$$

$$\begin{aligned}f+g &= \min(\max(f_1,f_2),\max(f_3,f_4)) + \min(\max(g_1,g_2),\max(g_3,g_4))\\ &= \min(\max(f_1,f_2)+\max(g_1,g_2)),\max(f_1,f_2)+\max(g_3,g_4),\\ &\qquad \max(f_3,f_4)+\max(g_1,g_2)),\max(f_3,f_4)+\max(g_3,g_4))\\ &= \min(\max(f_1+g_1,f_1+g_2,f_2+g_1,f_2+g_2),\max(f_1+g_3,f_1+g_4,f_2+g_3,f_2+g_4)\\ &\qquad \max(f_3+g_1,f_4+g_1,f_4+g_1,f_4+g_2),\max(f_3+g_3,f_3+g_4,f_4+g_3,f_4+g_4))\end{aligned}$$

$$\begin{aligned}\beta f &= \beta\min(\max(f_1,f_2),\max(f_3,f_4))\\ &= \min(\max(\beta f_1,\beta f_2),\max(\beta f_3,\beta f_4)) \quad \text{if} \beta \leq 0\\ &= -|\beta|\min(\max(f_1,f_2),\max(f_3,f_4))\\ &= -\min(\max(|\beta|f_1,|\beta|f_2),\max(|\beta|f_3,|\beta|f_4))\\ &= \max(-\max(|\beta|f_1,|\beta|f_2),-\max(|\beta|f_3,|\beta|f_4))\\ &= \max(\min(-|\beta|f_1,-|\beta|f_2),\min(-|\beta|f_3,-|\beta|f_4))\\ &= \max(\min(\beta f_1,\beta f_2),\min(\beta f_3,\beta f_4))\\ &= \min(\max(\beta f_1,\beta f_3),\max(\beta f_1,\beta f_4),\max(\beta f_2,\beta f_3),\max(\beta f_2,\beta f_4))\end{aligned}$$

Concluding that the conjunctive MMPS form is indeed a canonical form within the MMPS framework. □

Lattice Representations Another representation of conjunctive & disjunctive MMPS functions can be found in literature under the name *lattice representation* (of piecewise affine functions) [11, 26, 32]. Here conjunctive & disjunctive MMPS functions are represented using sets, rather than matrices:

$$f(x) = \min_{i=1,\dots,M}\{\max_{j\in I_i}\{\ell_j(x)\}\}, \quad \forall x \in f : \mathcal{R}^n \tag{2-8}$$

or $f = \max_{i=1,\dots,M_2}\{\min_{j\in\bar{I}_i}\{\ell_j\}\}$ in which ℓ_j is an affine function, M and M_2 are integers and I_i and $\bar{I}_i$ are index sets. The sets and parameters of the affine functions are usually stored in a structure matrix ψ and parameter matrix ϕ [31]. The lattice representation may then also be written as:

$$f(x|\psi,\phi) = \min_{i=1,\dots,M}\{\max_{\substack{j=1,\dots,M\\ \psi_{ij}=1}}\{l(x|\phi_j)\}\}, \quad \forall x \in \mathcal{R} \tag{2-9}$$

Where $\phi = [\phi_1,...,\phi_M]^{\mathrm{T}}$ is the $M \times (n+1)$ parameter matrix, and $\psi = [\psi_{ij}]$ a $M \times M$ zero-one matrix [31]. The parameters of the affine functions $l(x)$ are exactly the row vectors of ϕ. Hence the matrix ϕ is called the parameter matrix. The elements of the structure matrix ψ can be either a numerical one or zero:

$$\psi_{ij} = \begin{cases}1, & \text{if } j \in I_i\\ 0, & \text{otherwise}\end{cases} \tag{2-10}$$

Uniqueness of the conjunctive MMPS form It is important to note that the conjunctive form is not unique, i.e. there are different constructions of the matrices $\alpha_{(i,j)}$ and $\beta_{(i,j)}$, or the index sets I_i that have the same input-output behaviour. In the next section the topic of *irredundant form* will be discussed, where the function is described using the least amount of parameters.

Example 2. Consider a 1-dimensional disjunctive MMPS function given by:

$$f(x) = \max\Big(\min(0.5x + 0.5, -0.5x + 3), \min(2x - 1, 2, -2x + 9)\Big) \tag{2-11}$$

The α and β matrices from Definition 2-2.1 are:

$$\alpha_1 = \begin{pmatrix} 0.5 \\ \epsilon \\ \epsilon \\ \epsilon \\ -0.5 \end{pmatrix}, \quad \alpha_2 = \begin{pmatrix} \epsilon \\ 2 \\ 0 \\ -2 \\ \epsilon \end{pmatrix}, \beta_1 = \begin{pmatrix} 0.5 \\ \epsilon \\ \epsilon \\ \epsilon \\ 3 \end{pmatrix}, \quad \beta_2 = \begin{pmatrix} \epsilon \\ -1 \\ 2 \\ 9 \\ \epsilon \end{pmatrix}$$

The function can also be represented with a structure matrix ψ and parameter matrix ϕ according to the lattice representation:

$$\phi = \begin{pmatrix} 0.5 & 0.5 \\ 2 & -1 \\ 0 & 2 \\ -2 & 9 \\ -0.5 & 3 \end{pmatrix}, \quad \psi = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Each producing the same plot:

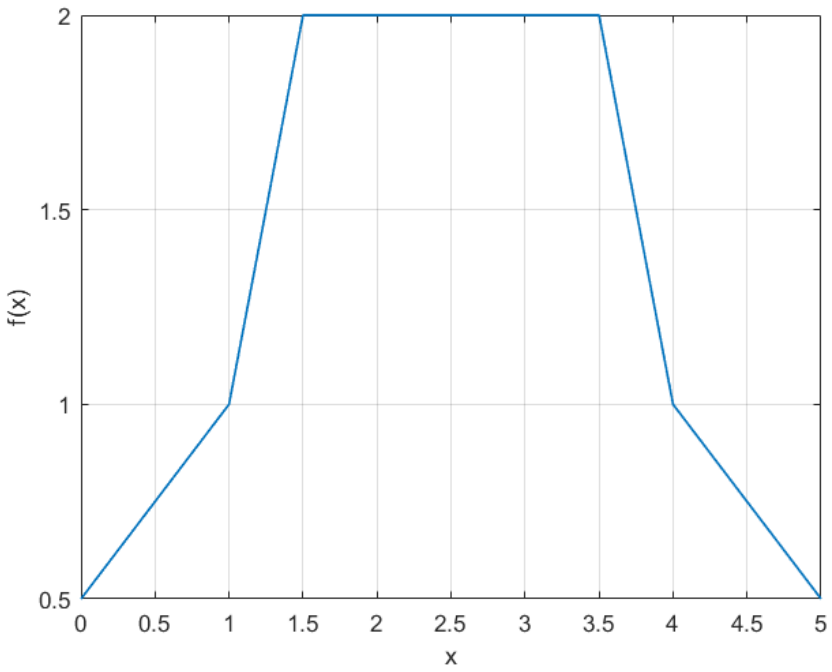


Figure 2-2: Function plot of Example 2

2-2-2 Kripfganz MMPS functions

Kripfganz functions are function that are composed of the difference of two convex functions, or likewise, two concave functions. Kripfganz functions first appeared in [19], where it was shown that any continuous PWA function can be written as the difference of two *convex* PWA functions:

Lemma 2-2.2. Every continuous PWA function $f : \Omega \to \mathcal{R}$ defined over a convex polyhedral partition of $\Omega \subseteq \mathcal{R}^n$ with full dimensional elements Ω_k can be written as the difference of two convex PWA functions $g(x)$ and $h(x)$ i.e.

$$f(x) = g(x) - h(x) \tag{2-12}$$

Here the functions $g(x)$ and $h(x)$ are again continuous PWA functions according to Definition 2-1.1. However because $g(x)$ and $h(x)$ are both convex functions, they can be rewritten into the MMPS-framework according to Definition 2-2.1 as two *maximization functions*:

Definition 2-2.3. (Kripfganz MMPS functions [19, 29]) A scalar-valued MMPS function $f : \mathcal{R}^n \to \mathcal{R}$ is called a Kripfganz MMPS functions if it can be written in the form:

$$f(x) = \max_{i=1,\dots,M}(\mu_{1,i}^T x + \nu_{1,i}) - \max_{j=1,\dots,K}(\mu_{2,i}^T x + \nu_{2,i}), \quad \forall x \in \mathcal{R}^n \tag{2-13}$$

Like conjunctive & disjunctive MMPS functions, the kripfganz MMPS function may also be written using index sets:

$$f = \max_{i \in I_j}\{\ell_j\} - \max_{i \in I_k}\{\ell_k\}, \quad \forall x \in \mathcal{R}^n \tag{2-14}$$

in which ℓ_i and ℓ_j are affine functions and I_j index sets.

Uniqueness of the kripfganz MMPS form Both the convex decomposition as well as the parameters of the two convex functions are not necessarily unique. The forthcoming section is dedicated to the uniqueness of the partition.

Example 3. Consider the following 1-dimensional kripfganz MMPS function with constrains $0 \leq x \leq 5$:

$$f(x) = \max(-1.5x + 2, 8, -1.5x + 9.5) - \max(2x - 1, 6, -2x + 9)$$

with the two convex parts $g(x) = \max(-1.5x + 2, 8, -1.5x + 9.5)$ and $h(x) = \max(2x - 1, 6, -2x + 9)$. This produces the following plot for $f(x)$, $g(x)$ and $-h(x)$:

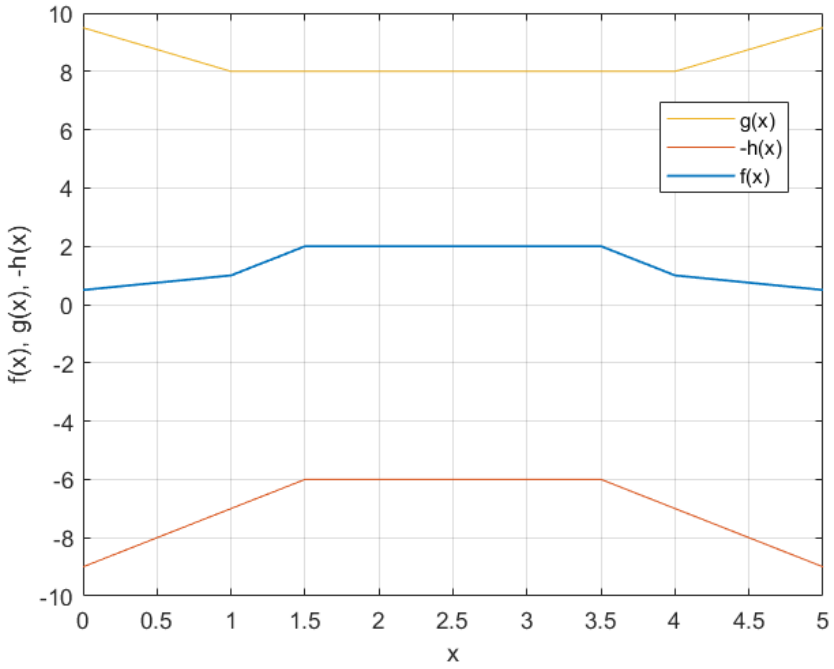


Figure 2-3: Function plot of Example 3

2-2-3 Equivalence MMPS functions & continuous PWA functions

It is demonstrated in [10, 21, 32] that any continuous PWA function can be expressed by a min-max or max-min composition of its affine components, i.e. the conjunctive MMPS form:

Theorem 2-2.3. [10, 21, 32] If f is a continuous piecewise affine function of the form given in Definition 2-1.1, then there exist index sets $I_1, ..., I_l \subseteq \{1, ..., N\}$ such that

$$f(x) = \min_{i=1,...,l} \max_{j \in I_i} (\alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)}), \quad \forall x \in \mathcal{R}^n \tag{2-15}$$

In [26] formal proofs are given demonstrating that any continuous PWA can be described by Equation 2-15, which are then called lattice PWA representations. From the definition of the canonical forms the following can be concluded:

Lemma 2-2.4. [29, 27] Any MMPS function is also a continuous PWA function

Well-defined When MMPS functions that are defined over a convex polyhedral $\Omega \subseteq \mathbb{R}^n$, then the relation is well-posed. MMPS function however can also be defined over $\mathbb{R}_\top$, $\mathbb{R}_\varepsilon$ and $\mathbb{R}_c$. In this case the Minkowski cone may be used to map the function to the real space, $\mathcal{R} \to \mathbb{R}$. In this thesis we consider any MMPS function defined over $\mathcal{R}^n$ as *well-defined* .

Example 4. (Example 1 continued) Let us consider again the previous examples. Be reminded that we had the following functions: The continuous PWA function in Example 1:

$$f_{\text{pwa}}(x) = \begin{cases} \ell_1(x) = 0.5x + 0.5 & x \in [0.0, 1.0] \\ \ell_2(x) = 2x - 1 & x \in [1.0, 1.5] \\ \ell_3(x) = 2 & x \in [1.5, 3.5] \\ \ell_4(x) = -2x + 9 & x \in [3.5, 4.0] \\ \ell_5(x) = -0.5x + 3 & x \in [4.0, 5.0] \end{cases}$$

The equivalent conjunctive MMPS function from Example 2:

$$f_{\text{cd}}(x) = \max\Big(\min(0.5x + 0.5, -0.5x + 3), \min(2x - 1, 2, -2x + 9)\Big)$$

And the equivalent kripfganz MMPS function from Example 3:

$$f_{\text{kg}}(x) = \max(-1.5x + 2, 8, -1.5x + 9.5) - \max(2x - 1, 6, -2x + 9)$$

By now it should have been noticed that each example produces the same output.

2-2-4 Applications of MMPS functions

The canonical MMPS representations are an extremely useful in various fields and problems such as explicit MPC, dynamic programming, modelling of DES, DC programming, neural networks, parametric linear programs [15, 29, 35, 7, 24] to just name a few. The MMPS framework can be used to model discrete phenomena such as concurrency, synchronisation and event triggered systems.

In many cases this is the case where the problem is described as a continuous PWA function. For example, the control law in explicit MPC is obtained by solving a finite-horizon open-loop optimal control problem at each sampling instant. At the next time instant, this is repeated where the horizon is shifted one step. This optimization relies on a model of the system (hence the name Model Predictive Control) and can take constraints on the input and output. A continuous PWA control law arises in the case these constraints are affine. As a result the control law can be taken offline and the online computation can be reduced to an online evaluation. MMPS canonical forms are an excellent tool to represent the control law as an analytical form, rather than an look-up table [31].

2-3 Summary

Continuous PWA function is a function that is defined by multiple affine segments, each with its own slope and intercept. The function can be thought of as being constructed by gluing together these linear segments at certain points. Generally Piecewise affine functions are a composition of affine functions defined over polyhedral regions. PWA functions are very intuitive and conceptually easy to understand. PWA functions can therefor be a powerful tool to approximate nonlinear functions. The key advantage of PWA functions is that they

are relatively simple to work with and can provide a good approximation of more complex nonlinear functions.

An MMPS expression uses a recursive grammar to construct functions composed of the operations maximization, minimization, scalar multiplication and addition. Two important canonical MMPS function are those that can be described as the minimization of multiple maximization terms, so called conjunctive MMPS functions, and those that can be described as the difference between two maximization (convex) parts. (Canonical) MMPS functions are not necessarily unique nor irredundant. The MMPS framework is an excellent tool to model hybrid systems and discrete event systems, or to construct efficient optimization problems.

Continuous PWA functions and MMPS functions are equivalent. Both canonical MMPS forms can be viewed as an analytical expression of continuous PWA functions. Solving a (canonical) MMPS function is computationally more efficient than that of a continuous PWA function.

Chapter 3

Transformations between MMPS functions and continuous PWA functions

In this chapter we look into transforming one MMPS form into another, as well as transforming continuous PWA functions into MMPS (canonical) functions and vice versa. Because each form has its own advantage, it is of great value to be able to transform one form into another.

There does not exist a general strategy in current literature to map one form into another. Moreover, not all transformation can be found in literature. The following transformations can be found:

- 3-1-1 General MMPS to conjunctive MMPS form
- 3-2-1 Continuous PWA to conjunctive MMPS form.
- 3-2-2 Continuous PWA to Convex Kripfganz MMPS form
- 3-1-2 Conjunctive MMPS form to Convex Kripfganz MMPS form
- 3-1-3 Conjunctive MMPS form to disjunctive MMPS form (and vice versa)
- 3-1-4 Convex Kripfganz MMPS form to Concave Kripfganz MMPS form (and vice versa)

Methods for mapping between continuous PWA and canonical MMPS functions rely on algorithms strategically partitioning the domain and collecting index sets, whereas the methods for mapping between canonical MMPS functions rely on algebraic properties of the max and min properties, and the reordering of terms. Each transformation will now be reviewed in the next sections.

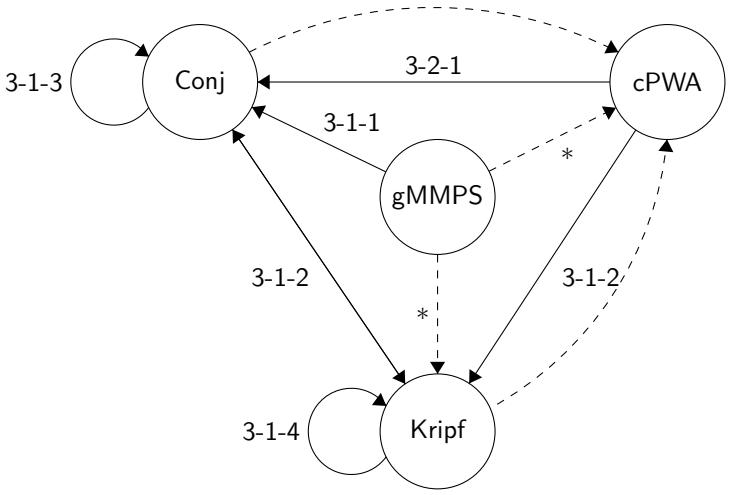


Figure 3-1: Connections between MMPS canonical forms. The numbers indicate existing techniques for mapping one form to another. Dashed lines indicate the lack of any methods that can may be found in current existing literature.

3-1 Transformations between MMPS (canonical) forms

3-1-1 General MMPS form to conjunctive MMPS form

We first start with the idea of having a MMPS function in *general form*, and finding a canonical MMPS function. With 'general form', we talk about any MMPS function (following the recursive grammar from Definition 2-2.1) that is not in any of the canonical forms.

The algebraic rules for the MMPS framework (i.e. the operations maximization, minimization, addition and scaling) can be used to transform each part of the MMPS framework into the conjunctive function. Expressing any MMPS function into the conjunctive form follows from the set of rules from the work of [7]. These rules are constructive and following these rules transforms a general MMPS function into a conjunctive (and likewise disjunctive) MMPS function algorithmic.

Example 5. Consider the following MMPS function in *general form*:

$$f(x) = \min(\ell_1, \ell_2) - 2\max\big(\min(\ell_3, \ell_4), \ell_5\big)$$

first we rewrite the left and right term in conjunctive form:

$$\min(\ell_1, \ell_2) = \min\big(\max(\ell_1), \max(\ell_2)\big)$$

and the right term:

$$\begin{aligned} -2\max\big(\min(\ell_3, \ell_4), \ell_5\big) &= -2\min\big(\max(\ell_3, \ell_5), \max(\ell_4, \ell_5)\big) \\ &= \min\big(\max(-2\ell_3, -2\ell_4), \max(-2\ell_3, -2\ell_5), \\ &\qquad \max(-2\ell_4, -2\ell_5), \max(-2\ell_5)\big) \end{aligned}$$

Now we combine the left and rigt side to a single conjunctive expression:

$$\begin{aligned}
f(x) = &\min\big(\max(\ell_1), \max(\ell_2))+ \\
&\quad \min\big(\max(-2\ell_3, -2\ell_4), \max(-2\ell_3, -2\ell_5), \\
&\qquad \max(-2\ell_4, -2\ell_5), \max(-2\ell_5)\big) \\
&= \min(\\
&\qquad \max(\ell_1 - 2\ell_3, \ell_1 - 2\ell_4), \\
&\qquad \max(\ell_1 - 2\ell_3, \ell_1 - 2\ell_5), \\
&\qquad \max(\ell_1 - 2\ell_4, \ell_1 - 2\ell_5), \\
&\qquad \max(\ell_1 - 2\ell_5), \\
&\qquad \max(\ell_2 - 2\ell_3, \ell_2 - 2\ell_4), \\
&\qquad \max(\ell_2 - 2\ell_3, \ell_2 - 2\ell_5), \\
&\qquad \max(\ell_2 - 2\ell_4, \ell_2 - 2\ell_5), \\
&\qquad \max(\ell_2 - 2\ell_5) \\
&\quad)
\end{aligned}$$

3-1-2 Conjunctive MMPS form to (Convex) Kripfganz MMPS form

In [30] a method for transforming a conjunctive MMPS form into a convex Kripfganz MMPS form was given. The general idea is to construct the left side convex function $g(x)$ by taking the sum of all maximization terms using combinatorics, and then to build $h(x)$ by solving $h(x) = g(x) - f(x)$ and again using a combinatorics approach:

proposition 3-1.1. [30, 29] The function $f(x) = \min_{i=1,\dots,K} \max_{j \in n_i} \left(\alpha_{(i,j)}^{\mathrm{T}} x + \beta_{(i,j)}\right)$ where $\alpha_{(i,j)}^{\mathrm{T}} x + \beta_{(i,j)}$ is affine in x, can be written as a difference of two convex functions, i.e. $f(x) = g(x) - h(x)$.

Proof. [29, 30] First define:

$$g_i = \max_{j \in (1,\dots,n_i)} \left(\alpha_{(i,j)}^{\mathrm{T}} x + \beta_{(i,j)}\right) \tag{3-1}$$

Then the following can be derived:

$$\begin{aligned}
\min_{i \in (1,\dots,K)} g_i - \sum_{t \in (1,\dots,K)} g_t &= \min_{i \in (1,\dots,K)} \Big(g_i - \sum_{t \in (1,\dots,K)} g_t\Big) && (3\text{-}2)\\
&= \min_{i \in (1,\dots,K)} \Big(- \sum_{t \in (1,\dots,K)/i} g_t\Big) && (3\text{-}3)\\
&= - \max_{i \in (1,\dots,K)} \Big(\sum_{t \in (1,\dots,K)/i} g_t\Big) && (3\text{-}4)\\
& && (3\text{-}5)
\end{aligned}$$

thus

$$f(x) = \min_{1=1,\dots,K} g_i = \sum_{t \in (1,\dots,K)} g_t - \max_{i \in (1,\dots,K)} \Big(\sum_{t \in (1,\dots,K)/i} g_t\Big) \tag{3-6}$$

Then it follows $f(x) = g(x) - h(x)$ where $g(x)$ and $h(x)$ are defined as:

$$\begin{aligned} g(x) &= \sum_{t\in(1,\dots,K)} g_t \\ &= \sum_{t\in(1,\dots,K)} \max_{j\in n_i} \left(\alpha_{(t,j)}^{\mathrm{T}} x + \beta_{(t,j)}\right) \\ h(x) &= \max_{i\in(1,\dots,K)} \Big(\sum_{t\in(1,\dots,K)/i} g_t \Big) \\ &= \max_{i\in(1,\dots,K)} \Big(\sum_{t\in(1,\dots,K)/i} (\alpha_{(t,j)}^{\mathrm{T}} x + \beta_{(t,j)}) \Big) \end{aligned}$$

□

Example 6. Let us consider the following conjunctive MMPS function, with affine functions ℓ_i, $i = 1, ..., 6$, expressed in sets:

$$f(x) = \min\big(\max(\ell_1, \ell_5), \max(\ell_2, \ell_3, \ell_4), \max(\ell_6)\big) \tag{3-7}$$

We first find the convex function $g(x)$ by summing over the maximization terms:

$$\begin{aligned} g(x) &= \sum \max(\ell_1, \ell_5), \max(\ell_2, \ell_3, \ell_4), \max(\ell_6) \\ &= \max(\ell_1, \ell_5) + \max(\ell_2, \ell_3, \ell_4) + \max(\ell_6) \end{aligned}$$

Note that we can further expand the last line using combinatorics, but we will save that for the last step to maintain a better overview and understanding of the algorithmic process.

Now let us rewrite $f(x)$ by changing the outer minimization operation as a maximization:

$$\begin{aligned} f(x) &= \min\big(\max(\ell_1, \ell_5), \max(\ell_2, \ell_3, \ell_4), \max(\ell_6)\big) \\ &= -\max\big(-\max(\ell_1, \ell_5), -\max(\ell_2, \ell_3, \ell_4), -\max(\ell_6)\big) \end{aligned}$$

Then we find the convex function $h(x)$, by solving $h(x) = g(x) - f(x)$, for the $g(x)$ and rewritten $f(x)$:

$$\begin{aligned} h(x) &= g(x) - f(x) \\ &= \max(\ell_1, \ell_5) + \max(\ell_2, \ell_3, \ell_4) + \max(\ell_6) \\ &\quad + \max\big(-\max(\ell_1, \ell_5), -\max(\ell_2, \ell_3, \ell_4), -\max(\ell_6)\big) \\ &= \max\big(\\ &\quad \max(\ell_1, \ell_5) + \max(\ell_2, \ell_3, \ell_4) + \max(\ell_6) - \max(\ell_1, \ell_5), \\ &\quad \max(\ell_1, \ell_5) + \max(\ell_2, \ell_3, \ell_4) + \max(\ell_6) - \max(\ell_2, \ell_3, \ell_4), \\ &\quad \max(\ell_1, \ell_5) + \max(\ell_2, \ell_3, \ell_4) + \max(\ell_6) - \max(\ell_6) \\ &\big) \\ &= \max\big(\\ &\quad \max(\ell_2, \ell_3, \ell_4) + \max(\ell_6), \\ &\quad \max(\ell_1, \ell_5) + \max(\ell_6), \\ &\quad \max(\ell_1, \ell_5) + \max(\ell_2, \ell_3, \ell_4) \\ &\big) \end{aligned}$$

Finally we can expand the nested maximization terms to get the full expression. :

$$\begin{aligned} h(x) = \max\big(& \\ & \ell_2+\ell_6, \ell_3+\ell_6, \ell_4+\ell_6, \\ & \ell_1+\ell_6, \ell_5+\ell_6 \\ & \ell_1+\ell_2, \ell_1+\ell_3, \ell_1+\ell_4, \ell_5+\ell_2, \ell_5+\ell_3, \ell_5+\ell_4 \\ \big) & \end{aligned}$$

3-1-3 Conjunctive MMPS form to disjunctive MMPS form (and vice versa)

Mapping between the conjunctive and disjunctive form is easily verifiable given the properties of the max and min operators [7]:

$$\begin{aligned} \min(\max(\alpha,\beta),\max(\gamma,\delta)) &= \max(\min(\alpha,\gamma),\min(\alpha,\delta),\min(\beta,\gamma),\min(\beta,\delta)) \\ \max(\min(\alpha,\beta),\min(\gamma,\delta)) &= \min(\max(\alpha,\gamma),\max(\alpha,\delta),\max(\beta,\gamma),\max(\beta,\delta)) \end{aligned}$$

Example 7. Let us again consider the conjunctive MMPS function, with affine functions ℓ_i, $i = 1, ..., 6$, expressed in sets:

$$f(x) = \min\big(\max(\ell_1,\ell_5),\max(\ell_2,\ell_3,\ell_4),\max(\ell_6)\big) \tag{3-8}$$

Then using the approach from above and combinatorics we get the following disjunctive MMPS function:

$$\begin{aligned} f(x) = \max\big(& \\ & \min(\ell_1,\ell_2,\ell_6),\min(\ell_1,\ell_3,\ell_6),\min(\ell_1,\ell_4,\ell_6), \\ & \min(\ell_5,\ell_2,\ell_6),\min(\ell_5,\ell_3,\ell_6),\min(\ell_5,\ell_4,\ell_6) \\ \big) & \end{aligned}$$

3-1-4 Convex Kripfganz MMPS form to Concave Kripfganz MMPS form (and vice versa)

The Kripfganz MMPS form is generally noted as the difference between two convex functions, i.e. using maximization. Through the relation between the operators maximization and minimization, the (convex) Kripfganz MMPS form may also be written as the difference between two concave functions, or using minimization[7]:

$$\max(\alpha,\beta) = -\min(-\alpha,-\beta) \tag{3-9}$$
$$\min(\alpha,\beta) = -\max(-\alpha,-\beta) \tag{3-10}$$

Example 8. Consider the following kripfganz MMPS function:

$$\begin{aligned} f(x) &= \max(\ell_1,\ell_2,\ell_3) - \max(\ell_4,\ell_5,\ell_6) \\ &= \min(-\ell_4,-\ell_5,-\ell_6) - \min(-\ell_1,-\ell_2,-\ell_3) \end{aligned}$$

3-2 Transformation between canonical MMPS functions and continuous PWA functions

3-2-1 Continuous PWA to Conjunctive MMPS form

There are several methods to create the conjunctive MMPS function from a continuous PWA function, but these methods use the principle of finding sets where specific affine functions are larger or equal to the active function. In other words the structure matrix ψ can be constructing in the following way:

$$\phi_{ij} = \begin{cases} 1 & \text{if } \ell_i(x) \geq \ell_j(x) \\ 0 & \text{else} \end{cases}$$

with $i \geq 1$, the number of regions, and $1 \leq j \leq N_{\text{aff}}$, the number of unique affine functions. However there are different choices over which regions should be looped. In [31, 26] the strategy is to loop over the original partition of the continuous PWA function. In [35] and in [34] the base region partition and convex projection region partition is suggested which in turn create both different sets, but also give different results when redundancy removal techniques are deployed. We discuss each method briefly.

Standard method In [26, 31] the following method is used to create the conjunctive MMPS form:

Lemma 3-2.1. Assume that Ω_i, Ω_j are two n-dimensional convex polytopes, where $\ell_i(x), \ell_j(x)$ are their local affine functions with $i, j \in \{1, ..., N_{\text{aff}}\}$. Then the structure matrix $\psi = [\psi_{ij}]^{N_{\text{aff}} \times N_{\text{aff}}}$ can be calculated as follows:

$$\phi_{ij} = \begin{cases} 1 & \text{if } \ell_i(v_k) \geq \ell_j(v_k), \quad 1 \leq k \leq K_i \\ 0 & \text{if } \ell_i(v_k) < \ell_j(v_k), \quad k\{1, ..., K_i\} \end{cases} \tag{3-11}$$

where v_k are the vertices of Ω_i with $1 \leq k \leq K_i$ and $K_i \in \mathbb{Z}^+$ is the number of vertices of Ω_i

Proof. [31] Since Ω_i is an n-dimensional polytope, it can be described by its vertices $v_i, ..., v_{K_i}$

$$\Omega_i = \left\{ x \in \mathcal{R}^n | x = \sum_{k=1}^{K_i} \lambda_k v_k, \quad 0 \leq \lambda_k \leq 1, \quad \sum_{k=1}^{K_i} \lambda_k = 1 \right\} \tag{3-12}$$

Then for any $x \in \Omega_i$ we have

$$\ell_i(x) = \ell_i \left(\sum_{k=1}^{K_i} \lambda_k v_k \right) = \sum_{k=1}^{K_i} \lambda_k \ell_i(v_k)$$
$$\ell_j(x) = \ell_j \left(\sum_{k=1}^{K_i} \lambda_k v_k \right) = \sum_{k=1}^{K_i} \lambda_k \ell_j(v_k)$$

if $\ell_i(v_k) \geq \ell_j(v_k), \quad \forall \mathbf{1} \leq k \leq K_i$, then $\ell_i(x) \geq \ell_j(x)$ holds for all $x \in \Omega_i$. It follows that $\psi_{ij} = 1$.

Similarly, if there exists any $k \in \{1, ..., K_i\}$ such that $\ell_i(v_k) < \ell_j(v_k)$, then $\ell_i(x)$ and $\ell_j(x)$ will intersect together with an $(n-1)-$dimensional hyperplane as the common boundary. This implies that $\psi_{ij} = 0$.

Using the same procedure stated above, all the elements in the structure matrix ψ can be calculated, and this completes the proof. □

Base Region Realisation In [19, 35, 26] a method for mapping the continuous PWA function to the conjunctive (and disjunctive) form was given. The method is based on the fact that for every continuous PWA its *full lattice representation* can be generated by partitioning the domain into subsequent sub-regions in which the active affine function is a solution of the minimization of a set of affine functions as described in [26]. The partition is performed such that in each new region the local function does not intersect with any of the functions. Formally the partition is defined as:

Considering each subregion $\Omega_i (i = 1, ..., \hat{N})$, then Ω_i can be further partitioned into base regions $\mathbb{D}_{i,t}$ with $t = 1, ..., m_i$, to make sure that no other affine function intersects with $\ell_{loc(i)}(x) = \alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)}$ at some point in the interior of $\mathbb{D}_{i,t}$, i.e.

$$(x | \ell_j(x) = \ell_{loc(i)}(x), \quad j \neq \mathrm{loc}(i)) \cap \mathrm{int}(\mathbb{D}_{i,t}) = \emptyset \tag{3-13}$$

This partitioning is defined according to the lemma [35]:

Theorem 3-2.2. (Base Region) [35] For any $i \in (1, ..., \bar{N})$, there is a partition of the subregion Ω_i, where

$$\Omega_i = \cup_{t=1}^{m_i} \mathbb{D}_{i,t} \tag{3-14}$$

such that the following holds:

1. The set $\mathrm{int}(\mathbb{D}_{i,t})$ is nonempty
2. For each $\mathbb{D}_{i,t}$ we have

$$\begin{aligned} I_{\geq,i,t} \cup I_{\leq,i,t} &= (1, .., M) \\ \text{with, } I_{\geq,i,t} &= (j | \ell_j(x) \geq \ell_{loc(i)}(x), \forall x \in \mathbb{D}_{i,t}) \\ \text{and, } I_{\leq,i,t} &= (j | \ell_j(x) \leq \ell_{loc(i)}(x), \forall x \in \mathbb{D}_{i,t}) \end{aligned}$$

3. For all $i, j \in (1, ..., \bar{N}), \bar{t} \in (1, ..., m_i), \hat{t} \in (1, ..., m_j, \bar{t} \neq \hat{t})$ or $i \neq j$

$$\mathrm{int}(\mathbb{D}_{i,\bar{t}}) \cap \mathrm{int}(\mathbb{D}_{i,\hat{t}}) = \emptyset \tag{3-15}$$

This partitioning creates the base regions $\mathbb{D}_{1,1}, ..., \mathbb{D}_{1,m_1}, ..., \mathbb{D}_{\hat{N},1}, ..., \mathbb{D}_{\hat{N},m_{\hat{N}}}$. Finally these are renumbered as: $\mathbb{D}_1, ..., \mathbb{D}_N$ where $N = m_1 + ... + m_{\hat{N}}$. Now three types of index sets are introduced. Before this a distinction is made between the local function *before* and *after* the secondary partition: denote the active affine function in base region $\mathbb{D}_i$ as $\ell_{\mathrm{act}(i)}$ given by:

$$\ell_{\mathrm{act}(i)} = \ell_{\mathrm{loc}(i)}, \quad \text{if } \mathbb{D}_i \subseteq \Omega_j \tag{3-16}$$

Then define the following two index sets:

$$I_{\geq,i} = (j|\ell_j(x) \geq \ell_{\text{act}(i)}, \quad \forall x \in \mathbb{D}_i)$$
$$I_{\leq,i} = (j|\ell_j(x) \leq \ell_{\text{act}(i)}, \quad \forall x \in \mathbb{D}_i)$$

And let the third index set $\mathcal{A}(\ell_i)$ be such that for each index $k \in \mathcal{A}(\ell_i)$, ℓ_i is the active affine function in $\mathcal{D}_k$, i.e. $f(x) = \ell_i(x), \ \forall x \in \mathbb{D}_k$. In the base region $\mathbb{D}_i$ for an affine function ℓ_j with $j \neq \text{act}(i)$, either $j \in I_{\geq,i}$ or $j \in I_{\leq,i}$. Then we have:

$$\ell_j(x) > \ell_{\text{act}(i)}(x), \quad \forall x \in \text{int}(\mathbb{D}_i), \quad \forall j \in I_{\geq,i} \tag{3-17}$$

and

$$\ell_j(x) < \ell_{\text{act}(i)}(x), \quad \forall x \in \text{int}(\mathbb{D}_i), \quad \forall j \in I_{\leq,i} \tag{3-18}$$

The following conclusion can now be made for which the proof can be found in [35]:

proposition 3-2.3. Let f be a 1-dimensional continuous PWA function as defined in definition 2-1.1, i.e. $n = 1$, then $\forall i, k \in (1, ..., N)$ we have

$$\min_{j \in I_{\geq,k}} (\ell_j(x)) \leq \min_{j \in I_{\geq,i}} (\ell_j(x)), \quad \forall x \in \mathbb{D}_i \tag{3-19}$$

Based on the proposition the *full lattice representation* is given[35]:

Theorem 3-2.4. Full lattice representation. Let f be a continuous PWA function as defined in definition 2-1.1. Then f can be represented as:

$$f(x) = f_{\text{BR}}(x) = \min_{i=1,...,N} (\max_{j \in I_{\geq,i}} (\ell_j(x))), \quad \forall x \in \mathbb{D} \tag{3-20}$$

Convex Projection Region Realisation This realisation is described and proven in [34]. Where base region realisation can be computationally demanding, the convex projection region realisation is relatively computationally efficient. Generally the number of convex projection regions is much smaller than that of base regions.

The algorithm essentially projects the continuous PWA function onto a plane. Secondly it checks every region for convexity. If a projected region is convex, it will be considered a convex projection region. If it is not convex, the region is partitioned by extending the adjacent affine element. This partitions the non convex projection region in multiple regions that are convex.

Below are the conditions and definitions reviewed as detailed in [34], however its proof is omitted.

Definition 3-2.1. [34] (Convex projection region): Assume $\Omega = \bigcup_{i=1}^{\bar{N}} \Omega_i$, the regions Ω_i are convex projection regions if,

1. Each Ω_i is a convex polyhedra
2. The boundaries of each Ω_i are $(n-1)$-dimensional hyperplanes

3. Each pair of $\text{int}(\Omega_i)$ and $\text{int}(\Omega_j)$ are disjoint, i.e.

$$\text{int}(\Omega_i) \cap \text{int}(\Omega_j) = \emptyset, \quad \forall i, j \in (1, ..., \bar{N}), i \neq j \tag{3-21}$$

   where $\text{int}(\Omega_i)$ denotes the interior of Ω_i

4. For each Ω_i, f equals a local affine function $\ell_{\text{loc}(i)}(x)$

It is noted that convex projection regions can be seen as convex polyhedral regions, as polyhedral regions in Definition 2-1.1 may be nonconvex.

Representation in $\mathbb{R}$

Lemma 3-2.5. For continuous PWA functions $f(x)$ defined in Definition 2-1.1, if the domain $\Omega \subset \mathbb{R}$, we have:

$$\min_{j \in J_{\geq,k}} (\ell_j(x)) \leq f(x), \quad \forall x \in \Omega \tag{3-22}$$

for any $k \in (1, ..., \bar{N})$, in which the index set $J_{\geq,k}$ is defined as:

$$J_{\geq,k} = (j \in (1, ..., M) | \ell_j(x) \geq \ell_{\text{loc}(k)}(x), \forall x \in \Omega_k) \tag{3-23}$$

Representation in $\mathbb{R}^n$

Theorem 3-2.6. For continuous PWA functions $f(x)$ defined in Definition 2-1.1, the following holds:

$$f(x) = f_{\text{CPR}}(x) = \max_{k=1,...,\bar{N}} \min_{j \in J_{\geq,k}} (\ell_j(x)), \quad \forall x \in \Omega \tag{3-24}$$

in which the index set $J_{\geq,k}$ is defined as previously. And the function $f_{CPR}(x)$ is called the CPR lattice representation of $f(x)$.

Note that through duality the same holds for the conjunctive case. For a full proof the reader is referred to [34]. Compared with base regions, convex projection regions satisfy one less condition. Hence a base region is also a convex region, but a convex projection region may not be a base region.

3-2-2 Continuous PWA to Kripfganz MMPS form

The problem of finding a Kripfganz MMPS form given a continuous PWA function can be found in literature as *convex decomposition* of PWA functions. There are two methods available from existing literature:

1. Decomposition via folds

2. Optimization-based decomposition

Decomposition via convex folds [19] The decomposition in [19] is based on the collection of all convex folds of $f(x)$ and to use them in such a way to construct $g(x)$. Let

$$\mathcal{I} := ((i,j) \in \mathbb{N} \times \mathbb{N} | \dim(\Omega_i \cap \Omega_j) = n-1, i < j) \tag{3-25}$$

be the set if tuples (i,j) indicating that the regions Ω_i and Ω_j are neighbors, i.e. they intersect along one hyperplane. Further, let:

$$\mathcal{V} := ((i,j) \in \mathcal{I} | \alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)} > \alpha_{(j)}^{\mathrm{T}} x + \beta_{(j)}, x \in \Omega_i \setminus \Omega_j) \tag{3-26}$$

denote the subset of $\mathcal{I}$ that collects facets on which f features a convex fold. Then:

$$g(x) := \sum_{(i,j) \in \mathcal{V}} \max(\alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)}, \alpha_{(j)}^{\mathrm{T}} x + \beta_{(j)}) \tag{3-27}$$

is obviously a convex function since the maximum of affine functions is convex and since sums preserve convexity. Note that every summand $\max(\alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)}, \alpha_{(j)}^{\mathrm{T}} x + \beta_{(j)})$ refers to a convex PWA function with two segments implicitly defined on the two halfspaces $\alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)} \geq \alpha_{(j)}^{\mathrm{T}} x + \beta_{(j)}$ and $\alpha_{(i)}^{\mathrm{T}} x + \beta_{(i)} \leq \alpha_{(j)}^{\mathrm{T}} x + \beta_{(j)}$ respectively. Interestingly, the function:

$$h(x) := g(x) - f(x) \tag{3-28}$$

is convex likewise [19], which completes the convex decomposition. In the last step, when computing $h(x)$ the number of partitions of the domain space may increase [24]. This may be seen as a disadvantage as it increases the storage requirements.

Optimization-based decomposition [15] In [15] a method for convex decomposition based on an optimization approach is presented. A benefit compared to the decomposition through convex folds is the fact this method preserves the original partition of the domain space, i.e. the functions f, g and h will all be affine on each polyhedron Ω_i. Moreover, the user has control over certain features of g and h by means of a cost-function. Now let the corresponding affine segments $g(x) = k_{(i)}^{\mathrm{T}} x + c_{(i)}$ and $h(x) = \ell_{(i)}^{\mathrm{T}} x + d_{(i)}$, then the decomposition requires:

$$\begin{aligned} \alpha_{(i)} &= k_i - \ell_i \\ \beta_{(i)} &= c_i - d_i \end{aligned} \tag{3-29}$$

for all $i \in (1, ..., \hat{N})$. Enforcing convexity of g and h, for every $(i,j) \in \mathcal{I}$ consider the inequality constraints

$$\begin{aligned} k_{(i)}^{\mathrm{T}} x + c_{(i)} &\geq k_{(j)}^{\mathrm{T}} x + c_{(j)} \\ \ell_{(i)}^{\mathrm{T}} x + d_{(i)} &\geq \ell_{(j)}^{\mathrm{T}} x + d_{(j)} \end{aligned}$$

$\forall x \in \Omega_i$ and

$$\begin{aligned} k_{(i)}^{\mathrm{T}} x + c_{(i)} &\leq k_{(j)}^{\mathrm{T}} x + c_{(j)} \\ \ell_{(i)}^{\mathrm{T}} x + d_{(i)} &\leq \ell_{(j)}^{\mathrm{T}} x + d_{(j)} \end{aligned}$$

$\forall x \in \Omega_j$. Then assuming half-space representations of the subsets $\Omega_{(i)}$ are at hand, i.e., $\Omega_{(i)} = (x \in \mathbb{R}^n | V_i x \leq w_i)$, the inequality constraints can be efficiently verified using Farka's

lemma. The inequality conditions are satisfied if and only if there exist (Lagrange multipliers) λ_{ij}, μ_{ij}, λ_{ji} and μ_{ji} of appropriate dimensions such that

$$\begin{aligned} 0 \leq \lambda_{ij}, \quad & V_i^{\mathrm{T}}\lambda_{ij} = (k_j - k_i)^{\mathrm{T}}, \quad w_i^{\mathrm{T}}\lambda_{ij} \leq c_i - c_j \\ 0 \leq \mu_{ij}, \quad & V_i^{\mathrm{T}}\mu_{ij} = (\ell_j - \ell_i)^{\mathrm{T}}, \quad w_i^{\mathrm{T}}\mu_{ij} \leq d_i - d_j \\ 0 \leq \lambda_{ji}, \quad & V_j^{\mathrm{T}}\lambda_{ji} = (k_i - k_j)^{\mathrm{T}}, \quad w_j^{\mathrm{T}}\lambda_{ji} \leq c_j - c_i \\ 0 \leq \mu_{ji}, \quad & V_j^{\mathrm{T}}\mu_{ji} = (\ell_i - \ell_j)^{\mathrm{T}}, \quad w_j^{\mathrm{T}}\mu_{ji} \leq d_j - d_i \end{aligned} \tag{3-30}$$

Then any feasible solution to 3-29 and 3-30 provides a valid decomposition of f into two convex PWA functions. Additionally the feasibility problem can be extended by a user-defined cost function or additional constraints in order to promote certain features of g and h. A major drawback however is the requirement of regulating the partition of (Ω_i) (see [15]), which is often not fulfilled even for simple partitions [24].

3-3 Minimal representations and uniqueness of MMPS canonical forms & continuous PWA functions

A MMPS canonical form, both conjunctive and kripfganz canonical forms, is not unique and redundant parameters may exist in the expression. For continuous PWA functions the domain over which they are defined may be partitioned into different regions, not necessarily the regions over which each affine function is active.

3-3-1 Conjunctive MMPS canonical functions

Conjunctive MMPS canonical function may have redundant parameters. Redundant parameters such as recurring sets are trivially redundant, however there are also less obvious redundant parameters.

In [31] However, the simplification lemmas have limitations and the result cannot guaranteed to be irredundant. In [33] an alternative method was proposed based on *base regions*, including necessary and sufficient conditions for irredundancy and the algorithm for obtaining an irredundant lattice PWA representation. In [34] a method based on *convex projection regions* was proposed to achieve a better performing algorithm. Interestingly, these algorithms produce different minimal representations. Irredudant representations are therefor not unique.

3-3-2 Kripfganz (MMPS canonical) functions

Kripfganz functions in MMPS form (two maximization functions) can have redundant parameters as recurring parameters and parameters that are never active. Recurring parameters are easily identified and removed. When the kripfganz function is in MMPS form and thus there is to direct region information, the (always) inactive functions are not directly identified.

Furthermore the decomposition itself is not unique. Both decomposition methods discussed in the previous section produce different results. Existing literature currently does not provide the sufficient conditions to show the minimal regions both functions produce.

3-3-3 Continuous PWA functions

The domain over which the Continuous PWA function is defined can be partitioned into different regions, namely regions, base regions. The minimum number of regions can be found by creating pairs of adjacent regions, checking whether both regions share the same active function and join the regions if this is the case.

The regions themselves can be represented as hyperplanes or vertices, or H-representation and v-representation respectively. More on the description of regions and polyhedral theory can be found in Appendix B. The number of hyperplanes or vertices used to describe a region is not necessarily minimal. (Non trivial) redundant hyperplanes may be found by solving linear program [16].

3-4 Summary

Continuous PWA functions and general-, conjunctive- and Kripfganz MMPS functions can be written into one another.

Continuous PWA functions can be decomposed into two convex functions to create kripfganz MMPS functions. The first convex function can be made by finding all convex folds over the whole domain and summing them. The second convex function can then be found by subtracting the original function from the first convex function. Alternatively one can take an optimization approach such that one has more control of the parameters of the convex functions. However the latter produces many more terms.

Continuous PWA functions can also be transformed into conjunctive MMPS functions by finding the correct sets. These sets are generally constructed by looping over all regions and finding those functions that are larger or equal to the affine function. However, depending on the regions one can get different results. By using different domain partitions and techniques to remove redundancy the resulting conjunctive MMPS function can produce different results.

Transformations between MMPS classes rely mostly on using MMPS algebraic rules. This way we can transform conjunctive MMPS functions into kripfganz functions and vice versa, we can transform general MMPS functions into conjunctive MMPS functions. The canonical duals are also rewritten into each other using the algebraic rules, i.e. the conjunctive and disjuctive MMPS forms and convex and concave Kripfganz MMPS forms. The biggest hurdle using these methods are the exploding terms as a result of the combinatorics used in the transformations.

Each transformation uses their own specific algorithm, the MMPS transformations produce exploding terms and transformation into continuous PWA form are not readily available.

Chapter 4

Finest Base Regions

In this chapter we presents a method to partition the domain space of any kind of Max-Min-Plus-Scaling (MMPS) function or continuous Piecewise Affine (PWA) function. The resulting partitioning can then be used as a tool to (re-)construct any type of canonical MMPS function or continuous PWA function. The new partitioning creates regions which are then called Finest Base Region (FBR).

The new partition adds new content to the current MMPS literature, namely:

- The FBR partition forms the basis to construct continuous PWA functions from any MMPS functions
- The FBR partition forms the basis for universal methods to construct one canonical form into another
- The FBR partition is a useful tool to study MMPS functions in better detail.

The partitioning method is based on hyperplane arrangement theory, and uses only the parameter matrix of the original function (or list of unique affine functions) to construct its regions. It will be shown that we can find this parameter matrix for any MMPS function using basic algebraic rules. The key idea is that this partition can be made with only the information of the unique affine functions: as long we can acquire all (unique) affine functions the FBR partition can be made. The procedure can be seen in Figure 4-1.

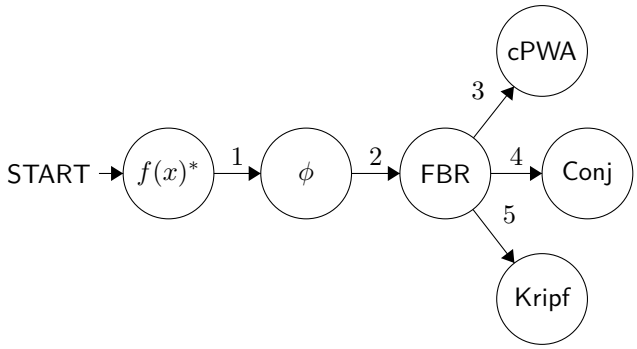


Figure 4-1: The procedure for using the FBR partition to create either a continuous PWA function or a conjunctive- or Kripfganz MMPS function from either a continuous PWA function or a general-, conjunctive-, kripfganz MMPS function. We start with a function $f(x)^*$ which can be any of the four form. At step 1, we collect all the affine functions inside the parameter matrix ϕ. At step 2, the FBR partition is made, creating polyhedral regions. From there we can take step 3, which finds the active function per region to create a continuous PWA function. Instead step 4 involves around finding the correct sets for the conjunctive MMPS function. Step 5 finds a convex decomposition given the parameter matrix and FBR regions.

The FBR partition allows us to construct continuous PWA functions from MMPS functions directly, filling in the gaps from Figure 3-1, and updating the techniques to Figure 4-2. The FBR partition is can be performed using any canonical MMPS function and thus may be seen as a 'universal' tool for the realisation of any form.

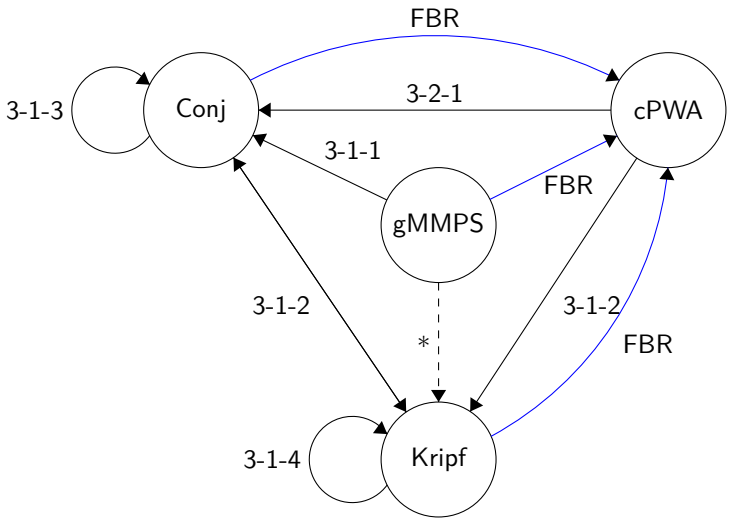


Figure 4-2: Connections between MMPS canonical forms. The numbers indicate existing techniques for mapping one form to another. Dashed lines indicate the lack of any methods that can may be found in current existing literature.

In the first part of this chapter an example of the FBR partition and Ordered Lattice (OL) representation is given to give a first impression and motivation for both proposals. After

the example some prerequisites about polyhedral theory are presented to accompany the algorithms later on.

In the second and third part of this chapter the FBR partitioning and OL realisation is discussed in further detail. This includes the reasoning behind the partitioning, what a FBR region is, how a FBR partitioning can be realized and algorithms including a complexity analysis of the partitioning. The ordered lattice representation is also further discussed in detail, including its realisation and how one can realize any canonical MMPS form from the OL representation.

4-1 Definition

FBR regions can be thought of as the image of the projection of all intersections between the unique affine functions. Consider a function $f(x) : \mathcal{R}^n \to \mathcal{R}$. Let $\ell_i(x)$ and $\ell_j(x)$ be two non parallel affine functions where for some x, $f(x) = \ell_i(x)$ and for some other x, $f(x) = \ell_j(x)$. Then the intersection of $\ell_i(x)$ and $\ell_j(x)$ is the hyperspace H_{ij}:

$$H_{ij} := \ell_i(x) - \ell_j(x) = 0$$

The domain space is divided into two regions by the hyperspace H_{ij}. Performing this procedure for each pair of affine functions will then split the domain into what would be left a FBR partitioning.

FBR regions are regions where no two affine functions intersect with one another in the interior of that FBR region, and where the boundaries of each FBR region are uniquely defined by the intersections of pairs of affine functions.

Finest base regions will be defined accordingly:

Definition 4-1.1. (Finest Base Region): Assume $\Omega = \bigcup_{i=1}^{N_\Phi} \Phi_i$ the regions Φ_i are FBR regions if

1. Each Φ_i is a convex polyhedron.
2. The boundaries of each Φ_i are $(n-1)$-dimensional hyper-planes.
3. Each pair of $\text{int}(\Phi_i)$ and $\text{int}(\Phi_i)$ are disjoint, i.e.

$$\text{int}(\Phi_i) \cap \text{int}(\Phi_j) = \emptyset, \quad \forall i, j \in \{1, ..., N_\Phi\}, i \neq j \tag{4-1}$$

   where $\text{int}(\Phi_i)$ denotes the interior of Φ_i
4. The intersection

$$\{x | \ell_j(x) = \ell_i(x), j = 1, ..., N_\Phi, j \neq i\} \cap \text{int}(\Phi_i) \tag{4-2}$$

   is empty for all $i \in \{1, ..., N_\Phi\}$

Base regions, denoted by $\mathbb{D}$, as described in Chapter 3, are similar; every base region is constructed such that the *local* (or *active*) function of that region does not intersect with any of the other affine functions within the interior. In other words, every FBR region is also a base region, but not every base region is a FBR region.

Example 9. [34] Now let us demonstrate the FBR partitioning with an example. Consider a 2-dimensional conjunctive MMPS function with $N_{\text{aff}} = 3$ unique affine functions, with constraints $0 \leq x_1 \leq 1$ and $0 \leq x_2 \leq 1$:

$$\begin{aligned} f(x) = \min \big(& \\ & \max(8x_1 - 5x_2 - 1, -5x_1 + 8x_2 - 1), \\ & \max(0) \\ \big) & \end{aligned}$$

Which generates the following output:

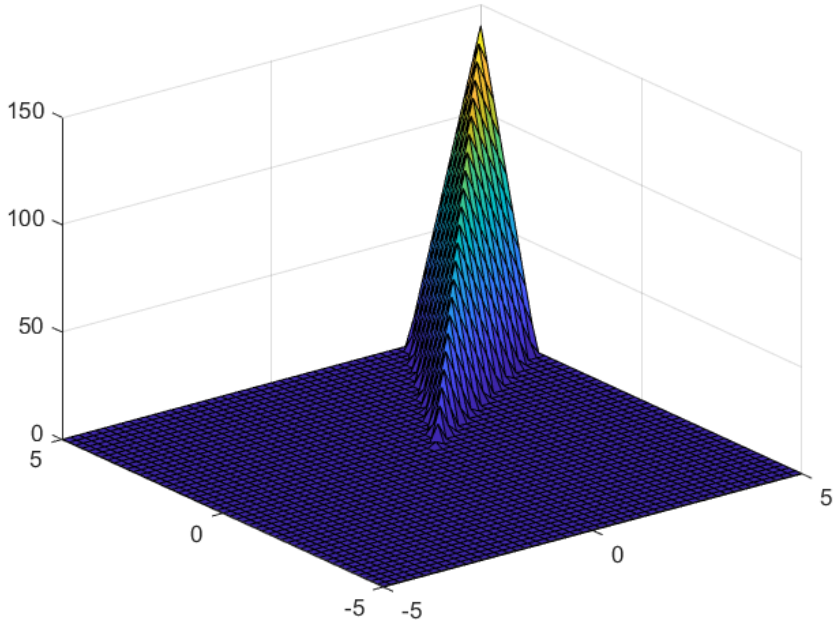


Figure 4-3: Plot of Example 9. A 2-dimensional conjunctive MMPS function.

Clearly the function has the unique affine functions $\ell_1(x), \ell_2(x), \ell_3(x)$, with:

$$\begin{aligned} \ell_1(x) &= 8x_1 - 5x_2 - 1 \\ \ell_2(x) &= -5x_1 + 8x_2 - 1 \\ \ell_3(x) &= 0 \end{aligned}$$

The half-spaces are defined as the intersections between the affine functions of $f(x)$:

$$\begin{aligned} H_1 &:= \ell_1(x) \cap \ell_2(x) := x_1 - x_2 = 0 \\ H_2 &:= \ell_1(x) \cap \ell_3(x) := 8x_1 - 5x_2 = 1 \\ H_3 &:= \ell_2(x) \cap \ell_3(x) := -5x_1 + 8x_2 = 1 \end{aligned}$$

We can now plot same function but with each half-space partitioning the domain:

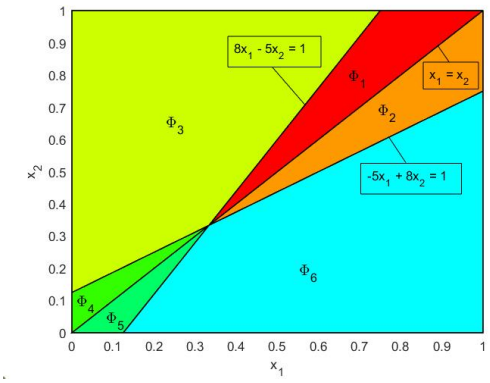


Figure 4-4: A projection of Example 9 with the hyperspaces partitioning the domain.

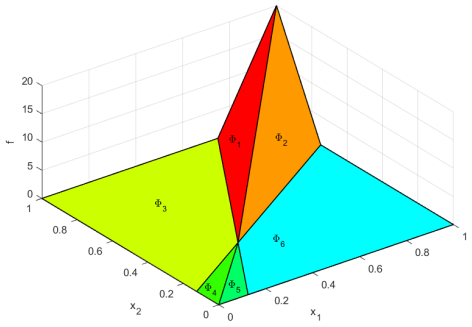


Figure 4-5: A plot of Example 9 with the hyperspaces partitioning the domain.

Finest Base Regions Plotting the hyperspaces reveals the FBR regions. Each FBR region can be constructed using every unique half-space: The partition is performed according to Definition 4-1.1. The FBR regions can then be expressed as convex polyhedra using the half-spaces, and the domain bounds. This gives us the following regions:

$$\begin{aligned}
\Phi_1 &: -8x_1 + 5x_2 \leq -1, -x_1 + x_2 \leq 0, x_2 \leq 1 \\
\Phi_2 &: x_1 - x_2 \leq 0, -5x_1 + 8x_2 \leq 1, x_1 \leq 1 \\
\Phi_3 &: 8x_1 - 5x_2 \leq 1, 5x_1 - 8x_2 \leq -1, -x_1 \leq 0, x_2 \leq 1 \\
\Phi_4 &: -5x_1 + 8x_2 \leq 1, x_1 - x_2 \leq 0, -x_1 \leq 0 \\
\Phi_5 &: -x_1 + x_2 \leq 0, 8x_1 - 5x_2 \leq 1, -x_2 \leq 0 \\
\Phi_6 &: -8x_1 + 5x_2 \leq -1, -5x_1 + 8x_2 \leq 1, x_1 \leq 1, -x_2 \leq 0
\end{aligned}$$

or alternatively in matrix notation:

$$\Phi_1 : \begin{pmatrix} -8 & 5 \\ -1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}, \Phi_2 : \begin{pmatrix} 1 & -1 \\ -5 & 8 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \Phi_3 : \begin{pmatrix} 8 & -5 \\ 5 & -8 \\ -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ -1 \\ 0 \\ 1 \end{pmatrix},$$

$$\Phi_4 : \begin{pmatrix} -5 & 8 \\ 1 & -1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \Phi_5 : \begin{pmatrix} -1 & 1 \\ 8 & -5 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \Phi_6 : \begin{pmatrix} -8 & 5 \\ -5 & 8 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

4-2 Parameter matrix

The FBR region partitioning assumes the parameter matrix ϕ (holding the parameters of all unique affine functions) to be available. Computing the parameter matrix is straightforward for the case of continuous PWA functions and conjunctive MMPS functions, and slightly less straightforward for kripfganz MMPS functions and general MMPS functions. Each case is discussed in the sections below

4-2-1 Parameter matrix of continuous PWA functions

For continuous PWA functions the rows i of the parameter matrix ϕ correspond to the parameters of the affine function $\ell_i(x)$:

$$\phi_i = \begin{pmatrix} \alpha_i & \beta_i \end{pmatrix} \tag{4-3}$$

The procedure is straight forward, as each affine function is explicitly listed in the function itself. By looping over all regions the affine functions can be stored in a parameter matrix. Recurring affine functions would be simply skipped before copying over, or remove afterwards.

Example 10. (Example 1 continued) Consider our previous 1-dimensional example again:

$$f(x) = \begin{cases} \ell_1(x) = 0.5x + 0.5 & x \in [0.0, 1.0] \\ \ell_2(x) = 2x - 1 & x \in [1.0, 1.5] \\ \ell_3(x) = 2 & x \in [1.5, 3.5] \\ \ell_4(x) = -2x + 9 & x \in [3.5, 4.0] \\ \ell_5(x) = -0.5x + 3 & x \in [4.0, 5.0] \end{cases} \tag{4-4}$$

It should be straight forward that this gives us the following parameter matrix:

$$\phi = \begin{pmatrix} 0.5 & 2 & 0 & -2 & -0.5 \\ 0.5 & -1 & 2 & 9 & 3 \end{pmatrix}^{\mathrm{T}}$$

4-2-2 Parameter matrix of conjunctive MMPS functions

For the conjunctive MMPS form the procedure of finding the parameter matrix is again straight forward. If the function is expressed in sets (a structure matrix ψ and parameter matrix ϕ)the parameter matrix is part of the expression. If instead the function is expressed as a 3-dimensional matrix α_{ij} and β_{ij} the procedure then would be to loop over each matrix and store each affine function:

$$\phi_i = \begin{pmatrix} \alpha_{ij} & \beta_{ij} \end{pmatrix} \tag{4-5}$$

Each recurring affine function (in the following matrix) would be simply skipped before copying over, or remove in a step afterwards.

Example 11. (Example 9 continued) Consider the following 2-dimensional disjunctive MMPS function:

$$\begin{aligned} f(x) = \max\big(& \\ & \min(8x_1 - 5x_2 - 1, -5x_1 + 8x_2 - 1), \\ & \min(-5x_1 + 8x_2 - 1), \\ & \min(8x_1 - 5x_2 - 1, 0) \\ \big) & \end{aligned}$$

Where the α and β matrices are:

$$\alpha_1 = \begin{pmatrix} 8 & -5 \\ -5 & 8 \\ \epsilon & \epsilon \end{pmatrix}, \quad \alpha_2 = \begin{pmatrix} \epsilon & \epsilon \\ -5 & 8 \\ \epsilon & \epsilon \end{pmatrix}, \quad \alpha_3 = \begin{pmatrix} 8 & -5 \\ \epsilon & \epsilon \\ 0 & 0 \end{pmatrix}$$

$$\beta_1 = \begin{pmatrix} -1 \\ -1 \\ \epsilon \end{pmatrix}, \quad \beta_2 = \begin{pmatrix} \epsilon \\ -1 \\ \epsilon \end{pmatrix}, \quad \beta_3 = \begin{pmatrix} -1 \\ \epsilon \\ 0 \end{pmatrix}$$

It should be clear that the parameter matrix is given by:

$$\phi = \begin{pmatrix} 8 & -5 & -1 \\ -5 & 8 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

Note that the recurring affine parts in the second and third sets (after already being included in parameter matrix) are not added again (or instead be removed after included each row).

4-2-3 Parameter matrix of Kripfganz MMPS functions

Extracting the unique affine functions from a kripfganz MMPS form requires some simple processing. Be reminded that the kripfganz form can be expressed as either the difference of two (convex) continuous PWA functions or the difference of two maximization (or minimization) functions. By using the algebraic rules for MMPS functions and finding all possible unique affine functions using combinatorics, the parameter matrix can be found.

Using algebraic rules As a general method one can find every possible combination between the affine functions of the two convex parts and compute the difference to acquire the set of unique affine functions. Suppose we have the convex functions $g(x)$ and $h(x)$ with each contain N_g and N_h number of unique affine function, and let these affine functions be contained inside their respective parameter matrices ϕ_g and ϕ_h, then the parameter matrix of the function $f(x)$ is given by:

$$\phi_i = \phi_{g,j} - \phi_{h,k}$$

for $j \in \{1, ..., N_g\}$ and $k \in \{1, ..., N_h\}$. The resulting parameter matrix ϕ will then be of size $(N_g N_h) \times (n + 1)$.

Example 12. (Example 1 continued) Consider the 1-dimensional kripfganz MMPS function:

$$f(x) = \max(1.5x + 2, 8, -1.5x + 9.5) - \max(2x - 1, 6, -2x + 9)$$

Then the parameter matrix can be found by considering all possible combinations between the parameter of $g(x)$ and $h(x)$. We have the following combination for $f(x)$:

$$f(x) = \begin{cases} (1.5x + 2) - (2x - 1) \\ (1.5x + 2) - (6) \\ (1.5x + 2) - (-2x + 9) \\ (8) - (2x - 1) \\ (8) - (6) \\ (8) - (-2x + 9) \\ (-1.5x + 9.5) - (2x - 1) \\ (-1.5x + 9.5) - (6) \\ (-1.5x + 9.5) - (-2x + 9) \end{cases} = \begin{cases} -0.5x + 3 \\ 1.5x - 4 \\ 3.5x - 7 \\ -2x + 9 \\ 2 \\ 2x - 1 \\ -3.5x + 10.5 \\ -1.5x + 15.5 \\ 0.5x + 0.5 \end{cases}$$

Which gives the parameter matrix:

$$\phi = \begin{pmatrix} -0.5 & 1.5 & 3.5 & -2 & 0 & 2 & -3.5 & -1.5 & 0.5 \\ 3 & -4 & -7 & 9 & 2 & -1 & 10.5 & 3.5 & 0.5 \end{pmatrix}^{\mathrm{T}}$$

4-2-4 Parameter matrix of general MMPS functions

To acquire the parameter matrix from a general MMPS function there are at least two methods we may try. First by using the algebraic rules for MMPS functions, and following a similar procedure as explained for kripfganz functions.

Method using algebraic rules The general MMPS function is recursively defined, using the operation maximization, minimization, addition and scalar multiplication. By finding all the possible combination between the maximization and minimization terms, and then by following the operations addition and scalar multiplication, each possible affine function can be found.

Example 13. Consider the following 1-dimensional general MMPS function in *general form*:

$$f(x) = \min(8x + 6, 1) - 2\max(\min(2x + 1, -2x + 1), -2x)$$

First we can find the parameter matrix by computing each possible outcome. This can be demonstrated by replacing the minimization and maximization operations with an logical *"or"*. Then we get the following possible combination for $f(x)$:

$$f(x) = \begin{cases} (8x+6) - 2(2x+1) \\ (8x+6) - 2(-2x+1) \\ (8x+6) - 2(-2x) \\ (1) - 2(2x+1) \\ (1) - 2(-2x+1) \\ (1) - 2(-2x) \end{cases} = \begin{cases} 4x + 4 \\ 12x + 4 \\ 12x + 6 \\ -4x - 1 \\ 4x - 1 \\ -4x + 1 \end{cases}$$

This would then create the following parameter matrix:

$$\phi = \begin{pmatrix} 4 & 12 & 12 & -4 & 4 & -4 \\ 4 & 4 & 6 & -1 & -1 & 1 \end{pmatrix}^{\mathrm{T}}$$

4-2-5 Redundancy in the parameter matrix

For each of the four cases the resulting matrix could have repeating repeated affine functions (identical rows). It should be trivial these rows can easily be removed with a single line of code.

However in the case for MMPS functions, there may exist redundant rows, i.e. affine functions that are never active. Moreover, the procedures used for finding the parameter matrix of the

kripfganz and general MMPS functions create redundant rows. Because (we assume) there is no knowledge of the regions or active functions, it is not trivial to find these redundant rows.

One strategy to find the redundant rows could be to use the conjunctive form as an intermediate step and by rewriting the kripfganz or general form as a conjunctive form. From the conjunctive form, the problem can be reduced to finding an irredundant H-representation for each of the sets as regions.

Removing the redundant rows is not a necessary step. In fact, after the partition has been made, identifying the redundant parameter becomes trivial. A consequence of including redundant rows in the parameter matrix, and therefor also the partition is that the resulting partition will be finer, i.e. there will be more regions created. For larger problems this quickly becomes unacceptable.

4-3 Ordered lattice representation

This section is an explorative study on the ordering of the affine functions inside a FBR region. The ordered lattice representation is proposed as a novel representation for the canonical MMPS forms. The ordered lattice representation can be viewed as an extension of the conjunctive MMPS form; as in the conjunctive form the sets are constructed as lattices of the affine function, with the addition of also storing the arrangement of the affine functions. From the ordered lattice representation the continuous PWA function and canonical MMPS functions can be easily realised.

We explore some of the properties which may be considered important within the scope of this thesis. Some techniques are presented to find specific relations between OL sets and bridge the gap between OL sets and polyhedral regions. The following topics are discussed:

- Finest base regions from ordered lattices
- Adjacent regions from ordered lattices
- Convex folds from ordered lattices
- PWA domain regions from ordered lattices

Example 14. (Example 9 continued) Let us first demonstrate the idea of an ordered lattice set with an example. Consider the 2-dimensional example again. We plot each function over the entire domain to get a better view of the ordering of each affine functions.

Now for each FBR region we would like to find the ordering of each affine function. After carefully looking at each region we can see that the affine functions $\ell_1(x), \ell_2(x), \ell_3(x)$ are ordered in the following way for each fbr region: $\Phi_1 := \ell_2 \geq \ell_1 \geq \ell_3$, $\Phi_2 := \ell_1 \geq \ell_2 \geq \ell_3$, $\Phi_3 := \ell_3 \geq \ell_1 \geq \ell_2$, $\Phi_4 := \ell_3 \geq \ell_2 \geq \ell_1$, $\Phi_5 := \ell_2 \geq \ell_3 \geq \ell_1$, $\Phi_6 := \ell_1 \geq \ell_3 \geq \ell_2$

4-3-1 Modified structure matrix

The general idea is to extend the structure matrix such that the ordering can be stored. One method would be to store the unique affine functions inside a modified structure matrix γ of size $N_{\text{aff}} \times N_{\Phi}$.

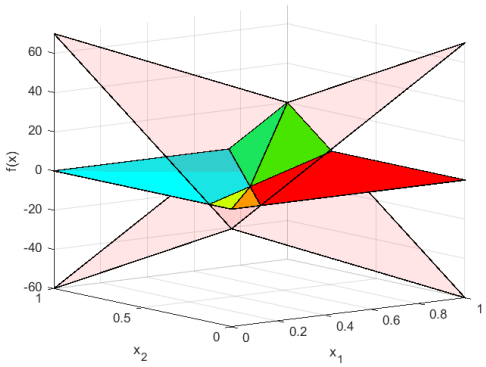


Figure 4-6: Front view of the plot of Example 9 with each affine function spanning the entire domain.

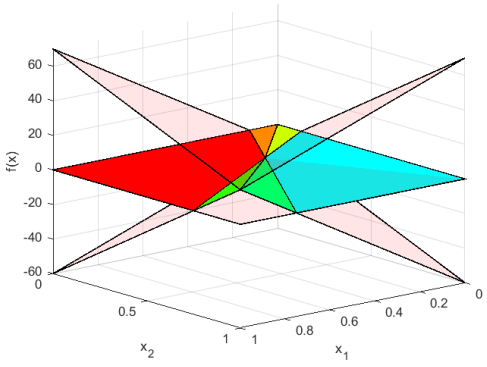


Figure 4-7: Rear view of the plot of Example 9 with each affine function spanning the entire domain.

The indices would represent the rows from the matrix that stores the unique affine functions, the component would be an integer k where $k \in \{1, ..., N_{\text{aff}}\}$. Here the value of k would represent the relative position inside the lattice between the affine functions.

Let the modified structure matrix be defined as follows:

$$\gamma_{ij} = k, \tag{4-6}$$

With $i, k \in \{1, ..., N_{\text{aff}}\}$, $j \in \{1, ..., N_{\Phi}\}$. The value k is determined by the relative magnitude of each i-th affine function. The largest affine function has the value $k = 1$, the smallest affine function $k = N_{\text{aff}}$, and all other possible values of k for the other affine functions. The ordered lattices can be constructed as using permutations:

Definition 4-3.1. [9] *Permutations.* Suppose that a set has n elements. Suppose that an experiment consists of selecting k of the elements one at a time without replacement. Let each outcome consist of the k elements in the order selected. Each such outcome is called a *permutation of n elements taken k at a time.* We denote the number of distinct such permutations by the symbol $P_{n,k}$.

Theorem 4-3.1. [9] *Permutations.* The number of permutations of n elements taken k at a time is $P_{n,k} = n(n-1)\cdots(n-k+1)$.

Each column j of the modified structure matrix is then one permutation of the affine functions.

Example 15. (Example 9 continued) Consider again our previous example. The structure matrix would then be written as follows:

$$\gamma = \begin{pmatrix} 2 & 1 & 3 & 3 & 2 & 1 \\ 1 & 2 & 1 & 2 & 3 & 3 \\ 3 & 3 & 2 & 1 & 1 & 2 \end{pmatrix}$$

4-3-2 Finding the active function using ordered lattices

With an ordered lattice it is easy to find the active function in the case of the conjunctive MMPS form. For each ordered lattice γ_i the conjunctive function can be evaluated to find

the active function. In other words, the solution of each maximization and minimization operation from the conjunctive MMPS form can be found in the ordered lattice set. Let $\ell_k(x)$ and $\ell_l(x)$ be two affine functions from some collection of affine functions. Then given a column i of an ordered lattice representation, the solution of the operations maximization and minimization are:

$$\max(\ell_k, \ell_l) = \min(\gamma_{ik}, \gamma_{il}) \tag{4-7}$$
$$\min(\ell_k, \ell_l) = \max(\gamma_{ik}, \gamma_{il}) \tag{4-8}$$

if the ordered lattice set is in ascending order.

Example 16. (Example 2 continued) First we repeat the conjunctive MMPS function:

$$\begin{aligned} f(x) = \max\big(& \\ & \min(\ell_1(x), \ell_2(x)), \\ & \min(\ell_2(x)) \\ & \min(\ell_1(x), \ell_3(x)) \\ \big) & \end{aligned}$$

Now let us analyse the following ordered lattice set: $\gamma_6 = \begin{pmatrix} 1 & 3 & 2 \end{pmatrix}^T$. First we can solve the inner maximization terms:

$$\begin{aligned} \min_{\gamma_6}(\ell_1(x), \ell_2(x))) &= \ell_2(x) \\ \min_{\gamma_6}(\ell_2(x))) &= \ell_2(x) \\ \min_{\gamma_6}(\ell_1(x), \ell_3(x))) &= \ell_3(x) \end{aligned}$$

Then we can solve the outer minimization:

$$\max_{\gamma_6}(\ell_2(x), \ell_3(x))) = \ell_3(x)$$

We can verify from the plot that the active function in the FBR region Φ_6 is indeed $\ell_3(x)$. Now we can perform this procedure for each column and get to following active functions: $\ell_1(x), \ell_2(x), \ell_3(x), \ell_3(x), \ell_3(x), \ell_3(x)$, for regions Φ_1-Φ_6 respectively.

4-3-3 Adjacent regions from ordered lattices

How would we know when two regions are adjacent to each other? First two regions will be considered adjacent when they are separated by a hyperplane of order $n-1$ with n the order of the function f. One strategy would be to find all pairs where the order or arrangement of the OL set are identical except where two values are switches. Thus two ordered lattice sets γ_i and γ_j are adjacent when they are equal, except for the rows m, n:

$$\begin{aligned} \gamma_{i,m} &= \gamma_{j,n} \\ \gamma_{i,n} &= \gamma_{j,m} \end{aligned}$$

For $m, n \in \{1, ..., N_{\text{aff}}\}$.

Example 17. (Example 9 continued) Consider the following two sets from example 2:

$$\gamma_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad \gamma_6 = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} \tag{4-9}$$

The two are identical with the exception that the last two values are switched. Thus this would conclude the sets are the representation of two regions that are adjacent. Following the same procedure gives the following pairs of adjacent regions:

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}, \quad \begin{pmatrix} 2 & 3 \\ 1 & 1 \\ 3 & 2 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 3 & 2 \end{pmatrix}, \quad \begin{pmatrix} 2 & 1 \\ 3 & 3 \\ 1 & 2 \end{pmatrix}, \quad \begin{pmatrix} 3 & 2 \\ 2 & 3 \\ 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} 3 & 3 \\ 1 & 2 \\ 2 & 1 \end{pmatrix}$$

4-3-4 Convex Folds from ordered lattices

Adjacent regions described as ordered lattice sets can be identified using the method described above. Now a method is proposed how any two regions may be a convex fold. We assume knowledge of the active functions for two adjacent regions is present. First let all the affine functions be given in the parameter matrix ϕ. Then let i be the index position of affine functions in the parameter matrix, i.e. $\ell_i(x)$ corresponds to the parameters from the i-th row of ϕ.

Now let the two ordered lattice sets be adjacent as a result of the m and n rows being switched, i.e. $\gamma_{i,m} = \gamma_{j,n}$ and $\gamma_{j,m} = \gamma_{i,n}$. Then the fold is convex if:

$$\gamma_{i,m} < \gamma_{i,n}, \quad \text{if } \ell_i(x) = \ell_m(x)$$

and concave if

$$\gamma_{i,m} > \gamma_{i,n}, \quad \text{if } \ell_i(x) = \ell_m(x)$$

The two adjacent regions are neither convex nor concave (i.e. they have the same active function) when:

$$\ell_i(x) = \ell_j(x)$$

Example 18. (Example 9 continued) We can analyse each pair of adjacent region and notice the following sets are convex:

$$\begin{pmatrix} 2 & 3 \\ 1 & 1 \\ 3 & 2 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 3 & 2 \end{pmatrix}$$

And the concave folds:

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

4-3-5 Finest base regions from ordered lattices

Ordered lattices in themselves may be considered FBR regions given the definition. However we now discuss how an ordered lattice set may be rewritten as a polytope in H-representation.

First note that each pair of rows form an inequality representing a hyperspace. Suppose we have rows i and j and the value of their respective modified structure matrix is k_i, k_j and $k_i < k_j$. Then the hyperspace H_{ij} of this pair is given by:

$$\begin{aligned} H_{ij} &:= \ell_i(x) \leq \ell_j(x) \\ &:= (\alpha_i - \alpha_j)x \leq \beta_j - \beta_i \end{aligned}$$

Then by considering all combinations between the affine functions from the modified structure matrix, the resulting region will be a FBR region:

$$\Phi := \ell_i(x) - \ell_j(x) \leq 0, \quad \text{if } \ell_i(x) \leq \ell_j(x), \quad \forall i, j \in \{1, ..., N_{\text{aff}}\} \tag{4-10}$$

This brings us to the following lemma:

Lemma 4-3.2. An ordered lattice set and finest base region are equivalent.

Although we have previously established the theory on the upper and lower bound of the FBR regions, the ordered lattice can be used to produce an answer to this question as well. The number of ordered lattice sets follows from the definition of permutation:

$$\begin{aligned} N_{\text{OL}} &= N_{\text{aff}}(N_{\text{aff}} - 1)(N_{\text{aff}} - 2)...(N_{\text{aff}} - N_{\text{aff}} + 1) \\ &= N_{\text{aff}}! \end{aligned}$$

By the equivalence between FBR regions and ordered lattices, we can conclude the number of FBR regions can also be found by the number of ordered lattices. Note that this is still an upper-bound.

Example 19. (Example 9 continued) Consider a single column of the structure matrix in the example, column $\gamma_3 = \begin{pmatrix} 2 & 1 & 3 \end{pmatrix}$. We can then construct the FBR region Φ_3

$$\begin{aligned} \Phi_3 : \begin{pmatrix} \ell_2 - \ell_1 \\ \ell_2 - \ell_3 \\ \ell_1 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &\leq \mathbf{0} \\ = \begin{pmatrix} -13 & 13 \\ -5 & 8 \\ 8 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &\leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

Each column represents a unique finest base region, which we then construct:

$$\Phi_1 : \begin{pmatrix} \ell_1 - \ell_2 \\ \ell_1 - \ell_3 \\ \ell_2 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \Phi_2 : \begin{pmatrix} \ell_1 - \ell_3 \\ \ell_1 - \ell_2 \\ \ell_3 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \Phi_3 : \begin{pmatrix} \ell_2 - \ell_1 \\ \ell_2 - \ell_3 \\ \ell_1 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0},$$

$$\Phi_4 : \begin{pmatrix} \ell_2 - \ell_3 \\ \ell_2 - \ell_1 \\ \ell_1 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \Phi_5 : \begin{pmatrix} \ell_3 - \ell_1 \\ \ell_3 - \ell_2 \\ \ell_1 - \ell_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \Phi_6 : \begin{pmatrix} \ell_3 - \ell_2 \\ \ell_3 - \ell_1 \\ \ell_2 - \ell_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}$$

Rewriting the above inequalities into *normal form* we get the regions:

$$\Phi_1 : \begin{pmatrix} 13 & -13 \\ 8 & -5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \Phi_2 : \begin{pmatrix} -8 & 5 \\ 13 & -13 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \Phi_3 : \begin{pmatrix} -13 & 13 \\ -5 & 8 \\ 8 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix},$$
$$\Phi_4 : \begin{pmatrix} -5 & 8 \\ -13 & 13 \\ -8 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \Phi_5 : \begin{pmatrix} -8 & 5 \\ -5 & 8 \\ 13 & -13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}, \Phi_6 : \begin{pmatrix} 5 & -8 \\ -8 & -5 \\ -13 & 13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}$$

4-3-6 PWA regions from ordered lattice sets

Consider again two adjacent regions which are identified by the rows m, n. When both regions share the same active function the regions can be united. Note that some regions may share the same active function, but are not necessarily adjacent. Only when both conditions are met may the regions be united. Multiple adjacent regions can be united.

Example 20. (Example 9 continued). The modified structure matrix was given by:

$$\gamma = \begin{pmatrix} 2 & 1 & 3 & 3 & 2 & 1 \\ 1 & 2 & 1 & 2 & 3 & 3 \\ 3 & 3 & 2 & 1 & 1 & 2 \end{pmatrix} \tag{4-11}$$

For which the pairs of adjacent regions we found (γ_1, γ_2), (γ_2, γ_3), (γ_3, γ_4), (γ_4, γ_5), (γ_5, γ_6), (γ_6, γ_1) and the active functions being $\ell_{\gamma_1} = \ell_1$, $\ell_{\gamma_2} = \ell_2$, $\ell_{\gamma_3} = \ell_3$, $\ell_{\gamma_4} = \ell_3$, $\ell_{\gamma_5} = \ell_3$, $\ell_{\gamma_6} = \ell_3$. Then we can see that the pairs (γ_3, γ_4), (γ_4, γ_5), (γ_5, γ_6), share the same local affine function. Which we can thus merge. This gives us the following continuous PWA regions:

$$\Omega_1 = \Phi_1 = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$$
$$\Omega_2 = \Phi_2 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$
$$\Omega_3 = \Phi_3 \cup \Phi_4 \cup \Phi_5 \cup \Phi_6 = \begin{pmatrix} 3 & 3 & 2 & 1 \\ 1 & 2 & 3 & 3 \\ 2 & 1 & 1 & 2 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 1 & -1 \\ -1 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

4-4 Continuous PWA function realisation from FBR regions

To maintain a better overview, the techniques for realising the partition into FBR regions will have a dedicated chapter, Chapter 5. It will be demonstrated first how the FBR regions can be exploited to construct the continuous PWA function. In the next chapter three different methods will be presented for the realisation of FBR regions and analysed in their complexity.

For the realisation of a continuous PWA function the following three assumptions are made:

- The FBR partition in H-representation is available.

- The original function, which can be a general-, conjunctive-, kripfganz MMPS function or a continuous PWA function, is available for evaluation.

The realisation also relies on having two secondary sources of information:

- The parameter matrix ϕ.
- An interior point for the FBR region. That is, a point that lies within the interior of a FBR regions, and not on it's boundary.

The methods for obtaining the parameter matrix has been discussed in the previous section. The interior point is used to evaluate the original function. Finding an interior point that is guaranteed to be within the FBR region is not trivial. It is also a step used by the algorithms to compute the FBR regions. It is therefor assumed this information is available. The methods used to calculate an interior point is discussed in Chapter 5.

4-4-1 Finding the active function

A continuous PWA function can be easily constructed by realising that each FBR region already is a continuous PWA region and only the active function would need to be found for the corresponding FBR region. First let the original function, in any MMPS form or continuous PWA form, be given by $f(x) : \Omega \to \mathbb{R}^n$ where Ω represent a convex polyhedral over which the function is defined. Then let the continuous PWA function be defined as:

$$f_{\text{cpwa}}(x) = f_i(x) := \alpha_i(x) + \beta_i, \quad x \in \Phi_i \tag{4-12}$$

Where $\Omega = \bigcup_{i=1}^{N_\Phi} \Phi_i$ and where $f_i : \Phi_i \to \mathbb{R}$ is called the local function on the FBR region Φ_i.

Finding the active function can be achieved by having some point inside the interior of the FBR region and evaluate the 'original' function $f(x)$ and the parameter matrix ϕ at this point. Because there are no intersecting affine functions inside the FBR region each affine function, or row of the parameter matrix, evaluates uniquely at this point, which can then be compared against the original function. If these two values are equal, the active function for the FBR region has been found. Let ℓ_{act,Φ_i} be the active function for FBR region Φ_i, $x_{\text{cc,i}}$ a point within the interior of the FBR region, $f(x)$ the original function and ϕ the parameter matrix. Then we find the active function ℓ_{act,Φ_i} as the row from $\phi(x_{\text{cc,i}})$, which equals the original function $f(x_{\text{cc,i}})$:

$$\ell_{\text{act},\Phi_i}(x) = \phi_j, \quad \text{if } f(x_{\text{cc,i}}) = \phi_j(x_{\text{cc,i}}) \tag{4-13}$$

The resulting continuous PWA function is clearly not in minimal form and the strategies discussed in Chapter 3 can be used to find a minimal form of the PWA function.

4-5 Canonical MMPS function realisation from FBR regions

For the realisation of the two canonical MMPS we again make the same assumptions as for the case for continuous PWA functions, i.e. we have the FBR partition in H-representation, the original function, the parameter matrix and an interior point for each FBR region available.

The procedure's to realise the canonical MMPS form from FBR regions closely follow the procedures from Chapter 3. In fact, the FBR partitioning has essentially created a continuous PWA function and we may deploy any of the procedures from Chapter 3. There are some small intermediate steps which will be discussed in the next two sections:

4-5-1 Conjunctive MMPS realisation from FBR regions

The conjunctive MMPS function can be realized using a similar approach as described in [35] and likewise in section 3-2-1. Be reminded that finding a conjunctive MMPS form revolves around finding the correct sets, or more specifically, the correct columns of the structure matrix ψ. For more information on the structure matrix, please refer to Chapter 2.

The strategies for the computation of the structure matrix in Chapter 3 essentially all use the same technique, but over a different domain partitioning. Each column of the structure matrix is made by finding all affine functions that are larger or equal to the active affine function for each region. In this case the regions are FBR regions. To find which affine functions are larger or equal to the active function, and interior point is used to evaluate all affine functions and the original function. Because of the way FBR regions are defined we are guaranteed to find a unique ordering inside the FBR regions.

Creating the structure matrix For each FBR region an interior point is assumed to be available. The interior point is then used to evaluate each affine function and original function for each region. This has the purpose to find the active function and each affine function smaller than the active function. This procedure creates index sets for each FBR region which are the index sets that describe the conjunctive function as described in Section 2-2-1. If the disjunctive function is requested, instead each affine function larger than the active function is collected.

First we assume the parameter matrix ϕ to be available. Be reminded that ϕ is a $N_{\text{aff}} \times (n+1)$ matrix, with n the dimension of $f(x)$ and N_{aff} the number of unique affine functions. Then we evaluate the original function and each affine function from the parameter matrix at the interior point. Then for each FBR regions Φ_i, with $i \in \{1, ..., N_\Phi\}$ and N_Φ the total number of FBR regions, we find the active function ℓ_{act,Φ_i} as the row from $\phi(x_{\text{cc,i}})$, which equals the original function $f(x_{\text{cc,i}})$:

$$\ell_{\text{act},\Phi_i}(x) = \phi_j, \quad \text{if } f(x_{\text{cc,i}}) = \phi_j(x_{\text{cc,i}}) \tag{4-14}$$

Where $j \in \{1, ..., N_{\text{aff}}\}$. Then selecting all rows that are equal or larger than the active function, and give their position a numerical 1:

$$\psi_{ij} = \begin{cases} 1, & \text{if } \phi_j(x_{\text{cc,i}}) \geq \ell_{\text{act},\Phi_i}(x) \\ 0, & \text{else} \end{cases} \tag{4-15}$$

for each FBR region Φ_i. Similarly a dual structure matrix for the disjunctive MMPS form $\hat{\psi}_{ij}$ is defined by

$$\hat{\psi}_{ij} = \begin{cases} 1, & \text{if } \phi_j(x_{\text{cc,i}}) \leq \ell_{\text{act},\Phi_i}(x) \\ 0, & \text{else} \end{cases} \tag{4-16}$$

4-5-2 Kripfganz realisation from FBR regions

For the realisation of a kripfganz MMPS function, we again refer to the methods from Chapter 3. The goal is to create a function in the form:

$$f(\mathrm{x}) = g(\mathrm{x}) - h(\mathrm{x}) \tag{4-17}$$

where $g(\mathrm{x})$ and $h(\mathrm{x})$ are two maximization (convex) functions or two minimization (concave) functions. Note that we had the following methods:

- Convex Folds
- Conjunctive MMPS realisation

Each of these methods can be deployed to the FBR partitioning. Again we assume the following: we have the FBR partition in H-representation, the original function, the parameter matrix and an interior point for each FBR region available.

Convex folds One strategy would be to borrow the technique used in [19] as described in Section 3-2-2, by collecting the convex folds, or pairs of FBR regions that are convex, as basis for the construction of $g(\mathrm{x})$, with:

$$g(\mathrm{x}) = \sum g_t \tag{4-18}$$

To find the convex function $h(\mathrm{x})$, one would then simply solve $h(\mathrm{x}) = g(\mathrm{x}) - f(\mathrm{x})$. However this procedure needs to be done for each FBR region to find $h(\mathrm{x})$. To do this we may again use an interior point to evaluate the function accordingly. For each FBR region each convex fold $g_t(\mathrm{x})$ is evaluated to find the active function. Then for that specific FBR region $h(\mathrm{x})$ can be constructed by summing the corresponding functions and subtracting the active function.

Conjunctive MMPS realisation Alternatively we can construct the kripfganz MMPS function by going to the conjunctive MMPS form as an intermediate step and then following the procedure described in Chapter 3. To find the conjunctive MMPS form from a FBR partition we follow the procedure described above.

4-6 Total number of FBR regions

One of the questions we would like to answer is how many FBR regions would be generated following the partition. That is *When n-dimensional space is partitioned by N_H hyperplanes (i.e., sub-spaces of dimension $n-1$), how many distinct m-dimensional sub-spaces are created?*. To find an answer to this question we make use of the study of *hyperplane arrangements*. As will be discussed, the answer to this question is straightforward for the upper-bound. However, for the lower-bound (some regions do not exist as a consequence of parallel and intersecting hyperplanes), the answer becomes much harder. The information necessary for the use of the FBR partition is reviewed in the following section.

4-6-1 Total number of intersection

First we consider the number of hyperplanes, or intersection that exist as a result of two intersecting affine functions. Thus for a set of affine functions where each two affine functions create a hyperplane, we are looking for all combinations between them. Combination are defined as follows:

Definition 4-6.1. [9] *Combinations.* Consider a set with n elements. Each subset of size k chosen from this set is called *combination of n elements taken k at a time.* We denote the number of distinct such combinations by the symbol $C_{n,k}$

Theorem 4-6.1. [9] *Combinations.* The number of distinct subsets of size k that can be chosen from a set of size n is

$$C_{n,k} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{4-19}$$

Now in the case of hyperplane arrangements we know that each half-space only produces two hyper-planes; one hyper-plane to describe each side of the half-space. Therefor we have $k = 2$. Now for any given set of N_{aff} affine functions we set $n = N_{\text{aff}}$. We can now simplify the possible combinations to

$$\begin{aligned} N_{\text{H}} &= \binom{N_{\text{aff}}}{2} \\ &= \frac{N_{\text{aff}}!}{2!(N_{\text{aff}}-2)!} \\ &= \frac{N_{\text{aff}}(N_{\text{aff}}-1)(N_{\text{aff}}-2)!}{2!(N_{\text{aff}}-2)!} \\ &= \frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2} \end{aligned}$$

Now that we know the number of hyperplanes, we can find an answer to how many regions are created by them.

4-6-2 Upper-bound on number of FBR regions

It is well known that N_{H} hyperplanes in the n-dimensional space $\mathbb{R}^n$, will divide $\mathbb{R}^n$ into a maximum number of regions according to the following theorem:

Theorem 4-6.2. (L. Schläfli [23]) *Let $\mathcal{A}$ be a collection of N_H hyperplanes in the affine space $\mathbb{R}^n$. Then the corresponding number of regions N_Φ^n satisfies the following inequality:*

$$\begin{aligned} N_\Phi^n &\leq \binom{N_{\text{H}}}{0} + \binom{N_{\text{H}}}{1} + \cdots + \binom{N_{\text{H}}}{n} \\ &= \sum_{i=0}^{n} \binom{N_{\text{H}}}{i} \end{aligned}$$

Note that where $\binom{p}{q} = 0$ if $q > p$. This theorem has been proven using a so called "Sweep" approach in [1, 5], or by a "Pascal triangle" method in [17, 37]. Note that this is for the case where the domain is with $\mathbb{R}^n$ and the domain is not bounded. The number of bounded regions (closed from all sided) is given by

$$N_{\Phi}^{n} = \binom{N_{\mathrm{H}} - 1}{n} \tag{4-20}$$

This upper limit on the number of FBR regions assumes the hyper-planes are in *general position*: there are no identical and parallel hyperplanes and no more than 2 hyperplanes share the same intersection. Finding the lower bound in $\mathbb{R}^2$ is fairly simple and intuitive. For the general case where $\mathbb{R}^n$ with $n > 2$ the problem becomes much harder.

4-6-3 Lower bound on number of FBR regions

Theorem 4-6.3. (Roberts' formula [22]) *Let $\mathcal{A}$ be a collection of d lines $\ell_1, ..., \ell_d$ in the plane $\mathbb{R}^2$. Then the corresponding number of regions is given by:*

$$N_{\Phi} = 1 + N_{\mathrm{H}} + \binom{N_{\mathrm{H}}}{2} - \sum_{i=1}^{k} n_k \binom{k-1}{2} - \sum_{j=1}^{p} \binom{\ell_j}{2} \tag{4-21}$$

where n_k is the number of k-fold intersection points in $\mathcal{A}$ for $k \geq 3$ and there are p families of parallel lines, containing respectively $\ell_1, ..., \ell_d$ lines with $\ell_j \geq 2$

It reads as: "the number of regions formed by n lines in general position" minus "the number of regions lost because of the multiple points" minus "the number of regions lost because of the parallels."

In higher dimensions, i.e. $\mathbb{R}^n$ with $n > 3$, the problem of finding a lower bound by considering parallel hyper-planes and more than two hyper-planes that share an intersection, becomes much harder.

Theorem 4-6.4. Zaslavsky's Theorem [36]: The number of regions into which space is split by the arrangement A can be found by adding together the absolute values of the Möbius function values given to the corresponding (semi-)lattice.

Given a vector space V and an arrangement A on V, the number of regions the vector space V is split into by A (denoted $N(A)$), can be expressed as follows:

$$N(A) = \sum_{x \in L(A)} (-1)^{r(x)} \mu(0, x) \tag{4-22}$$

The answer to this question is not of particular importance for the understanding of the algorithm, and we therefor consider the answer to this question outside of scope of this thesis.

4-7 Example: demonstration of a continuous PWA realisation using FBR regions

Two examples will be given that demonstrates the realisation of a continuous PWA function from a general MMPS function and a conjunctive MMPS function. First a simple 1-dimensional example and a 2-dimensional example.

Conjunctive MMPS function

Consider again the 1-dimensional conjunctive MMPS from Example 2:

$$f_{\text{cd}}(x) = \max\Big(\min(0.5x + 0.5, -0.5x + 3), \min(2x - 1, 2, -2x + 9)\Big) \tag{4-23}$$

The parameter matrix ϕ is clearly given by

$$\phi = \begin{pmatrix} 0.5 & 2 & 0 & -2 & -0.5 \\ 0.5 & -1 & 2 & 9 & 3 \end{pmatrix}^{\text{T}}$$

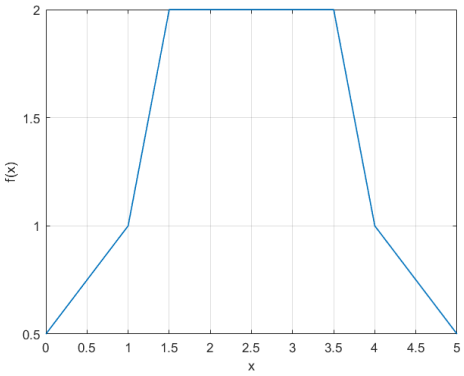


Figure 4-8: The plot from the 1-dimensional conjunctive example.

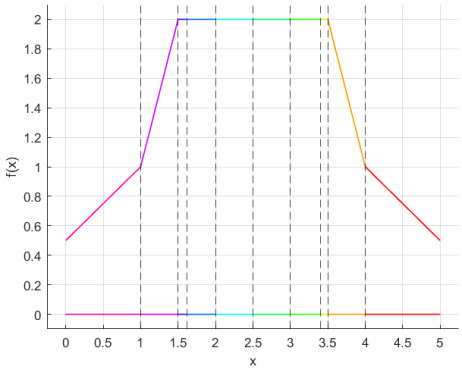


Figure 4-9: The same plot from the 1-dimensional conjunctive example, but with a FBR partition

Finest Base Region Partitioning From the parameter matrix we can find the total number of FBR regions. We have $N_{\text{aff}} = 5$ total of unique affine functions. Then the total number of hyperplanes N_{H} is given by:

$$\begin{aligned} N_{\text{H}} &= \frac{N_{\text{aff}}(N_{\text{aff}} - 1)}{2} \\ &= \frac{5(5-1)}{2} = 10 \end{aligned}$$

For a $n = 1$-dimensional problem, being divided by $N_{\mathrm{H}} = 10$ hyperplanes. However when looking at the graph, we notice two hyperplanes to be equal. Thus giving us $N_{\mathrm{H}} = 9$ unique hyperplanes. We thus expect a total number of FBR regions N_Φ to be:

$$\begin{aligned} N_\Phi &= \binom{N_{\mathrm{H}}}{0} + \binom{N_{\mathrm{H}}}{1} \\ &= 1 + N_{\mathrm{H}} \\ &= 1 + 9 = 10 \end{aligned}$$

Now by analysing the graph, the FBR regions can be identified. In H-representation the FBR regions are given by:

$$\Phi_1 = \begin{pmatrix} -1 & 0 \\ 1 & 1 \end{pmatrix}, \Phi_2 = \begin{pmatrix} -1 & 1 \\ 1 & 1.5 \end{pmatrix}, \Phi_3 = \begin{pmatrix} -1 & 1.5 \\ 1 & 1.6 \end{pmatrix}, \Phi_4 = \begin{pmatrix} -1 & 1.6 \\ 1 & 2 \end{pmatrix}, \Phi_5 = \begin{pmatrix} -1 & 2 \\ 1 & 2.5 \end{pmatrix},$$
$$\Phi_6 = \begin{pmatrix} -1 & 2.5 \\ 1 & 3 \end{pmatrix}, \Phi_7 = \begin{pmatrix} -1 & 3 \\ 1 & 3.4 \end{pmatrix}, \Phi_8 = \begin{pmatrix} -1 & 3.4 \\ 1 & 3.5 \end{pmatrix}, \Phi_9 = \begin{pmatrix} -1 & 3.5 \\ 1 & 4 \end{pmatrix}, \Phi_{10} = \begin{pmatrix} -1 & 4 \\ 1 & 5 \end{pmatrix}$$

Interior points Now we can compute a set of interior points. For this we will use the Chebycenter. This returns the following points:

$$x_{cc} = \begin{pmatrix} 0.5 & 1.25 & 1.56 & 1.8 & 2.25 & 2.75 & 3.2 & 3.45 & 3.75 & 4.5 \end{pmatrix}$$

Continuous PWA realisation from FBR regions Now for each point x_{cc} we can evaluate the original function $f(x)$ and each affine functions from the parameter matrix:

$$f(x_{cc}) = \begin{pmatrix} 0.75 & 1.5 & 2 & 2 & 2 & 2 & 2 & 2 & 1.5 & 0.75 \end{pmatrix}$$

$$\phi(x_{cc}) = \begin{pmatrix} 0.75 & 1.125 & 1.28 & 1.4 & 1.625 & 1.875 & 2.1 & 2.225 & 2.375 & 2.75 \\ 0 & 1.5 & 2.12 & 2.6 & 3.5 & 4.5 & 5.4 & 5.9 & 6.5 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 8 & 6.5 & 5.88 & 5.4 & 4.5 & 3.5 & 2.6 & 2.1 & 1.5 & 0 \\ 2.75 & 2.375 & 2.22 & 2.1 & 1.875 & 1.625 & 1.4 & 1.275 & 1.125 & 0.75 \end{pmatrix}$$

Finally we can identify the active function where $f_i(x_{cc}) = \phi_{i,j}(x_{cc})$ for $i \in \{1, ..., 5\}, j \in \{1, ..., 10\}$ Now we can rebuild the continuous PWA function:

$$f(x) = \begin{cases} \ell_1(x) = 0.5x + 0.5 & x \in \Phi_1 \\ \ell_2(x) = 2x - 1 & x \in \Phi_2 \\ \ell_3(x) = 2 & x \in \Phi_3 \\ \ell_3(x) = 2 & x \in \Phi_4 \\ \ell_3(x) = 2 & x \in \Phi_5 \\ \ell_3(x) = 2 & x \in \Phi_6 \\ \ell_3(x) = 2 & x \in \Phi_7 \\ \ell_3(x) = 2 & x \in \Phi_8 \\ \ell_4(x) = -2x + 9 & x \in \Phi_9 \\ \ell_5(x) = -0.5x + 3 & x \in \Phi_{10} \end{cases} \tag{4-24}$$

It is easy to verify that the regions $\Phi_3, ..., \Phi_7$ have the same active function $\ell_3(x)$. We can join the neiboring regions which share a same active function. This then gives us:

$$f(x) = \begin{cases} \ell_1(x) = 0.5x + 0.5 & x \in \Phi_1 \\ \ell_2(x) = 2x - 1 & x \in \Phi_2 \\ \ell_3(x) = 2 & x \in \Phi_3 \cup \Phi_4 \cup \Phi_5 \cup \Phi_6 \cup \Phi_7 \cup \Phi_8 \\ \ell_4(x) = -2x + 9 & x \in \Phi_9 \\ \ell_5(x) = -0.5x + 3 & x \in \Phi_{10} \end{cases} \tag{4-25}$$

Conjunctive MMPS function in 2-dimensions

Consider the 2-dimensional conjunctive MMPS function from Example 9. Remember that we have previously computed the parameter matrix and FBR regions. From the parameter matrix we can find the total number of FBR regions. We have $N_{\text{aff}} = 3$ total of unique affine functions. Then the total number of hyperplanes N_{H} is given by:

$$\begin{aligned} N_{\text{H}} &= \frac{N_{\text{aff}}(N_{\text{aff}} - 1)}{2} \\ &= \frac{3(3-1)}{2} = 3 \end{aligned}$$

Which we can verify by the previous results. The same can be done for the number of FBR regions. We thus expect a total number of FBR regions N_Φ to be:

$$\begin{aligned} N_\Phi &= \binom{N_{\text{H}}}{0} + \binom{N_{\text{H}}}{1} + \binom{N_{\text{H}}}{2} - \binom{2}{2} \\ &= 1 + N_{\text{H}} + \frac{N_{\text{H}}(N_{\text{H}} - 1)}{2} - \frac{2(2-1)}{2} \\ &= 1 + 3 + 3 - 1 = 6 \end{aligned}$$

And indeed we found a total of $N_\Phi = 6$ FBR regions. To find the active functions we can analyse the ordered lattices instead as previously shown. Then for each FBR region we get the following active functions:

$$\begin{aligned} &\ell_{\Phi_1} := \ell_1(x), \quad \ell_{\Phi_2} := \ell_2(x), \quad \ell_{\Phi_3} := \ell_3(x) \\ &\ell_{\Phi_4} := \ell_3(x), \quad \ell_{\Phi_5} := \ell_3(x), \quad \ell_{\Phi_6} := \ell_3(x) \end{aligned}$$

where ℓ_{Φ_i} indicates the active function for the FBR region Φ_i. The continuous PWA function would then be:

$$f_{\text{cpwa}}(x) = \begin{cases} \ell_1(x) = 8x_1 - 5x_2 - 1 & x \in \Phi_1 \\ \ell_2(x) = -5x_1 + 8x_2 - 1 & x \in \Phi_2 \\ \ell_3(x) = 0 & x \in \Phi_3 \\ \ell_3(x) = 0 & x \in \Phi_4 \\ \ell_3(x) = 0 & x \in \Phi_5 \\ \ell_3(x) = 0 & x \in \Phi_6 \end{cases} \tag{4-26}$$

It should be obvious that the resulting continuous PWA has FBR regions with equivalent active functions that could be combined into one region.

$$f_{\text{cpwa}}(x) = \begin{cases} \ell_1(x) = 8x_1 - 5x_2 - 1 & x \in \Phi_1 \\ \ell_2(x) = -5x_1 + 8x_2 - 1 & x \in \Phi_2 \\ \ell_3(x) = 0 & x \in \Phi_3 \cup \Phi_4 \cup \Phi_5 \cup \Phi_6 \end{cases} \tag{4-27}$$

4-8 Summary

In this chapter the concept of FBR regions (FBR regions) as a method to construct continuous PWA functions from MMPS functions was realised. FBR regions can be thought of as the projection of all intersections between the unique affine functions. The resulting grid is a partition of the domain over which the original function was defined.

The main benefit of this partition is that it can be performed using only information of the parameter matrix, or the unique affine functions, of the original function. Meaning the FBR partition can be used create a map of regions from any type of MMPS function.

Together with the parameter matrix, an interior point for each FBR region and the original function, a continuous PWA function can be constructed. General MMPS functions, conjunctive MMPS functions and Kripfganz MMPS functions can be rewritten in continuous PWA functions using the FBR partitioning. The resulting continuous PWA function is in a redundant form and as an additional step neighbouring FBR regions may be joined which share the same active function.

In the next chapter, three methods for the actual partitioning into FBR regions are discussed. The goal is to create an algorithm which accepts only a parameter matrix which then create the FBR regions.

Chapter 5

Finest Base Region Computation

In this chapter three algorithms are presented that can partition any Max-Min-Plus-Scaling (MMPS) function or continuous Piecewise Affine (PWA) function into a finest base regions. In the first algorithm an approach of hyperplane arrangements is used. The algorithm creates each Finest Base Region (FBR) region using all possible hyperplanes arrangements. In the second algorithm the property of lattices is exploited to create the FBR regions. Finally an algorithm that iteratively cuts the domain into finer regions for every added hyperplane.

5-1 Preliminaries

Polyhedral sets and their manipulation are an integral part of the algorithms. The fundamentals on polyhedral theory can be found in Appendix B. Two topics on polyhedral sets are used by each algorithm:

- Identifying empty regions (defined as polytopes)
- Finding an interior points regions

Before discussing the algorithms, two topics are treated.

5-1-1 Computing an interior point of a convex polytope

The algorithms use the calculation of an (arbitrary) interior point of a convex set. Under the assumption that the convex set is given in (non-minimal) H-representation the following methods are generally used for the calculation of an interior point:

- Analytical center
- Chebychev center

- Methods for V-represented sets

The first three methods evolve around solving a specific linear program. There is some (minor) performance differences, but are generally solved in polynomial time.

Analytical center The analytic center[4] of a set of m inequalities is defined as an optimal point for the (convex) problem

$$\min -\sum_{i=1}^{m} \log(b_i - a_i^\top x), \quad \text{for } i = \{1, ..., m\} \tag{5-1}$$

The objective function can be found in literature as the *logarithmic barrier*.

Chebyshev center Secondly, we may find the Chebyshev center for each region. Computation of the Chebyshev center corresponds to inscribing the largest ball inside a polyhedron[4]. The Chebyshev center of a closed bounded set is the set of solutions of the problem. The Chebyshev center and radius can be computed by solving the following linear program [16]:

$$\max r \tag{5-2}$$

$$\text{s.t. } a_i^\top x_c + ||a_i||_2 r \leq b_i, \quad i = 1, ..., m \tag{5-3}$$

$$A_e x_c = b_e \tag{5-4}$$

where a_i is the i-th row of the inequality matrix A in $Ax_c \leq b$ and $||a_i||_2$ is the 2-norm of that row.

Vertex description Alternatively, there exists the option to find the corresponding v-description (given a convex set in the h-description). This can also be found in literature as *vertex enumeration*. Then the available tools that use the v-description can instead be used. The existence of a total polynomial time vertex enumeration algorithm for polytopes remains unresolved [18]. Therefor this option is not considered viable throughout the rest of this thesis.

5-1-2 Polytopes with an empty interior

In the algorithms following this section one of the challenges is to identify empty or infeasible sets. Within the scope of the algorithm with "empty" we mean situation where the solutions of a set of inequalities lie within at least $\mathbb{R}^{n-1}$. Any solutions that only lie on a facet or vertices, i.e. the solutions lie within $\mathbb{R}^m$ where $m < n-1$, then we consider the set empty, or invalid to be a finest base region.

The reason for this convention is that if a projection becomes of a smaller order than $n-1$, is does not necessarily indicate an "empty region". For example in the case of $f : \mathbb{R}^2 \to \mathbb{R}$ where a region becomes a line, or a point.

To find a certificate of emptiness or feasibility we may use two approaches:

- Farka's lemma
- Chebshev center
- Hyperplane testing

Farka's lemma It is well know in the study of polyhedral sets that Farka's Lemma can give a certificate of feasibility of a set. Farka's lemma is stated as follows:

Lemma 5-1.1. Farka's lemma - Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then exactly one of the following two assertions is true:

1. There exists an $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$
2. There exists an $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{A}^\top \mathbf{y} \geq 0$ and $\mathbf{b}^\top \mathbf{y} < 0$

Chebycenter Alternatively, we may find the Chebyshev center for each region:

Definition 5-1.1. Chebyshev center - Computation of the Chebyshev center corresponds to inscribing the largest ball inside a polyhedron. The Chebyshev center of a closed bounded set is the set of solutions of the problem.

By finding the radius of the inscribed ball we can check if the region is empty or not: if the radius is greater than 0 we know the region is not empty; otherwise the region is empty.

The Chebyshev center and radius can be computed by solving the following linear program [16]:

$$\max r \tag{5-5}$$

$$\text{s.t. } a_i^\top x_c + ||a_i||_2 r \leq b_i, \quad i = 1, ..., m \tag{5-6}$$

$$A_e x_c = b_e \tag{5-7}$$

where a_i is the i-th row of the inequality matrix A in $Ax_c \leq b$ and $||a_i||_2$ is the 2-norm of that row.

5-2 Method 1 - FBR partition using hyperplane arrangements

The following algorithm is an initial attempt at acquiring the finest base region partition. In the first part of the algorithm, a continuous PWA function is partitioned into finest base regions.

In the first attempt to create a finest base region partition, the combinatorics of all intersections between the pairs of unique affine functions (of which the original function consists of), is performed to acquire all hyperspaces. Each region can then be constructed using again combinatorics of all possible hyperplane arrangements. This method creates multiple empty regions which would need to be identified and removed.

The algorithm takes as input either a continuous PWA or any MMPS function, and the domain over which the function is defined as hyperplanes. The algorithm outputs an continuous PWA function partitioned into FBR regions. The complete algorithm is structured as follows in pseudocode:

Algorithm 1 Method 1 - FBR partition using hyperplane arrangements

Input: [ϕ, bounds]
Output: [Φ]
$H = C_{N_{\text{aff}},2}(\phi)$ ▷ Collect all N_H hyper-spaces
$\hat{\Phi} = C_{N_{\text{aff}},2}(H^{-/+})$ ▷ Construct every i-th FBR region
for $i = 1 : 2^{N_H}$ **do**
    if $\hat{\Phi}_{(i)}$ is not empty **then**
        $\Phi_{(j)} = \hat{\Phi}_{(i)}$ ▷ Save region
    else
        Region is empty, do nothing
    end if
end for

5-2-1 FBR region construction

The first attempt to construct all finest base regions is based on combinatorics. The use of combinatorics is two fold:

- Half-space realisation
- Hyper-plane arrangement for region construction

In step one the intersect of each unique pair of affine functions (of n-order) can be found. The intersect is represented as a $(n-1)$-order half-space. Doing so for each possible pair of affine functions, returns all $(n-1)$-order half-spaces. Each half-space divides the space into two region which are both represented as a hyper-plane. Suppose this returns N-number of half-spaces, then we can construct each finest base region using N-number of hyper-planes. In step two, using combinatorics, every possible arrangement of the $(n-1)$-order hyper-planes is constructed to return every finest base region. Note that this step also generates invalid or empty regions.

Half-space realisation using combinatorics We can find all half-spaces by finding every unique pair of affine functions. The half-space can be found by solving for the pair of intersecting affine functions i and j:

$$\alpha_{(i)}\mathbf{x} + \beta_{(i)} = \alpha_{(j)}\mathbf{x} + \beta_{(j)} \tag{5-8}$$

with $i, j \in \{1, ..., N_\Phi\}$, $i \neq j$. Then every possible hyper-planes are:

$$(\alpha_{(i)} - \alpha_{(j)})\mathbf{x} \leq \beta_{(j)} - \beta_{(i)}$$
$$(\alpha_{(i)} - \alpha_{(j)})\mathbf{x} \geq \beta_{(j)} - \beta_{(i)}$$

The number of hyper-spaces that can be found are dependent on the number of functions $f_(i)$ that are parallel to each other. Consider the case where each function f_i is unique and not parallel, then the number of intersections L described as hyper-spaces are all the possible

combinations of pairs, where it is subject to the commutative property and thus can be found by classical combinations.

Now in the case of hyperplane arrangements we know that each half-space only produces two hyper-planes; one hyper-plane to describe each side of the half-space. Therefor we have $k = 2$. Now for any given set of N_{aff} affine functions we set $n = N_{\text{aff}}$. We can now simplify the possible combinations to

$$N_H = \binom{n}{k} = \binom{N_{\text{aff}}}{2} = \frac{N_{\text{aff}}!}{2!(N_{\text{aff}} - 2)!} \tag{5-9}$$

$$= \frac{N_{\text{aff}}(N_{\text{aff}} - 1)(N_{\text{aff}} - 2)!}{2!(N_{\text{aff}} - 2)!} \tag{5-10}$$

$$= \frac{N_{\text{aff}}(N_{\text{aff}} - 1)}{2} \tag{5-11}$$

Finally, using the combinations described above we would find all possible half-spaces and collect them in a single matrix A and b in *central form*:

$$H : A\mathbf{x} = b \tag{5-12}$$

Where A and b are given for each row k by

$$A_k = \alpha_i - \alpha_j$$
$$b_k = \beta_i - \beta_j$$

with $A^{N_H \times 2}$ and $b^{N_H \times 1}$, for all combinations of $i, j \in \{1, ..., N_{\hat{\Phi}}\}$, $i \neq j$ and $k \in \{1, ..., N_H\}$. Furthermore we get both hyper-plane matrices:

$$H^- : A^-\mathbf{x} \leq b^-$$
$$H^+ : A^+\mathbf{x} \geq b^+$$

With $A^- = A$, $b^- = b$ and $A^- = -A^+$, $b^- = -b^+$

Hyper-plane arrangements using combinatorics Next we can find every possible hyper-plane arrangement using a naive approach. The general idea is that each unique region can be constructed using every possible combination of hyper-planes. This generates both valid regions with redundant hyper-planes (non minimal form) and invalid or empty regions. However it does guarantee the creation of each region. Therefor the strategy is to initially create every region and include an additional step to identify and remove each empty region and optionally include a third step to find a minimal representation of the valid regions.

Consider a continuous PWA function of n-dimension with N_{aff} affine functions and no parallel functions. Then the number of hyper-spaces are N_H and the number of hyper-planes are $2N_H$. For each hyper-space only one hyper-plane can be active per region as any combination of two hyper-planes from the same hyper-space results in an empty region. Therefor each region can be described using a set of N_H hyper-planes. This set can be constructed using any

combination of hyper-planes. Let the total number of possible hyper-plane arrangement be denoted by $N_{\hat{\Phi}}$ then we find

$$N_{\hat{\Phi}} = 2^{N_H} \tag{5-13}$$

where N_H are the number of hyper-spaces. By adding the expression for the number of hyperplanes we can express the total number of initialised regions in terms of affine functions:

$$N_{\hat{\Phi}} = 2^{N_H} \tag{5-14}$$

$$= 2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)} \tag{5-15}$$

Note that $N_{\hat{\Phi}}$ is independent of the dimension n of the cPWA function and only dependent on that of the number of non parallel affine functions $N - K$. Let each resulting region be $\hat{\Phi}_i$ for $i \in \{1, ..., N_{\hat{\Phi}}\}$ that is defined by

$$\hat{\Phi}_i := \begin{cases} H_1^- \vee H_1^+ \\ H_1^- \vee H_2^- \\ \quad \vdots \\ H_{N_{\hat{\Phi}}}^- \vee H_{N_{\hat{\Phi}}}^+ \end{cases} \tag{5-16}$$

where $\vee$, stands for the logical operator *"or"*. This method also generates empty sets and thus an additional step is included to remove these.

Removing empty sets

The above described step generates combinations of hyper-planes, which result in empty sets. We may use one of the approaches described in the preliminaries to identify empty sets and remove them from the partition.

5-3 Method 2 - FBR partition using ordered lattices

In the next algorithm we perform an modification to algorithm 1 to reduce the number of empty sets that are created.

One may realise that it is not necessary to compute the hyper-spaces, but directly create the region using hyper-planes. Instead of using combinatorics, hyper-spaces and hyper-plane arrangements, we can create all possible regions by realising that each affine function occurs in a unique arrangement and that this arrangement can be written as inequalities which define that particular region.

The reason this is an improvement lies in the fact that this method of region constructing results in less empty sets that the method proposed previously. The complete algorithm is structured as follows in pseudocode:

Algorithm 2 Method 2 - FBR partition using lattices

Input: [ϕ, bounds]
Output: [Φ]
$I = P_{N_{\text{aff}}}$ ▷ Create all possible orders using combinatorics
$\hat{\Phi} = C_{I,2}$ ▷ Create region using all combinations of orders
for $i = 1 : N_{\text{aff}}!$ **do**
    if $\hat{\Phi}_i$ is not empty **then**
        $\Phi_j = \hat{\Phi}_i$ ▷ Save region
    else
        Region is empty, do nothing
    end if
end for

5-3-1 FBR region construction

Given a set of N_{aff} unique affine functions, all possible arrangements are the permutations of N_{aff}. An arranged set is in fact a set of inqualities as it lists each function equal or smaller (or larger if in descending order) than the order: Suppose we have a ordered lattice set $\{\ell_1(x), \ell_3(x), \ell_2(x), \ell_4(x)\}$, in ascending order. Then the region in which this order is contained can be described by the inequalities $\ell_1 \leq \ell_3$, $\ell_1 \leq \ell_2$, $\ell_1 \leq \ell_4$, $\ell_3 \leq \ell_2$, $\ell_3 \leq \ell_4$, $\ell_2 \leq \ell_4$. The region would thus be given by the set of inequalities:

$$H := \begin{pmatrix} \ell_1 - \ell_3 \\ \ell_1 - \ell_2 \\ \ell_1 - \ell_4 \\ \ell_3 - \ell_2 \\ \ell_3 - \ell_4 \\ \ell_2 - \ell_4 \end{pmatrix} \leq \mathbf{0}$$

All possible arrangements of the affine functions are then given by the permutations of the vector $[1 \;\; 2 \;\; \cdots \;\; N_{\text{aff}}]$. Using the permutations described previously we would find all possible

arrangements which give an upper bound to the total number of FBR regions $N_{\hat{\Phi}}$. According to the definition of permutations the total number of arrangements would then be:

$$N_{\hat{\Phi}} = N_{\text{aff}}! \tag{5-17}$$

Let each arrangement γ_i of the set $\{1, 2, ..., N_{\text{aff}}\}$ be given by:

$$\gamma_i = P_{N_{\text{aff}}, N_{\text{aff}}, i} \tag{5-18}$$

With $P_{N_{\text{aff}}, N_{\text{aff}}, i}$ being the i-th permutation. For each permutation the region is constructed using the combinations of that particular arrangement. Consequently each region is described using a total of N_H hyperplanes, where:

$$N_H = \binom{N_{\text{aff}}}{2} = \frac{N_{\text{aff}}(N_{\text{aff}} - 1)}{2} \tag{5-19}$$

For a region $\hat{\Phi}_i$ given in central form as:

$$\hat{\Phi}_i : A_i \mathbf{x} \leq b_i \tag{5-20}$$

The matrix A_i and vector b_i for each arrangement γ_i are then:

$$A_i = \alpha_j - \alpha_k$$
$$b_i = \beta_k - \beta_j$$

with $A^{N_H \times n}$ and $b^{N_H \times 1}$, for all combinations of $j, k \in \{1, ..., N_{\hat{\Phi}}\}$, $i \neq j$ and $i \in \{1, ..., N_H\}$.

Empty sets

Again this method creates empty regions as the result of parallel and intersecting affine functions. Some of these empty or infeasible regions can be identified in their arrangements before constructing the regions and thus lowering the storage complexity as well as the time complexity of not having to solve a linear program.

5-4 Method 3 - FBR partition using domain cutting

The third method takes a more iterative approach compared to the previous two. The general idea is to start with the domain in which $f(\mathbf{x})$ is contained and cut the domain in half for each half-space that is the result of an intersection between two affine functions. In the next loop the now two regions are again cut with the hyperspaces as a result of the third affine function with the two previous affine functions. In other words, cutting the domain becomes increasingly complex where the domain is cut where the next affine function intersects with all previous affine functions, over every region. The complete algorithm is structured as follows in pseudocode:

Algorithm 3 Method 3 - FBR partition using domain cutting

```
Input: [ϕ, bounds]
Output: [Φ]
Initialize domain Φ (bounds)
for i = 1 : N_aff − 1 do
    c = ϕ(1 : i, :)                          ▷ All current i affine functions
    d = ϕ(i + 1, :)                          ▷ Next i + 1 affine function
    for j = 1 : i do
        for k = 1 : N_Φ do
            if hyper-space ∩ R then
                Φ_{i+1} = Φ_i^+,  Φ_{i+2} = Φ_i^−      ▷ Create partition
            else
                Otherwise jump to next loop
            end if
        end for
    end for
end for
```

5-4-1 FBR region construction

The algorithm loops over the affine functions and checks for each region and each new hyperplane if a cut is made. If it does create a cut the partition is updated and the outer most loop starts with the next affine function.

Consider the initial conditions. The domain consists of one region defined by the bounds on the the function:

$$\Phi_{init} := A_{bounds}\mathbf{x} \leq b_{bounds} \tag{5-21}$$

Then for the first pair of affine functions the hyperspace is configured:

$$H := \ell_1 - \ell_2 = \mathbf{0} \tag{5-22}$$

If H crosses the interior of Φ_{init}, i.e., $H \cap \Phi_{init} \neq \emptyset$ then Φ_{init} is partitioned into either side of the hyperspace H. The resulting two regions are then given by:

$$\Phi_1 := \Phi_{init} \cup H^+, \quad \Phi_2 := \Phi_{init} \cup H^- \tag{5-23}$$

In the following loops the next affine function is used to compute the hyper-spaces with all of the previous looped affine functions. Then each region in Φ is checked for each hyper-space if it is being cut. Thus for each new loop i we have the set of hyper-spaces:

$$H_i := \begin{pmatrix} \ell_1 - \ell_i \\ \ell_2 - \ell_i \\ \vdots \\ \ell_{i-1} - \ell_i \end{pmatrix} = \mathbf{0} \tag{5-24}$$

Which are checked for each region Φ_k where

$$\Phi = \bigcup_{k=1}^{v} \Phi_k \tag{5-25}$$

with $v \leq 2^{i-1}$. These are two nested loops inside loop i. Thus for each new hyperspace H_j (loop j) with $j = \{1, 2, ..., i\}$ each region Φ_k (loop k) with $k = \{1, 2, ..., N_\Phi\}$ is tested for being cut. Each time the region Φ_k is cut the domain is updated to:

$$\Phi_k = \Phi_k \cup H_j^+, \quad \Phi_{k+1} = \Phi_k \cup H_j^-, \tag{5-26}$$

Tests for being cut

During the partitioning each combination of the hyperspace H_j and region Φ_k is being tested for cutting. One obvious method is to test if either one of the sets $\Phi_k \cap H_j^+$ or $\Phi_k \cap H_j^-$ is empty, using the results of Farka's lemma and a linear program as described previously. However it is also possible to eliminate the need for a linear program when the hyper-space is identical to any of the hyper-planes that the region is composed of. Then only a simple test if H_j^+ or H_j^- equals any of the hyper-planes in region Φ_k is enough to determine if the cut is made or not.

5-5 Example of method 1, 2 & 3

Example 21. Let us consider the continuous PWA function from example 2:

$$f(\mathbf{x}) = \begin{cases} \ell_1(\mathbf{x}) = 8x_1 - 5x_2 - 1 & \mathbf{x} \in \Omega_1 \\ \ell_2(\mathbf{x}) = -5x_1 + 8x_2 - 1 & \mathbf{x} \in \Omega_2 \\ \ell_3(\mathbf{x}) = 0 & \mathbf{x} \in \Omega_3 \cup \Omega_4 \end{cases} \tag{5-27}$$

The parameter matrix ϕ and hyperplanes H are given by

$$\phi = \begin{pmatrix} 8 & -5 & 1 \\ -5 & 8 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad H := \begin{pmatrix} 13 & -13 \\ 8 & -5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

FBR region construction using hyperplane arrangements (Method 1) All combination of the hyperspaces creates the following regions:

$$\hat{\Phi}_1 : \begin{pmatrix} 13 & -13 \\ 8 & -5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \hat{\Phi}_2 : \begin{pmatrix} 13 & -13 \\ -8 & 5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}, \hat{\Phi}_3 : \begin{pmatrix} 13 & -13 \\ 8 & -5 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix},$$
$$\hat{\Phi}_4 : \begin{pmatrix} -13 & 13 \\ 8 & -5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \hat{\Phi}_5 : \begin{pmatrix} -13 & 13 \\ -8 & 5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}, \hat{\Phi}_6 : \begin{pmatrix} -13 & 13 \\ 8 & -5 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix},$$
$$\hat{\Phi}_7 : \begin{pmatrix} 13 & -13 \\ -8 & 5 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}, \hat{\Phi}_8 : \begin{pmatrix} -13 & 13 \\ -8 & 5 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}$$

FBR region construction using ordered sets (Method 2) We construct the ordered-matrix directly by considering each possible arrangement (in ascending order) that may occur within the domain:

$$\gamma = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 2 & 3 & 1 & 3 & 1 & 2 \\ 3 & 2 & 3 & 1 & 2 & 1 \end{pmatrix} \tag{5-28}$$

Each column represents a unique finest base region, which we then construct:

$$\hat{\Phi}_1 : \begin{pmatrix} \ell_1 - \ell_2 \\ \ell_1 - \ell_3 \\ \ell_2 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \hat{\Phi}_2 : \begin{pmatrix} \ell_1 - \ell_3 \\ \ell_1 - \ell_2 \\ \ell_3 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \hat{\Phi}_3 : \begin{pmatrix} \ell_2 - \ell_1 \\ \ell_2 - \ell_3 \\ \ell_1 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0},$$
$$\hat{\Phi}_4 : \begin{pmatrix} \ell_2 - \ell_3 \\ \ell_2 - \ell_1 \\ \ell_1 - \ell_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \hat{\Phi}_5 : \begin{pmatrix} \ell_3 - \ell_1 \\ \ell_3 - \ell_2 \\ \ell_1 - \ell_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}, \hat{\Phi}_6 : \begin{pmatrix} \ell_3 - \ell_2 \\ \ell_3 - \ell_1 \\ \ell_2 - \ell_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \mathbf{0}$$

Rewriting the above inequalities into *normal form* we get the regions:

$$\hat{\Phi}_1 : \begin{pmatrix} 13 & -13 \\ 8 & -5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \hat{\Phi}_2 : \begin{pmatrix} -8 & 5 \\ 13 & -13 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \hat{\Phi}_3 : \begin{pmatrix} -13 & 13 \\ -5 & 8 \\ 8 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix},$$
$$\hat{\Phi}_4 : \begin{pmatrix} -5 & 8 \\ -13 & 13 \\ -8 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \hat{\Phi}_5 : \begin{pmatrix} -8 & 5 \\ -5 & 8 \\ 13 & -13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}, \hat{\Phi}_6 : \begin{pmatrix} 5 & -8 \\ -8 & -5 \\ -13 & 13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}$$

FBR region construction using ordered sets (Method 3) Cutting the domain in a loop gives us the following FBR regions:

$$\Phi_1 : \begin{pmatrix} 13 & -13 \\ 8 & -5 \\ -5 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \Phi_2 : \begin{pmatrix} -8 & 5 \\ 13 & -13 \\ 5 & -8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \Phi_3 : \begin{pmatrix} -13 & 13 \\ -5 & 8 \\ 8 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix},$$
$$\Phi_4 : \begin{pmatrix} -5 & 8 \\ -13 & 13 \\ -8 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \Phi_5 : \begin{pmatrix} -8 & 5 \\ -5 & 8 \\ 13 & -13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}, \Phi_6 : \begin{pmatrix} 5 & -8 \\ -8 & -5 \\ -13 & 13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}$$

5-6 Analysis of the FBR algorithms

The three algorithms are analysed based on their storage requirements and time complexity. The storage requirements are mostly determined by the number of hyperplanes each (initialised) FBR region. In addition, the algorithms store each unique affine function. However, the number of hyperplanes quickly exceeds the number of unique affine functions as the problem becomes more complex and may be neglected in the analysis. The time complexity for the algorithm to finish the partitioning and have no redundant or non existing regions.

5-6-1 Storage requirements

Method 1 In the first method each possible FBR region is initialised using hyperplane arrangements. Initially for a set of N_{aff} unique affine functions the algorithm needs to store $N_{\text{aff}}(n+1)$ real numbers, inside the parameter matrix ϕ. For each initialised FBR region $\hat{\Phi}_i$ with $i \in \{1, ..., N_{\hat{\Phi}}\}$ a total number of N_H hyperplanes of dimension $(n-1)$ are stored, where N_H is given by 5-9 and the total number of initialised FBR regions $N_{\hat{\Phi}}$, given in 5-13. Then the storage requirements of algorithm 1 is:

$$\begin{aligned}\#\text{real numbers} &:= \underbrace{(n+1)N_{\text{aff}}}_{\phi} + \underbrace{(n+1)N_H N_{\hat{\Phi}}}_{\hat{\Phi}} \\ &= (n+1)N_{\text{aff}} + (n+1)\frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}2^{\frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}} \\ &= (n+1)N_{\text{aff}} + \frac{1}{2}(n+1)N_{\text{aff}}(N_{\text{aff}}-1)2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)}\end{aligned}$$

For larger problems with $N_{\text{aff}} > 5$ the first term may be neglected, giving:

$$\#\text{real numbers} \approx \frac{1}{2}(n+1)N_{\text{aff}}(N_{\text{aff}}-1)2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)} \tag{5-29}$$

After these regions are initialised, the algorithm filters out each empty region. When the algorithm is complete the upperbound for the storage requirements is determined by the total number of FBR regions:

$$\begin{aligned}\#\text{real numbers} &:= \underbrace{(n+1)N_{\text{aff}}}_{\phi} + \underbrace{(n+1)N_H N_{\hat{\Phi}}}_{\Phi} \\ &= (n+1)N_{\text{aff}} + (n+1)\frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}N_{\text{aff}}! \\ &\approx \frac{1}{2}(n+1)N_{\text{aff}}(N_{\text{aff}}-1)N_{\text{aff}}!\end{aligned}$$

Note that this results from replacing $N_{\hat{\Phi}}$ with N_{Φ}.

Method 2 The second method initialises each possible FBR region by considering the ordered lattices. Then the storage requirements of algorithm 2 is:

$$\begin{aligned}\#\text{real numbers} &:= \underbrace{(n+1)N_{\text{aff}}}_{\phi} + \underbrace{(n+1)N_H N_{\hat{\Phi}}}_{\hat{\Phi}} \\ &= (n+1)N_{\text{aff}} + (n+1)\frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}N_{\text{aff}}!\end{aligned}$$

And where for larger problems with $N_{\text{aff}} > 5$ the first term may be neglected, giving the storage requirements as:

$$\#\text{real numbers} :\approx \frac{1}{2}(n+1)N_{\text{aff}}(N_{\text{aff}}-1)N_{\text{aff}}! \tag{5-30}$$

After these regions are initialised, the algorithm filters out each empty region. The upper-bound of the storage requirements equal that of the initialised storage requirements. The lower-bound depends on the number of intersecting and parallel hyperplanes as. An exact answer to this question lies outside of the scope of this thesis.

Method 3 The approach of method 3 is to construct the regions by checking if newly added affine functions cause the domain to be partitioned. The regions are constructed using hyperplanes only when this partitioning happens. Its is expected that in general this leads to less hyperplanes than the the naive approach of building the regions using a combinatorics approach. Thus:

$$\begin{aligned}\#\text{real numbers} &\leq (n+1)N_{\text{aff}} + (n+1)N_H N_{\hat{\Phi}} \\ &= (n+1)N_{\text{aff}} + (n+1)\frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}N_{\text{aff}}!\end{aligned}$$

5-6-2 Time complexity

Method 1 The algorithm checks every initially created set, or $\hat{\Phi}$, for its existence or emptiness by solving a linear program. The time complexity can thus be expressed by the total number of linear programs that need to be solved by the algorithm or:

$$\#\text{linear programs} := \#\text{initialized sets} \tag{5-31}$$

The number of initialized sets $N_{\hat{\Phi}}$ can be expressed as the unique number of affine function:

$$\begin{aligned}N_{\hat{\Phi}} &= 2^{N_H} \\ &= 2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)}\end{aligned}$$

Therefor a worse case total of $2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)}$ linear programs need to be solved. The worse-case time complexity of the first method is thus:

$$\mathcal{O}\Big(\cdot\, 2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)}L\Big) \tag{5-32}$$

with N_{aff} the distinct number of affine functions, and in which L is the bit length of the input data of the LP problem.

Method 2 In method 2, the algorithm checks every initially created set, or $\hat{\Phi}$, for its existence or emptiness by solving a linear program (based on one of the methods described in previously). The time complexity can thus be expressed by the total number of linear programs that need to be solved by the algorithm or:

$$\#\text{linear programs} := \#\text{initialized sets}$$

The number of initialized sets $N_{\hat{\Phi}}$ can be expressed in the unique number of affine function. The worst case scenario for this methods, the number of initialised regions is equal to the upperbound of finest base regions. The algorithm checks every region for emptiness. Therefor a worse case total of $N_{\text{aff}}!$ linear programs need to be solved. The worse-case time complexity then is given by:

$$\mathcal{O}(LN_{\text{aff}}!) \tag{5-33}$$

Method 3 The algorithm has three nested loops i, j and k, where $i = \{1, 2, ..., N_{\text{aff}} - 1\}$, $j = \{1, 2, ..., i\}$ and $k = \{1, 2, ..., N_{\Phi}\}$. The upperbound for N_{Φ} is:

$$N_{\Phi} = \frac{(i-1)(i-2)}{2} \tag{5-34}$$

The worst-case total number of loops is thus:

$$\mathcal{O}\Big(L \sum_{i=1}^{N_{\text{aff}}} \frac{i}{2}(i-1)(i-2)\Big) \tag{5-35}$$

And thus a worst-case total of $\sum_{i=1}^{N_{\text{aff}}} \frac{i}{2}(i-1)(i-2)$ linear programs need to be solved. Any hyper-planes that coincide with one another however can be removed from the evaluation.

5-7 Results - Performance comparison

To benchmark the different algorithm fairly sets of increasingly complex hyperplanes are constructed using randomly generated sets. The algorithms are tested in $\mathbb{R}^n$ for $n = 3, 4, 5, 6, 7$, for an increasing number of affine functions. When the runtime of a particular method exceeded $120s$, the operation was cancelled as well as any subsequent tests. To validate each region the chebyshev method was used. Further more in the first experiment the algorithms use only the chebyshev method to filter out empty sets. In the second experiment the "optimized" algorithms are used as described in the sections previously.

The experiments where conducted with Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz. The code was written in matlab and is attached to the appendix. For the measurement of the cpu time the build in matlab tool "tic toc" was used.

	$\mathbb{R}^3$			$\mathbb{R}^4$			$\mathbb{R}^5$		
Alg	N_{aff}	N_Φ	cpu (s)	N_{aff}	N_Φ	cpu (s)	N_{aff}	N_Φ	cpu (s)
M1			0.046493			0.036888			0.011701
M2	3	6	0.022745	3	6	0.024951	3	6	0.013951
M3			0.027385			0.041231			0.033310
M1			0.141130			0.125962			0.059710
M2	4	18	0.023852	4	24	0.023485	4	24	0.023561
M3			0.072739			0.081061			0.087351
M1			1.078261			1.123901			0.969331
M2	5	46	0.127073	5	96	0.118510	5	120	0.126312
M3			0.268093			0.483023			0.539911
M1			33.523213			38.133172			34.328601
M2	6	101	0.728852	6	326	0.824096	6	594	0.761949
M3			0.852704			2.689630			3.619389
M1			$\sim$			$\sim$			$\sim$
M2	7	192	5.437076	7	932	5.704437	7	2532	6.594432
M3			2.554187			9.922502			25.189844
M1			$\sim$			$\sim$			
M2	8	304	42.571758	8	2297	45.642199			
M3			5.384373			35.972784			
M1			$\sim$			$\sim$			
M2	9	536	$\sim$	9	5111	$\sim$			
M3			13.167			117.095308			
M1			$\sim$						
M2	10	901	$\sim$						
M3			29.493						
M1			$\sim$						
M2	11	1341	$\sim$						
M3			49.661						

Table 5-1: CPU time for the finest base region partition for different configuration. Processor: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz.

Chapter 6

Discussion

In this chapter the FBR regions as a method for the realisation of continuous PWA functions and canonical MMPS functions is discussed. This includes discussion of the FBR regions themselves, the parameter matrix, the ordered lattices and the three algorithms for the computation.

6-1 Realisation of canonical MMPS functions and continuous PWA functions

FBR regions can be computed from the parameter matrix only. Thus only information of the unique affine functions is necessary. Within an FBR regions the ordering of the affine functions is fixed and therefor the ordered lattice can describe a FBR region. The upperbound of FBR regions that are computed are determined by the total number of unique affine functions. This total number of FBR regions N_Φ is given by $N_\Phi = \sum_{i=0}^{n} \binom{N_{\text{aff}}}{i}$ where n is the dimension, N_{aff} the total number of unique affine functions and $\binom{N_{\text{aff}}}{n}$ the combinations. The partitioning is thus highly complex as a result of the combinatorial complexity or *combinatorial explosion*. To put this into perspective: for a problem in $\mathbb{R}^2$ with $N_{\text{aff}} = 5$ the number of FBR regions is $N_\Phi = 10$. For a problem with $N_{\text{aff}} = 20$ the number of FBR regions is $N_\Phi = 211$.

Parameter matrix The parameter matrix can be easily obtained from continuous PWA functions and conjunctive MMPS functions. This is because the unique affine functions are embedded in the parameters of both forms. The operations addition and scalar multiplication are not present in the expression which do not change any of the parameters.

Kripfganz MMPS functions and general MMPS functions do use the operations addition and scalar multiplication (the latter only for general MMPS functions) in the expression and therefor information of the unique affine functions is not directly present in the expression. By using combinatorics of the expression and parameters, the parameter matrix can be obtained. Likewise, both can also be rewritten into conjunctive form from which the unique affine

functions can be obtained. This procedure however can produce redundant affine functions, i.e. affine functions that may never be active.

Redundancy in the parameter matrix will produce a larger number of Finest Base Region (FBR) regions. These regions can still be used for the realisation of the different forms. However the number of regions explodes with the number of affine functions and therefor it can be necessary to remove the redundancy.

6-2 FBR partitioning algorithms

Three algorithms have been proposed for the computation of the FBR regions. Each method uses the parameter matrix as only input. Finest base regions are the result of each affine element intersecting with one another. The relation between the number of affine function and the number of finest base regions is therefor exponential in nature. This simple result already reveals the computational burden of realising the partition being heavy to say the least. Accepting the fact that the partition will already be exponential in nature, the best the algorithm can do is be as close to the lower bound of computed regions as well as describing the regions using the least amount of hyperplanes.

Method 1 uses the arrangements of all possible hyperplanes to construct each region. The hyperspaces are first initialized by considering all pairs of affine functions. Then each region is constructed using all possible combinations of the hyperplanes. Thus a total of $\hat{N_\Phi} = 2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)}$ regions are initialised, each described by $N_{\text{H},\Phi_i} = \frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}$ hyperplanes. This method produces more than the upperbound of possible FBR regions. The first method thus creates a total of at least:

$$\#\text{ empty sets} \geq 2^{\frac{1}{2}N_{\text{aff}}(N_{\text{aff}}-1)} - N_{\text{aff}}! \tag{6-1}$$

Method 2 exploits the ordered lattices for the computation of FBR regions. Each region is initialised using the possible arrangements of the affine functions, rather than the hyperplanes. The number of initialised empty sets is determined only by the properties of the affine functions that may have intersecting and parallel hyperplanes. The ordered lattices are each permutation of the affine functions. Therefor a total of $\hat{N_\Phi} = N_{\text{aff}}!$ regions are initialised, each described by $N_{\text{H},\Phi_i} = \frac{N_{\text{aff}}(N_{\text{aff}}-1)}{2}$ hyperplanes.

Method 3 uses an iterative approach, first initializing the domain and making necessary cuts for each new hyperplane, thus relaxes the issue of initialising a (very) large number of regions. Furthermore each region is constructed with less hyperplanes as they are only added to the region of a cut is made.

In both method 1 and method 2 the regions are described using a fixed number of hyperplanes: $N_H = \frac{1}{2}N_{\text{aff}}(N_{\text{aff}} - 1)$. However method three only adds necessary hyperplanes to make the cut. The resulting regions are not in a irredundant description, however it will lower the number of hyperplanes used to describe the FBR region.

Chapter 7

Conclusion

The FBR region partitioning is a method for realising continuous PWA functions and canonical MMPS functions from either continuous PWA functions, conjunctive MMPS functions, kripfganz MMPS functions or general MMPS functions. The only requirement is knowledge of the unique affine functions which can be found for each of the cases. The FBR partitioning can be used for all problems in $\mathbb{R}^n$ with $n > 1$, and for any type of function to any type of function.

At the point of writing this thesis and to the best knowledge of the author, the FBR method is the first known strategy to transform any type of MMPS function into a continuous PWA function. The FBR method is also the only known "universal" strategy that can be deployed for the transformation between any two type of functions.

The practicality of the FBR method is limited as a result of the combinatorial nature of the strategy. Combinatorial explosion happens by an increasing number of affine function and/or dimension increase. Three algorithms have been proposed to perform the partition. Method 1 is based on hyperplane arrangements, or all the possible combinations of the hyperplanes. Method 2 uses all the combinations of the affine functions instead. Finally method 3 cuts the domain for each added hyperplane. Method 3 performs the best of all three methods.

Chapter 8

Summary & recommendations for future research

MMPS functions and continuous PWA functions are equivalent. While both functions span a variety of fields and applications, each has their own advantage. MMPS functions can be seen as an analytical description for continuous PWA functions, providing a much more efficient tool for the evaluation of the function. However knowledge of specific regions is present in continuous PWA functions and lost in MMPS functions. It is therefor of great value when one can transform one form into another a tap into the benefits of the form for specific applications.

There are multiple methods for transforming a continuous PWA into a (canonical) MMPS function available in existing literature. However the transformation of MMPS functions into continuous PWA function there is not. The FBR partitioning fills in the gaps in this research field.

The FBR region partitioning is a method for realising continuous PWA functions and canonical MMPS functions from either continuous PWA functions, conjunctive MMPS functions, kripfganz MMPS functions or general MMPS functions. The only requirement is knowledge of the unique affine functions which can be found for each of the cases. The FBR partitioning can be used for all problems in $\mathbb{R}^n$ with $n > 1$, and for any type of function to any type of function.

At the point of writing this thesis and to the best knowledge of the author, the FBR method is the first known strategy to transform any type of MMPS function into a continuous PWA function. The FBR method is also the only known "universal" strategy that can be deployed for the transformation between any two type of functions.

The practicality of the FBR method is limited as a result of the combinatorial nature of the strategy. Combinatorial explosion happens by an increasing number of affine function and/or dimension increase. Three algorithms have been proposed to perform the partition. Method 1 is based on hyperplane arrangements, or all the possible combinations of the hyperplanes. Method 2 uses all the combinations of the affine functions instead. Finally method 3 cuts the domain for each added hyperplane. Method 3 performs the best of all three methods.

8-1 Topics for future research

- **Efficient algorithm** The FBR partitioning is only feasible for smaller problems. For online transformations the FBR method is simply too complex. For offline transformations the FBR method becomes in feasible for problems $N_{\text{aff}} > 12$ or $n > 6$.

  Scenarios with unique affine functions exceeding the hundreds, or systems with many states are not uncommon, where the FBR method would not be a feasible solution. Given the benefits of the transformations is would be of great value if much more capable method would exist for the realisation of continuous PWA functions from MMPS functions.

- **Research on ordered lattices** The order of the affine functions uniquely define a FBR region. It has been shown that considering the ordered lattice sets, it is possible to directly transform to conjunctive MMPS functions, find adjacent regions, find convex folds and transform into polyhedral regions.

  It is possible the ordered lattices can be further exploited. It could be further researched in relation with the kripfganz MMPS functions, and how the convex decomposition performs with the FBR partitioning. It is also possible to only consider infeasible sets. As the ordered lattice sets are simply permutations, there could be the possibility of describing the function as the sets that do not exist as these are likely much smaller than the number of existing regions. These sets could be established based on parallel affine functions for example, where only one specific order between two functions exist.

  The active function can also be easily retrieved from the ordered lattices if the original function is in conjunctive MMPS form. It would be interesting to see if it is possible to find the active function in the cases of kripfganz and general MMPS function, from the ordered lattice sets. This could then be compared with the strategy of computing an interior point and evaluating the function at this point.

- **Improvement of the FBR algorithms** The three FBR algorithms are first idea's of computing the FBR regions. The algorithms could possible be improved, or completely different strategies may be used. Improvement could be made in the runtime, the total number of initialised regions and the total number of hyperplane each region consists of.

- **Parameter matrix redundancy** The redundancy in the parameter matrix is unwanted, especially given the combinatoric nature of the algorithms. Redundancy is clearly visible for continuous PWA functions, but not necessarily for MMPS functions. This thesis has briefly touched on the methods for removing redundancy, but this has been far from thorough. It would be of great interest, both for the FBR regions, as well as MMPS functions in general if we have the tools to remove the redundancy and have knowledge of the unique affine functions.

- **Expanding the MMPS toolbox** A first attempt has been made to create an MMPS toolbox with the idea to increase productivity while working with MMPS functions and to combine current tools in one package. The toolbox still misses plenty of tools, and the current code can be made much more accessible and fool proof.

Appendix A

MMPS algebraic rules

A-1 Algebraic rules

Consider this sheet as a helpful reminder of the properties of the maximization and minimization operators. These rules may be considered common knowledge, but taken from [7]. Given the arguments $\alpha, \beta, \gamma, \delta$ and $\rho \in \mathbb{R}$ and $\rho \geq 0$, then the following holds:

- Relation between operators maximization and minimization:

$$\max(\alpha, \beta) = -\min(-\alpha, -\beta) \tag{A-1}$$
$$\min(\alpha, \beta) = -\max(-\alpha, -\beta) \tag{A-2}$$

- Properties of addition (addition is distributive with respect to minimization and maximization):

$$\min(\alpha, \beta) + \min(\gamma, \delta) = \min(\alpha + \gamma, \alpha + \delta, \beta + \gamma, \beta + \delta) \tag{A-3}$$
$$\max(\alpha, \beta) + \max(\gamma, \delta) = \max(\alpha + \gamma, \alpha + \delta, \beta + \gamma, \beta + \delta) \tag{A-4}$$

- Properties of scaling

$$\rho \max(\alpha, \beta) = \max(\rho\alpha, \rho\beta) \tag{A-5}$$
$$\rho \min(\alpha, \beta) = \min(\rho\alpha, \rho\beta) \tag{A-6}$$

- Since minimization is distributive with respect to maximization and vice versa it follows:

$$\min(\max(\alpha, \beta), \max(\gamma, \delta)) = \max(\min(\alpha, \gamma), \min(\alpha, \delta), \min(\beta, \gamma), \min(\beta, \delta)) \tag{A-7}$$
$$\max(\min(\alpha, \beta), \min(\gamma, \delta)) = \min(\max(\alpha, \gamma), \max(\alpha, \delta), \max(\beta, \gamma), \max(\beta, \delta)) \tag{A-8}$$

A-2 Conjunctive rewriting

[7, 29] Consider the two affine functions f_k and f_l, then the functions that result from applying the basic constructors of an MMPS function (a list with properties of the max, min, + and scaling operations in the MMPS framework are provided in Appendix A.) are in conjunctive MMPS form. Then a recursive argument can be used that consists in showing that if the basic constructors of an MMPS function are applied to two (or more) MMPS functions in min-max MMPS form, then the result can again be transformed into min-max MMPS form.

Let two MMPS functions f and g be in min-max canonical form : $f = \min(\max(f_1, f_2)\max(f_3, f_4))$ and $g = \min(\max(g_1, g_2)\max(g_3, g_4))$. Now it can be shown that $\max(f, g)$, $\min(f, g)$, $f + g$ and βf with $\beta \in \mathbb{R}$ can again be written in min-max MMPS form.

Maximization

$$\begin{aligned}
\max(f,g) &= \max[\min(\max(f_1,f_2),\max(f_3,f_4)),\min(\max(g_1,g_2),\max(g_3,g_4))] \\
&= \max[\max(\min(f_1,f_3),\min(f_1,f_4),\min(f_2,f_3),\min(f_2,f_4), \\
&\qquad \max(\min(g_1,g_3),\min(g_1,g_4),\min(g_2,g_3),\min(g_2,g_4)] \\
&= \max(\min(f_1,f_3),\min(f_1,f_4),\min(f_2,f_3),\min(f_2,f_4), \\
&\qquad \min(g_1,g_3),\min(g_1,g_4),\min(g_2,g_3),\min(g_2,g_4)) \\
&= \min(\max(f_1,f_1,f_2,f_2,g_1,g_1,g_2,g_2),\max(f_1,f_1,f_2,f_2,g_1,g_1,g_2,g_4),... \\
&\qquad \max(f_3,f_4,f_3,f_4,g_3,g_4,g_3,g_4))
\end{aligned}$$

Minimization

$$\begin{aligned}
\min(f,g) &= \min[\min(\max(f_1,f_2),\max(f_3,f_4)),\min(\max(g_1,g_2),\max(g_3,g_4))] \\
&= \min(\max(f_1,f_2),\max(f_3,f_4),\max(g_1,g_2),\max(g_3,g_4))
\end{aligned}$$

Addition

$$\begin{aligned}
f+g &= \min(\max(f_1,f_2),\max(f_3,f_4)) + \min(\max(g_1,g_2),\max(g_3,g_4)) \\
&= \min(\max(f_1,f_2)+\max(g_1,g_2)),\max(f_1,f_2)+\max(g_3,g_4), \\
&\qquad \max(f_3,f_4)+\max(g_1,g_2)),\max(f_3,f_4)+\max(g_3,g_4)) \\
&= \min(\max(f_1+g_1,f_1+g_2,f_2+g_1,f_2+g_2),\max(f_1+g_3,f_1+g_4,f_2+g_3,f_2+g_4) \\
&\qquad \max(f_3+g_1,f_4+g_1,f_4+g_1,f_4+g_2),\max(f_3+g_3,f_3+g_4,f_4+g_3,f_4+g_4))
\end{aligned}$$

Scalar multiplication

$$\begin{aligned}
\beta f &= \beta \min(\max(f_1, f_2), \max(f_3, f_4)) \\
&= \min(\max(\beta f_1, \beta f_2), \max(\beta f_3, \beta f_4)) \quad \text{if} \beta \leq 0 \\
&= -|\beta| \min(\max(f_1, f_2), \max(f_3, f_4)) \\
&= -\min(\max(|\beta| f_1, |\beta| f_2), \max(|\beta| f_3, |\beta| f_4)) \\
&= \max(-\max(|\beta| f_1, |\beta| f_2), -\max(|\beta| f_3, |\beta| f_4)) \\
&= \max(\min(-|\beta| f_1, -|\beta| f_2), \min(-|\beta| f_3, -|\beta| f_4)) \\
&= \max(\min(\beta f_1, \beta f_2), \min(\beta f_3, \beta f_4)) \\
&= \min(\max(\beta f_1, \beta f_3), \max(\beta f_1, \beta f_4), \max(\beta f_2, \beta f_3), \max(\beta f_2, \beta f_4))
\end{aligned}$$

Concluding that the conjunctive MMPS form is indeed a canonical form within the MMPS framework.

Appendix B

Convex Polytopes

B-1 Theory on Polyhedra

B-1-1 Polyhedral sets

Let us first establish the following definitions[16]:

Definition B-1.1. [16] (Convex set): A set $\mathcal{S} \subseteq \mathbb{R}^n$ is convex if the line segment connecting any pair of points of $\mathcal{S}$ lies entirely in $\mathcal{S}$, i.e. if for any $s_1, s_2 \in \mathcal{S}$ and any α with $0 \leq \alpha \leq 1$ we have $\alpha s_1 + (1-\alpha)s_2 \in \mathcal{S}$

Definition B-1.2. [16] (Polyhedron): A polyhedron is a convex set given as the intersection of a finite number of hyper-planes and half-spaces or as a convex combination of a finite number of vertices and rays.

Definition B-1.3. [16] (Polytope): A polytope is a bounded polyhedron.

From the above definitions it is clear that every polytope represents a convex, compact (i.e., bounded and closed) set. A polytope $\mathcal{P} \subset \mathbb{R}^n$, $\mathcal{P} = \{x \in \mathbb{R}^n | P^x x \leq P^c\}$ is said to be *full dimensional* if $\forall x \in \mathbb{R}^n : P^x x < P^c$. Furthermore, if $||(P^x)_i|| = 1$, where $(P^x)_i$ denotes i-th row of a matrix P^x, we say that the polytope $\mathcal{P}$ is *normalized*.

Definition B-1.4. [16] (H-representation): The polyhedron $\mathcal{P}$ is formed by the intersection of m inequalities and m_e equalities, i.e.

$$\mathcal{P} = \{x \in \mathbb{R}^n | Ax \leq b, A_e x = b_e\} \tag{B-1}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathcal{R}^m$, $A_e \in \mathbb{R}^{m_e \times n}$, $b_e \in \mathcal{R}^{m_e}$ are the data representing the halfspaces and hyperplanes respectively.

Definition B-1.5. [16] (V-representation): The polyhedron $\mathcal{P}$ is formed by a convex combination of n_v vertices and n_r rays, i.e.

$$\mathcal{P} = \{x \in \mathbb{R}^n | x = \lambda^{\mathrm{T}} V + \gamma^{\mathrm{T}} R, \gamma \geq 0, 1^{\mathrm{T}} \lambda = 1\} \tag{B-2}$$

where $V \in \mathbb{R}^{n \times n_e}$, $R \in \mathbb{R}^{n \times n_r}$ represent vertices and rays respectively.

Definition B-1.6. [16] (face): The linear inequality $a'x \leq b$ is called *valid* for a polyhedron $\mathscr{P}$ if $a'x \leq b$ holds for all $x \in \mathscr{P}$. A subset of a polyhedron is called a *face* of $\mathscr{P}$ if it is represented as

$$\mathscr{F} = \mathscr{P} \cap \{x \in \mathbb{R}^n | a'x = b\} \tag{B-3}$$

for some valid inequality $a'x \leq b$. The faces of polyhedron $\mathscr{P}$ of dimension 0,1, $(n-2)$ and $(n-1)$ are called vertices, edges, ridges and facets respectively.

A polytope $\mathscr{P} \subset \mathbb{R}^n$, $\mathscr{P} = \{x \in \mathbb{R}^n | P^x x \leq P^c\}$ is in a *minimal representation* if a removal of any of the rows in $P^x x \leq P^c$ would change it.

B-2 Hyperplane arrangements

The topic of space divisions is well studied. For the use of the algorithm we are interested in the number of regions that the partitioning will result in. First let the hyper-plane arrangement be defined as described in [25]:

A *finite hyperplane arrangement* $\mathscr{A}$ is a finite set of affine hyperplanes in some vector space $V = K^n$, where K is a field. Generally we take $K = \mathbb{R}^n$. First define a linear hyperplane to be an $(n-1)$-dimensional subspace H of V, i.e.,

$$H = \{v \in V : \alpha \cdot v = 0\} \tag{B-4}$$

where α is a fixed nonzero vector in V and $\alpha \cdot v$ is the usual dot product:

$$(\alpha_i, ..., \alpha_n) \cdot (v_i, ..., v_n) = \sum \alpha_i v_i \tag{B-5}$$

an *affine hyperplane* is a translate J of a linear hyperplane, i.e.

$$J = \{v \in V : \alpha \cdot v = a\} \tag{B-6}$$

where α is a fixed nonzero vector in V and $a \in K$.

Appendix C

Algorithms

fbr1

Purpose

Creates finest base regions for a set of affine functions

Synopsis

$[\Phi] = \text{fbr1}(\phi,\Omega)$

Description

The function creates finest base regions from a set of affine functions. A finest base region is defined such that no intersection between any two affine functions (from the input) lies within the interior of the finest base region.

The algorithm uses combinatorics of the hyperspaces to construct each region in H-representation. Empty and infeasible sets that are generated in this step are identified and removed using a linear program.

See Also

Matlab implementation

```
1 function [Phi] = fbr1(phi,Omega)
2     % initialize
3     Phi = [];
4
```

```
5       % sizes
6       N_aff = height(phi);        % Number of distinct affine
            functions
7       N_H = N_aff*(N_aff-1)/2;    % Number of hyper spaces
8
9       % Hyperspaces
10      in = nchoosek(1:1:N_aff,2);
11      H = [phi(in(:,1),1:end-1)-phi(in(:,2),1:end-1) phi(in(:,2),
            end)-phi(in(:,1),end)];
12
13      % 2^N_H Phi_hat regions of size
14      k = dec2bin(0:2^N_H-1)-'0' + dec2bin(0:2^N_H-1)-'0' - 1;
15      k = k';
16
17      % Loop to collect all non empty polyhedra
18      for i = 1:2^N_H
19          Ab = H.*k(:,i);
20          Phi_hat = Polyhedron('A',[Ab(:,1:end-1);Omega(:,1:end-1)
                ],'b',[Ab(:,end);Omega(:,end)]);
21
22          %% (Step 1.4, Region Filtering) - Remove all empty
                regions
23          tf = isEmpty(Phi_hat);
24          % Conditional to filter out epmty polyhedra:
25          % If chebycenter radius equals zero (or Inf somehow?)
26          % we know the polyhedral is empty
27          if tf==0
28              Phi = [Phi;Phi_hat];
29          else
30
31          end
32      end
33  end
```

fbr2

Purpose

Creates finest base regions for a set of affine functions

Synopsis

$[\Phi] = \text{fbr2}(\phi,\Omega)$

Description

The function creates finest base regions from a set of affine functions. A finest base region is defined such that no intersection between any two affine functions (from the input) lies within the interior of the finest base region.

The algorithm uses the permutations of the affine functions to initialise the regions. Empty and infeasible regions that are generated in this step are identified and removed using a linear program.

See Also

Matlab implementation

```
function [Phi] = fbr2(phi,Omega)
    % initialize
    Phi = [];

    % sizes
    N_aff = height(phi);

    % all orders
    I = perms(1:1:N_aff);

    % Loop to collect all non empty polyhedra
    for i = 1:factorial(N_aff)
        % Get hyperplanes from the ordered matrix column
        H = nchoosek(I(i,:),2);
        A = phi(H(:,1),:);
        B = phi(H(:,2),:);
        % Create region from hyperplanes
        Ab = [A(:,1:end-1)-B(:,1:end-1) B(:,end)-A(:,end)];
        % Create MPT polyhedron object
        Phi_hat = Polyhedron('A',[Ab(:,1:end-1);Omega(:,1:end-1)
           ],'b',[Ab(:,end);Omega(:,end)]);
```

```
21
22          tf = isEmpty(Phi_hat);
23          % Conditional to filter out epmty polyhedra:
24          % If chebycenter radius equals zero (or Inf somehow?)
25          % we know the polyhedral is empty
26          if tf==0
27              Phi = [Phi;Phi_hat];
28          else
29
30          end
31      end
32  end
```

fbr3

Purpose

Creates finest base regions for a set of affine functions

Synopsis

$[\Phi] = \text{fbr3}(\phi,\Omega)$

Description

The function creates finest base regions from a set of affine functions. A finest base region is defined such that no intersection between any two affine functions (from the input) lies within the interior of the finest base region. The algorithm uses combinatorics of the hyperspaces to construct each region in H-representation. Empty and infeasible sets that are generated in this step are identified and removed using a linear program.

See Also

Matlab implementation

```
1  function [Phi] = fbr3(phi,Omega)
2
3      N_aff = height(phi);
4
5      % Initialize domain
6      Phi = Polyhedron('A',Omega(:,1:end-1),'b',Omega(:,end));
7
8      % Start loop (i) - Loops over all affine functions
9      for i = 1:N_aff-1
10         % (Step 1): collect all hyperplanes as a result from all
               intersections with new function
11         c = phi(1:i,:);
12         d = phi(i+1,:);
13
14         % Create all new hyperplanes (less than)
15         A_l = c(:,1:end-1) - d(:,1:end-1);
16         b_l = d(:,end) - c(:,end);
17
18         % Create all new hyperplane (greater than)
19         A_r = -A_l;
20         b_r = -b_l;
21
```

```
22          % Step 2: For each new hyperplane loop the region it
                lies on
23          for j = 1:height(c)
24              % for the (j)th intersection loop over every (k)
                    region to find the
25              % region it is contained
26
27              % (T) is a temporary variable to store the new
                    partitions. At the
28              % end of the (k) loop (and next (j)loop) R is set
                    equal to T,
29              % and T is reset.
30              T = [];
31
32              % (H) is a temporary variable to copy over all
                    hyperplanes present
33              % in (R)
34              H = [];
35              for jj = 1:length(Phi)
36                  H = [H;Phi(jj).A Phi(jj).b];
37              end
38
39              %%% Check if hyperplane lies on any boundary of any
                    of the regions. Stop (j)loop if it is.
40              tfhl = sum(find(ismember(H,[A_l(j,:) b_l(j,:)],"rows
                    ")==1));
41              tfhr = sum(find(ismember(H,[A_r(j,:) b_r(j,:)],"rows
                    ")==1));
42              if tfhl>0 || tfhr>0
43                  % DONT make cut, cut has already been made
44                  % disp("Already made a cut!")
45              else
46                  % Make cut, for every region
47                  for k = 1:length(Phi)
48                      % Get current region
49                      Ab = [Phi(k).A Phi(k).b];
50
51                      % Create new region with cut and check if it
                            lies within the region
52                      P_l = Polyhedron('A',[Phi(k).A; A_l(j,:)],'b
                            ',[Phi(k).b;b_l(j,:)]);
53                      P_r = Polyhedron('A',[Phi(k).A; A_r(j,:)],'b
                            ',[Phi(k).b;b_r(j,:)]);
54
55                      % Check if it cuts the region (same as
                            checking if either of the two results in
                            an empty set)
```

```
56                      tfl = isEmpty(P_l);
57                      tfr = isEmpty(P_r);
58                      if tfl ~= 1 && tfr ~= 1
59                          % Region k is being cut. Remove region j
                                and add two new ones
60                          T = [T;P_l];
61                          T = [T;P_r];
62                      else
63                          T = [T;Phi(k)];
64                      end
65                  end
66                  Phi = T;
67              end
68          end
69      end
70  end
```

getAdjacent

Purpose

Finds any pair of adjacent regions

Synopsis

[ad] = getAdjacent(U)

Description

This algorithm loops over all regions of a continuous PWA function U. Two regions are adjacent when the two regions meet at an hyperplane.

See Also

getConvex

Matlab implementation

```
function [ad]=getAdjacent(U)
    N = nchoosek(1:1:U.Num,2);
    ad = {};
    for i=1:height(N)
        if U.Set(N(i,1)).isAdjacent(U.Set(N(i,2)))
            j = height(ad)+1;
            ad{j,1} = U.Set(N(i,1));
            ad{j,2} = U.Set(N(i,2));
        end
    end
end
```

getConvex

Purpose

Checks any two regions that are adjacent if they are convex or concave

Synopsis

[cv] = getConvex(adj)

Description

Checks any two regions that are adjacent if they are convex or concave

See Also

getAdjacent

Matlab implementation

```
function [cv]=getConvex(ad)
    N_ad = height(ad);
    cv = {};
    for i = 1:N_ad
        x1cc = ad{i,1}.chebyCenter.x;
        x2cc = ad{i,2}.chebyCenter.x;

        fx11 = ad{i,1}.getFunction('primal').F*x1cc+ ad{i,1}.
           getFunction('primal').g;
        fx12 = ad{i,1}.getFunction('primal').F*x2cc+ ad{i,1}.
           getFunction('primal').g;

        fx21 = ad{i,2}.getFunction('primal').F*x1cc + ad{i,2}.
           getFunction('primal').g;
        fx22 = ad{i,2}.getFunction('primal').F*x2cc + ad{i,2}.
           getFunction('primal').g;
        % Check slopes

        if fx11>=fx21 && fx22>=fx12
            % Yes its convex
            j = height(cv)+1;
            cv{j,1} = ad{i,1};
            cv{j,2} = ad{i,2};
        end
```

```
21      end
22  end
```

cpwa2mmpskg

Purpose

Computes the kripfganz MMPS function of a continuous PWA function

Synopsis

[mmpskg] = cpwa2mmpskg(U)

Description

A continuous PWA function can be decomposed into two convex functions, who's difference is again the original function. The algorithm follows the procedure from [19] where the first convex part is constructed by summing all the convex folds. The second convex part is constructed by subtracting the original function from the convex function.

See Also

cpwa2mmpscd

Matlab implementation

```
1  function [g,h] = cpwa2mmpskg(U)
2      % Get all pairs of adjacent matrixes
3      ad = getAdjacent(U);
4      % Check each set if it is convex
5      cv = getConvex(ad);
6      % Build G
7      % Partition
8      % Initialize domain
9      G = U.convexHull;
10     % get all hyperplanes
11     A = []; b = []; T = [];
12     for i = 1:height(folds)
13         A(i,:) = folds{i,1}.getFunction('primal').F-folds{i,2}.
               getFunction('primal').F;
14         b(i,:) = folds{i,2}.getFunction('primal').g-folds{i,1}.
               getFunction('primal').g;
15
16         for j = 1:length(G)
17             % Create new region with cut and check if it lies
                   within the region
18             P_l = Polyhedron('A',[G(j).A; A(i,:)],'b',[G(j).b; b
                   (i,:)]);
```

```
19              P_r = Polyhedron('A',[G(j).A; -A(i,:)],'b',[G(j).b;
                    -b(i,:)]);
20
21              % Check if it cuts the region (same as checking if
                    either of the two results in an empty set)
22              tfl = isEmptyCheby(P_l);
23              tfr = isEmptyCheby(P_r);
24              if tfl ~= 1 && tfr ~= 1
25                  % Region k is being cut. Remove region j and add
                        two new ones
26                  T = [T; P_l];
27                  T = [T; P_r];
28              else
29                  T = [T; G(j)];
30              end
31          end
32          G = T;
33          T = [];
34      end
35
36
37      % Functions
38      for i=1:height(G)
39          cc = G(i).chebyCenter.x;
40          for j=1:length(folds)
41              fun(1,1) = folds{j,1}.getFunction('primal').F*cc +
                    folds{j,1}.getFunction('primal').g;
42              fun(1,2) = folds{j,2}.getFunction('primal').F*cc +
                    folds{j,2}.getFunction('primal').g;
43              [~,I(j)] = max(fun);
44              a(j,:) = [folds{j,I(j)}.getFunction('primal').F
                    folds{j,I(j)}.getFunction('primal').g];
45          end
46          g(i,:) = sum(a);
47          f(i) = AffFunction(g(i,1:end-1), g(i,end));
48          G(i).addFunction(f(i),'primal');
49      end
50      G = PolyUnion('Set', G, 'Bounded', true, 'Connected', true,
            'Overlaps', false, 'Convex', true);
51
52      % Build H
53      % Partition
54      % Initialize domain
55      H = U.convexHull;
56
57      % get all hyperplanes
58      Ab = [];
```

```
59      for i = 1:G.Num
60          Ab = [Ab;G.Set(i).H];
61      end
62
63      for i = 1:U.Num
64          Ab = [Ab;U.Set(i).H];
65      end
66
67      % Remove (some) redundant hyperplanes
68      Ab = unique(Ab,'rows');
69      A = Ab(:,1:end-1); b = Ab(:,end); T = [];
70      for i = 1:height(Ab)
71          for j = 1:length(H)
72              % Create new region with cut and check if it lies
                    within the region
73              P_l = Polyhedron('A',[H(j).A; A(i,:)],'b',[H(j).b; b
                    (i,:)]);
74              P_r = Polyhedron('A',[H(j).A; -A(i,:)],'b',[H(j).b;
                    -b(i,:)]);
75              % Check if it cuts the region (same as checking if
                    either of the two results in an empty set)
76              tfl = isEmptyCheby(P_l);
77              tfr = isEmptyCheby(P_r);
78              if tfl ~= 1 && tfr ~= 1
79                  % Region k is being cut. Remove region j and add
                        two new ones
80                  T = [T; P_l];
81                  T = [T; P_r];
82              else
83                  T = [T; H(j)];
84              end
85          end
86          H = T;
87          T = [];
88      end
89
90      % Functions
91      for i=1:height(H)
92          cc = H(i).chebyCenter.x;
93          [~,~,Ig,~] = G.feval(cc);
94          g(i,:) = [G.Set(Ig).getFunction('primal').F G.Set(Ig).
                    getFunction('primal').g];
95          [~,~,If,~] = U.feval(cc);
96          f(i,:) = [U.Set(If).getFunction('primal').F U.Set(If).
                    getFunction('primal').g];
97          h(i,:) = g(i,:) - f(i,:);
98          fun(i) = AffFunction(h(i,1:end-1), h(i,end));
```

```
99          H(i).addFunction(fun(i),'primal');
100     end
101     H = PolyUnion('Set', H, 'Bounded', true, 'Connected', true,
            'Overlaps', false, 'Convex', true);
102
103     % Collect all functions
104     for i = 1:G.Num
105         g(i,:) = [G.Set(i).getFunction('primal').F G.Set(i).
                getFunction('primal').g];
106     end
107     for i = 1:H.Num
108         h(i,:) = [H.Set(i).getFunction('primal').F H.Set(i).
                getFunction('primal').g];
109     end
110
111     g = unique(g,'rows');
112     h = unique(h,'rows');
113
114 end
```

cpwa2mmpscd

Purpose

Computes the conjunctive MMPS function of a continuous PWA function

Synopsis

[mmpskg] = cpwa2mmpscd(U)

Description

A continuous PWA function can be transformed into a conjunctive MMPS function. The algorithm follows the FBR partition to construct the sets for the conjunctive MMPS function as described in this paper.

See Also

cpwa2mmpscd

Matlab implementation

```
1  function [mmpscd] = cpwa2mmpscd(U)
2
3      %% Partition into fbr regions
4      [OL,fbr] = fbr(U);
5
6      % ChebyCenter to generate a point inside each fbr region.
7      for i = 1:fbr.Num
8          % active function
9          a = [fbr.Set(i).getFunction('primal').F fbr.Set(i).
               getFunction('primal').g];
10         active(:,i) = double(ismember(list,a,'rows'));
11         % ordered lattice
12         cc(:,i) = fbr.Set(i).chebyCenter.x;
13
14         %% Evaluate all functions with input point
15         lp = list(:,1:end-1)*cc(:,i) + list(:,end);
16         % Ordered index set
17         [~,Iv] = sort(lp,'ascend');
18         [~,ordered(:,i)] = sort(Iv);
19
20     end
21
```

```
22      OL.list = list;
23      OL.ordered = ordered;
24      OL.active = active;
25
26      %% Ordered lattice to conjunctive lattices
27      for i = 1:width(OL.active)
28          ori_domin_index{i} = find(OL.active(:,i),1);
29          order_up_matrix(:,i) = double(OL.ordered(:,i)>=OL.
               ordered(ori_domin_index{i},i));
30          order_down_matrix(:,i) = double(OL.ordered(:,i)<=OL.
               ordered(ori_domin_index{i},i));
31      end
32
33      %% Lattice Object
34      %lat = lattice();
35      lattice_fun.affine=OL.list';
36      lattice_fun.Poly=fbr.Set;
37      lattice_fun.oriindex=ori_domin_index;
38      lattice_fun.uporder=order_up_matrix';
39      lattice_fun.downorder=order_down_matrix';
40
41      %% Return fMMPScd object
42      phi = lat.affine';
43      psiup = lat.uporder';
44      bounds = pwa.convexHull.H;
45      type = 1;
46      % Creation fMMPScd object
47      mmpscd = fMMPScd(phi,psiup,bounds,type);
48  end
```

mmpscd2mmpskg

Purpose

Computes the kripganz MMPS function of a conjunctive MMPS function, using the method described in [30]

Synopsis

[mmpskg] = mmpscd2mmpskg(fMMPScd)

Description

A conjunctive MMPS function can be decomposed into two convex functions. The resulting function is called a kripfganz MMPS function. The method for transforming a conjunctive MMPS function is based on the results from [30].

See Also

cpwa2mmpscd, cpwa2mmpskg

Matlab implementation

```
function [mmpskg] = mmpscd2mmpskg(mmpscd,method)

    if strcmp(method,'wang')
        %% Compute g
        % Transform set matrix into set cells
        [N_aff,N_sets] = size(mmpscd.Sets);
        st = mmpscd.Sets'.*(1:1:N_aff);
        % Fill cells
        for i = 1:N_sets
            I{i} = nonzeros(st(i,:))';
        end

        % Get all combinations between the sets
        setcombsg = combsets(I)';
        % Initialize matrices
        [m,n] = size(setcombsg);
        AB_g = mmpscd.Type*inf*ones(m,mmpscd.Dim + 1,N_sets);

        % Get all combinations of matrices
        for i = 1:N_sets
            AB_g(:,:,i) = mmpscd.Functions(setcombsg(:,i),:);
```

```
22          end
23
24          % Sum over dimension 3 to get g matrix
25          g = sum(AB_g,3);
26
27          %% Compute h
28          setcombsh = [];
29          for i = 1:length(setcombsg)
30              setcombsh = [setcombsh;nchoosek(setcombsg(i,:),n-1)
                    ];
31          end
32          % remove all recurring rows
33          setcombsh = unique(setcombsh,'rows');
34          % get sizes
35          [k,l] = size(setcombsh);
36          AB_h = mmpscd.Type*inf*ones(k,mmpscd.Dim + 1,l);
37
38          % Get all combinations of matrices
39          for i = 1:l
40              AB_h(:,:,i) = mmpscd.Functions(setcombsh(:,i),:);
41          end
42
43          % Sum over dimension 3 to get g matrix
44          h = sum(AB_h,3);
45
46          %% Create fMMPSkg object
47          mmpskg = fMMPSkg(g,h);
48      end
49  end
```

fMMPScd

Purpose

Conjunctive MMPS function object

Synopsis

[CD] = fMMPScd(Alpha, Beta, phi, psi)

Description

Class object to store conjunctive MMPS functions. Can be initialised using either alpha and beta matrices, or via structure- and parameter matrices.

See Also

fMMPSkg, fMMPSg

Matlab implementation

```
classdef fMMPScd < handle

    properties
        % Conjunctive == 1, Disjunctive == -1
        Type;
        % Matrix form
        Alpha;
        Beta;
        % Index Set form
        phiMatrix;
        psi;
        % Dimension
        Dim;
        % Domain
        Domain;
    end

    methods
        function obj = fMMPScd(phi,psi,bounds,type)
            % Set/Index description
            obj.phiMatrix = phi;
            obj.psi = psi;
            obj.Domain = bounds;
```

```
24              obj.Type = type;
25              [~,n] = size(phi);
26              obj.Dim = n - 1;
27              obj.computeAB();
28          end
29
30          function obj = computeAB(obj)
31              [m,n] = size(obj.psi);
32              % Initialize matrices
33              A = obj.Type*inf*ones(m,obj.Dim,n);
34              B = obj.Type*inf*ones(m,1,n);
35              st = obj.psi'.*(1:1:m);
36
37              % Fill matrices
38              for i = 1:n
39                  I = nonzeros(st(i,:));
40                  A(I,:,i) = obj.phiMatrix(I,1:obj.Dim);
41                  B(I,:,i) = obj.phiMatrix(I,end);
42              end % end for
43
44              obj.Alpha = A;
45              obj.Beta = B;
46          end % end function
47
48          function f_x = feval(obj,x)
49              % Evaluate each affine function
50              F_t = pagemtimes(obj.Alpha,x) + obj.Beta;
51
52              % minimize
53              F_tmin = min(F_t,[],1);
54
55              % maximize
56              F_tmax = max(F_tmin,[],3);
57
58              % Output
59              f_x = F_tmax;
60          end
61
62          function active = getActive(obj,x)
63              fx = obj.feval(x);
64              F = obj.phiMatrix(:,1:obj.Dim)*x + obj.phiMatrix(:,
                    end);
65              active = find(F==fx);
66          end
67
68          function plt = fplot(obj,range,h)
69              % 1D
```

```
70              if height(range) == 1
71                  x1 = range(1,1):h:range(1,2);
72                  Fx = obj.feval(x1);
73                  plt = plot(x1,Fx);
74              end % end if
75
76              % 2D
77              if height(range) == 2
78                  x1 = range(1,1):h:range(1,2);
79                  x2 = range(2,1):h:range(2,2);
80                  [~,n] = size(x1);
81                  for i = 1:n
82                      x = [x1(i)*ones(1,n);x2];
83                      % Evaluate at each point
84                      Fx(i,:) = obj.feval(x);
85                  end % end for
86                  plt = surf(x1,x2,Fx);
87              end % end if
88          end % end function
89
90          function obj = computeFBR(obj,method)
91              if strcmp(method,'ol')
92                  fbr = fbr2(obj.phiMatrix,obj.Domain);
93              elseif strcmp(method,'full')
94                  fbr = fbr1(obj.phiMatrix,obj.Domain);
95              else
96                  fbr = fbr3(obj.phiMatrix,obj.Domain);
97              end
98
99              obj.FBR = fbr;
100         end
101
102         function obj = reduce(obj)
103             obj.psi = unique(obj.psi',"rows")';
104             obj.computeAB;
105         end
106
107         function r = cd2kg(obj,method)
108            r = mmpscd2mmpskg(obj,method);
109         end
110
111         function r = cd2cpwa(obj,method)
112            r = mmpscd2mmpskg(obj,method);
113         end
114
115     end
116 end
```

fMMPSkg

Purpose

Kripfganz MMPS function object

Synopsis

[KG] = fMMPSkg(g,h)

Description

Class object to store kripfganz MMPS functions. Can be initialised using the parameter matrices g and h.

See Also

fMMPScd, fMMPSg

Matlab implementation

```
classdef fMMPSkg
    properties
        % Convex == 1, Concave == -1
        Type;
        % g and h
        g;
        h;
        % Functions
        phiMatrix;
        % Dimension
        Dim;
        % Domain
        Domain;
    end

    methods
        function obj = fMMPSkg(g,h)
            %MMPSKG Construct an instance of this class
            %   Detailed explanation goes here
            obj.g = g;
            obj.h = h;

            [m,n] = size(g);
```

```
24              [k,~] = size(h);
25              obj.Dim = n - 1;
26
27              cb = combvec(1:1:m,1:1:k)';
28              obj.phiMatrix = g(cb(:,1),:) - h(cb(:,2),:);
29          end
30
31          function f_x = feval(obj,x)
32
33              % Two Maximizations
34              Fg = obj.g(:,1:end-1)*x + obj.g(:,end);
35              Fh = obj.h(:,1:end-1)*x + obj.h(:,end);
36
37              Fmax = max(Fg,[],1) - max(Fh,[],1);
38              % Output
39              f_x = Fmax;
40          end
41
42          function active = getActive(obj,x)
43              fx = obj.feval(x);
44              F = obj.phiMatrix(:,1:obj.Dim)*x + obj.phiMatrix(:,
                    end);
45              active = find(F==fx);
46          end
47
48          function plt = fplot(obj,range,h)
49              % 1D
50              if height(range) == 1
51                  x1 = range(1,1):h:range(1,2);
52                  Fx = obj.feval(x1);
53                  plt = plot(x1,Fx);
54              end % end if
55
56              % 2D
57              if height(range) == 2
58                  x1 = range(1,1):h:range(1,2);
59                  x2 = range(2,1):h:range(2,2);
60                  [~,n] = size(x1);
61                  for i = 1:n
62                      x = [x1(i)*ones(1,n);x2];
63                      % Evaluate at each point
64                      Fx(i,:) = obj.feval(x);
65                  end % end for
66                  plt = surf(x1,x2,Fx);
67              end % end if
68          end % end function
69
```

```
70          function plt = pplot(obj,range,h)
71              % 1D
72              if height(range) == 1
73
74                  x1 = range(1,1):h:range(1,2);
75                  Fx = obj.feval(x1);
76
77                  Fg = max(obj.g(:,1:end-1)*x1 + obj.g(:,end));
78                  Fh = max(obj.h(:,1:end-1)*x1 + obj.h(:,end));
79
80                  plt = plot(x1,Fx,x1,Fg,x1,-Fh);
81              end % end if
82
83              % 2D
84              if height(range) == 2
85                  x1 = range(1,1):h:range(1,2);
86                  x2 = range(2,1):h:range(2,2);
87                  [~,n] = size(x1);
88                  for i = 1:n
89                      x = [x1(i)*ones(1,n);x2];
90                      % Evaluate at each point
91                      Fx(i,:) = obj.feval(x);
92                      Fg(i,:) = max(obj.g(:,1:end-1)*x + obj.g(:,
                            end));
93                      Fh(i,:) = max(obj.h(:,1:end-1)*x + obj.h(:,
                            end));
94                  end % end for
95                  plt = figure;
96                  surf(x1,x2,Fx); hold on;
97                  surf(x1,x2,Fg);
98                  surf(x1,x2,-Fh);
99                  plt = plt;
100             end % end if
101         end % end function
102
103     end
104 end
```

fMMPSg

Purpose

General MMPS function object

Synopsis

[G] = fMMPSg(p,s,beta,bounds)

Description

Class object to store general MMPS functions. Is initialised by a matrix holding the parameters for each affine function (p), a matrix holding each operator subjected to the respective affine function (s), the scaling magnitudes (beta) and the domain (bounds)

See Also

fMMPSkg, fMMPScd

Matlab implementation

```
classdef fMMPSg < handle

    properties
        pMatrix;
        sMatrix;
        betaMatrix;
        bounds;
        phiMatrix;
    end

    methods
        function obj = fMMPSg(p,s,beta,bounds)
            obj.pMatrix = p;
            obj.sMatrix = s;
            obj.betaMatrix = beta;
            obj.bounds = bounds;
        end

        % Evaluate the function
        function f_x = feval(obj,x)
            f_x = obj.pMatrix(:,1:end-1)*x + obj.pMatrix(:,end);
            for i = 1:width(obj.sMatrix)
```

```
23                  mx = find(obj.sMatrix(:,i)==1); mn = find(obj.
                        sMatrix(:,i)==2);
24                  sm = find(obj.sMatrix(:,i)==3); sc = find(obj.
                        sMatrix(:,i)==4);
25                  if ~isempty(mx)
26                      f_x(mx,:) =  repmat(max(f_x(mx,:)),length(mx
                            ),1);
27                  end
28                  if ~isempty(mn)
29                      f_x(mn,:) =  repmat(min(f_x(mn,:)),length(mn
                            ),1);
30                  end
31                  if ~isempty(sm)
32                      f_x(sm,:) =  repmat(sum(f_x(sm,:)),length(sm
                            ),1);
33                  end
34                  if ~isempty(sc)
35                      f_x(sc,:) =  f_x(sc,:).*beta(sc,i);
36                  end
37              end
38          end
39
40          % Get the active function for a point x
41          function active = getActive(obj,x)
42              fx = obj.feval(x);
43              F = obj.phiMatrix(:,1:obj.Dim)*x + obj.phiMatrix(:,
                    end);
44              active = find(F==fx);
45          end
46
47          % Plot the function
48          function plt = fplot(obj,range,h)
49              % Ugly but working way of dealing with dimensions
50              % 1D
51              if height(range) == 1
52                  x1 = range(1,1):h:range(1,2);
53                  Fx = obj.feval(x1);
54                  plt = plot(x1,Fx);
55              end % end if
56
57              % 2D
58              if height(range) == 2
59                  x1 = range(1,1):h:range(1,2);
60                  x2 = range(2,1):h:range(2,2);
61                  [~,n] = size(x1);
62                  for i = 1:n
63                      x = [x1(i)*ones(1,n);x2];
```

```
64                      % Evaluate at each point
65                      Fx(i,:) = obj.feval(x);
66                  end % end for
67                  plt = surf(x1,x2,Fx);
68              end % end if
69          end
70
71          % getPhiMatrix - computes all possible affine function
               outcomes
72          function obj = getPhiMatrix(obj)
73
74              % Copy over the structure matrix of the general MMPS
                   function
75              stemp = obj.sMatrix;
76
77              % Copy over the parameter matrix of the general MMPS
                   function
78              for i = 1:height(p)
79                  phi{i} = p(i,:);
80              end
81
82              % Compute all possible outcomes.
83              % Four nested loops to check for each MMPS operation
84              for i = 1:width(stemp)
85
86                  % Nested Loop 1 - Check for maximizations
87                  mx = find(stemp(:,i)==1);
88                  if ~isempty(mx)
89                      temp = [];
90                      for j = 1:length(mx)
91                          temp = [temp; phi{mx(j)}];
92                      end
93
94                      % Add the possibilities to a single cell and
                           purge the other cells
95                      phi{mx(1)} = temp;
96                      stemp(mx(2:end,:),:) = [];
97                      phi(mx(2:end,:)) = [];
98                  end
99
100                 % Nested Loop 2 - Check for minimizations
101                 mn = find(stemp(:,i)==2);
102                 if ~isempty(mn)
103                     % Collect all possibilities
104                     temp = [];
105                     for j = 1:length(mn)
106                         temp = [temp; phi{mn(j)}];
```

```
107                     end
108
109                     % Add the possibilities to a single cell and
                            purge the other cells
110                     phi{mn(1)} = temp;
111                     stemp(mn(2:end,:),:) = [];
112                     phi(mn(2:end,:)) = [];
113                 end
114
115                 % Nested Loop 3 - Check for summations
116                 sm = find(stemp(:,i)==3);
117                 if ~isempty(sm)
118                     temp = [];
119                     for j = 1:length(sm)
120                         temp{j} = 1:1:height(phi{sm(j)});
121                     end
122                     combs = combsets(temp);
123
124                     temp2 = [];
125                     for j = 1:width(combs)
126                         temp3 = [];
127                         for k = 1:height(combs)
128                             temp3 = [temp3; phi{sm(k)}(combs(k
                                    ,:),:)];
129                         end
130                         temp2 = [temp2; sum(temp3,1)];
131                     end
132
133                     % Add the possibilities to a single cell and
                            purge the other cells
134                     phi{sm(1)} = temp2;
135                     stemp(sm(2:end,:),:) = [];
136                     phi(sm(2:end,:)) = [];
137                 end
138
139                 % Nested Loop 4 - Check for scalar
                        multiplications
140                 sc = find(stemp(:,i)==4);
141                 if ~isempty(sc)
142                     for j = 1:length(sc)
143                         phi{sc(j)} =  phi{sc(j)}*beta(sc(j),i);
144                     end
145                 end
146             end
147
148             % Remove any recurring affine functions and store
149             obj.phiMatrix = unique(phi{1},'rows');
```

```
150             end
151         end
152     end
```

loadExample

Purpose

Loading multiple examples

Synopsis

[U] = loadExample('example#')

Description

Helper function to load examples of continuous PWA functions that can then be used to transform into MMPS objects. Both 1- and 2-dimensional examples.

See Also

fbr1, fbr2, fbr3

Matlab implementation

```
function [phi,Omega,bounds] = loadExample(example)
    % Check which example is requested
    if strcmp(example,'example1')
        %% Example 1 - 2D - CPWA - Example from Jun Xu paper (two sided pyramid)
        name = "Example 1";
        phi = [80 -50 -10;-50 80 -10;0 0 0; 0 0 0];
        % Domain as hyperplanes
        bounds = [-1 0 0;0 -1 0;1 0 1;0 1 1];
        % Regions
        Omega{1} = [-8 5 -1; 1 -1 0; 0 1 1;-1 0 0;0 -1 0;1 0 1;0 1 1];
        Omega{2} = [-1 1 0; 5 -8 -1; 1 0 1;-1 0 0;0 -1 0;1 0 1;0 1 1];
        Omega{3} = [-5 8 1; -1 1 0; 0 -1 0; 1 0 1;-1 0 0;0 -1 0;1 0 1;0 1 1];
        Omega{4} = [8 -5 1; 1 -1 0; 0 1 1; -1 0 0;-1 0 0;0 -1 0;1 0 1;0 1 1];
    elseif strcmp(example,'example2')
        %% Example 2 - 2D - CPWA -  "Pyramid in desert"
        name = "Example 2";
        a = 5; b = 3; c = b/a;
        % Continuous Piecewise Affine Function stored in alpha and beta
```

```
19          phi = [0 a b;a 0 b;0 -a b;-a 0 b;0 0 0;0 0 0;0 0 0;0 0
                0];
20          % Domain as hyperplanes
21          bounds = [-1 0 1;0 -1 1;1 0 1;0 1 1];
22          % Regions
23          Omega{1} = [-1 1 0;1 1 0;0 -1 c;-1 0 1;0 -1 1;1 0 1;0 1
                1];
24          Omega{2} = [1 -1 0;1 1 0;-1 0 c;-1 0 1;0 -1 1;1 0 1;0 1
                1];
25          Omega{3} = [1 -1 0;-1 -1 0;0 1 c;-1 0 1;0 -1 1;1 0 1;0 1
                 1];
26          Omega{4} = [-1 1 0;-1 -1 0;1 0 c;-1 0 1;0 -1 1;1 0 1;0 1
                 1];
27          Omega{5} = [1 0 -c;1 -1 0;1 1 0;-1 0 1;0 -1 1;1 0 1;0 1
                1];
28          Omega{6} = [-1 0 -c;-1 1 0;-1 -1 0;-1 0 1;0 -1 1;1 0 1;0
                 1 1];
29          Omega{7} = [0 -1 -c;1 -1 0;-1 -1 0;-1 0 1;0 -1 1;1 0 1;0
                 1 1];
30          Omega{8} = [0 1 -c;-1 1 0;1 1 0;-1 0 1;0 -1 1;1 0 1;0 1
                1];
31      elseif strcmp(example,'example3')
32          %% EXAMPLE 3 - 2D - CPWA - (tetrahedron) three sided
                pyramid
33          name = "Example 3";
34          a = 50; b = 30; c = b/a;
35          % Continuous Piecewise Affine Function stored in alpha
                and beta
36          phi = [a 0 b;0 -a b;-a a 0;0 0 0;0 0 0;0 0 0];
37          % Domain as hyperplanes
38          bounds = [-1 0 10;0 -1 10;1 0 10;0 1 10];
39          % Regions
40          Omega{1} = [1 -1 0;1 1 0;-1 0 c;2*a -a -b;-1 0 10;0 -1
                10;1 0 10;0 1 10];
41          Omega{2} = [1 -1 0;-1 -1 0;0 1 c;a -2*a -b;-1 0 10;0 -1
                10;1 0 10;0 1 10];
42          Omega{3} = [1 -1 0;-2*a a b;-a 2*a b;-1 0 10;0 -1 10;1 0
                 10;0 1 10];
43          Omega{4} = [-1 1 0;-1 0 10;0 -1 10;1 0 10;0 1 10];
44          Omega{5} = [1 0 -c;1 -1 0;1 1 0;-1 0 10;0 -1 10;1 0 10;0
                 1 10];
45          Omega{6} = [0 -1 -c;1 -1 0;-1 -1 0;-1 0 10;0 -1 10;1 0
                10;0 1 10];
46      elseif strcmp(example,'example4')
47          %% EXAMPLE 4 - 1D - CPWA - Jun Xu paper
48          name = "Example 4";
```

```
49          % Continuous Piecewise Affine Function stored in alpha
                and beta
50          phi = [0.5 0.5;2 -1;0 2;-2 9;-0.5 3];
51          % Domain as hyperplanes
52          bounds = [-1 0;1 5];
53          % Regions
54          Omega{1} = [-1 0;1 1;-1 0;1 5];
55          Omega{2} = [-1 -1;1 1.5;-1 0;1 5];
56          Omega{3} = [-1 -1.5;1 3.5;-1 0;1 5];
57          Omega{4} = [-1 -3.5;1 4;-1 0;1 5];
58          Omega{5} = [-1 -4;1 5;-1 0;1 5];
59      elseif strcmp(example,'example5')
60          %% Example 5 - 2D - CPWA - Example from Hempel paper
61          name = "Example 5";
62          a = 1; b = -3; c = 1; d = 4;
63          % Continuous Piecewise Affine Function stored in alpha
                and beta
64          phi = [0 0 -2;a 0 b;0 a b;-a 0 b;0 -a b;a 0 c;0 a c;-a 0
                    c;0 -a c;a 0 c;0 a c;-a 0 c;0 -a c];
65          % Domain as hyperplanes
66          bounds = [1 0 5;0 1 5;-1 0 5;0 -1 5];
67          % Regions
68          Omega{1} = [1 0 1;-1 0 1;0 1 1;0 -1 1;1 0 5;0 1 5;-1 0
                5;0 -1 5];
69          Omega{2} = [-1 0 -1;-1 1 0;-1 -1 0;1 1 d;1 -1 d;1 0 5;0
                1 5;-1 0 5;0 -1 5];
70          Omega{3} = [0 -1 -1;1 -1 0;-1 -1 0;1 1 d;-1 1 d;1 0 5;0
                1 5;-1 0 5;0 -1 5];
71          Omega{4} = [1 0 -1;1 1 0;1 -1 0;-1 1 d;-1 -1 d;1 0 5;0 1
                    5;-1 0 5;0 -1 5];
72          Omega{5} = [0 1 -1;-1 1 0;1 1 0;-1 -1 d;1 -1 d;1 0 5;0 1
                    5;-1 0 5;0 -1 5];
73          Omega{6} = [1 0 0;-1 -1 0;1 -1 -d;1 0 5;0 1 5;-1 0 5;0
                -1 5];
74          Omega{7} = [0 1 0;1 -1 0;1 1 -d;1 0 5;0 1 5;-1 0 5;0 -1
                5];
75          Omega{8} = [-1 0 0;1 1 0;-1 1 -d;1 0 5;0 1 5;-1 0 5;0 -1
                    5];
76          Omega{9} = [0 -1 0;-1 1 0;-1 -1 -d;1 0 5;0 1 5;-1 0 5;0
                -1 5];
77          Omega{10} = [1 0 0;-1 1 0;1 1 -d;1 0 5;0 1 5;-1 0 5;0 -1
                    5];
78          Omega{11} = [0 1 0;-1 -1 0;-1 1 -d;1 0 5;0 1 5;-1 0 5;0
                -1 5];
79          Omega{12} = [-1 0 0;1 -1 0;-1 -1 -d;1 0 5;0 1 5;-1 0 5;0
                    -1 5];
```

```
80          Omega{13} = [0 -1 0;1 1 0;1 -1 -d;1 0 5;0 1 5;-1 0 5;0
                -1 5];
81      elseif strcmp(example,'example6')
82          %% EXAMPLE 4 - 1D - CPWA - Jun Xu paper
83          name = "Example 6";
84          % Continuous Piecewise Affine Function stored in alpha
                and beta
85          phi = [-2 5;0 2;-0.5 3.5];
86          % Domain as hyperplanes
87          bounds = [-1 0;1 4];
88          % Regions
89          Omega{1} = [-1 0;1 1.5;-1 0;1 4];
90          Omega{2} = [-1 -1.5;1 3;-1 0;1 4];
91          Omega{3} = [-1 -3;1 4;-1 0;1 4];
92
93      elseif strcmp(example,'example7')
94          name = "Example 7";
95
96          p = [8 6;0 1;2 1;-2 1;-2 0];
97
98          s(1,:) = [0 0 2 3];
99          s(2,:) = [0 0 2 3];
100         s(3,:) = [2 1 4 3];
101         s(4,:) = [2 1 4 3];
102         s(5,:) = [0 1 4 3];
103
104         beta = [
105             0 0 0 0;
106             0 0 0 0;
107             0 0 -2 0;
108             0 0 -2 0;
109             0 0 -2 0;
110             ];
111
112         bounds = [-1 1;1 1];
113
114         f = fMMPSg(p,s,beta,bounds);
115     end
116 end
```

Bibliography

[1] G.L. Alexanderson and John E. Wetzel. "Arrangements of planes in space". In: *Discrete Mathematics* 34.3 (1981), pp. 219–240. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(81)90002-9. URL: https://www.sciencedirect.com/science/article/pii/0012365X81900029.

[2] A. Bemporad, W.P.M.H. Heemels, and B. De Schutter. "On hybrid systems and closed-loop MPC systems". In: *IEEE Transactions on Automatic Control* 47.5 (2002), pp. 863–869. DOI: 10.1109/TAC.2002.1000287. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0036576888&doi=10.1109%2fTAC.2002.1000287&partnerID=40&md5=40731eec086ea01c500e9a947e1d73ad.

[3] A. Bemporad and M. Morari. "Control of systems integrating logic, dynamics, and constraints". In: *Automatica* 35.3 (1999), pp. 407–427. ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(98)00178-2. URL: https://www.sciencedirect.com/science/article/pii/S0005109898001782.

[4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Robert Creighton Buck. "Partition of space". In: *The American Mathematical Monthly* 50.9 (1943), pp. 541–544.

[6] L.O. Chua and A.C. Deng. "Canonical piecewise-linear representation". In: *IEEE Transactions on Circuits and Systems* 35.1 (1988), pp. 101–111. DOI: 10.1109/31.1705.

[7] B. De Schutter and T.J.J. van den Boom. "MPC for continuous piecewise-affine systems". In: *Systems and Control Letters* 52.3-4 (2004), pp. 179–192. DOI: 10.1016/j.sysconle.2003.11.010. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-2642538423&doi=10.1016%2fj.sysconle.2003.11.010&partnerID=40&md5=a90b3a6413c160051d18defd379d8aa8.

[8] B. De Schutter and T.J.J. van den Boom. "On model predictive control for max-min-plus-scaling discrete event systems". In: *Technical Report Bds 00-04: Control Systems Engineering, Faculty of Information Technology and Systems* (2000).

[9] Morris H DeGroot and Mark J Schervish. *Probability and statistics*. Pearson Education, 2012.

[10] V.V. Gorokhovik and O.I. Zorko. "Piecewise Affine Functions and Polyhedral Sets". In: *Optimization* 31.3 (1994), pp. 209–221. DOI: 10.1080/02331939408844018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0010761911&doi=10.1080%2f02331939408844018&partnerID=40&md5=5fd6ea425cecc9bb994892c420fabd58.

[11] J. Gunawardena. "Min-max functions". In: *Discrete Event Dynamic Systems: Theory and Applications* 4.4 (1994), pp. 377–407. DOI: 10.1007/BF01440235. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0001192686&doi=10.1007%2fBF01440235&partnerID=40&md5=0bdb3be9074fc6e62f34597d544d866e.

[12] W.P.M.H. Heemels, B. De Schutter, and A. Bemporad. "Equivalence of hybrid dynamical models". In: *Automatica* 37.7 (2001), pp. 1085–1091. DOI: 10.1016/S0005-1098(01)00059-0. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0035400417&doi=10.1016%2fS0005-1098%2801%2900059-0&partnerID=40&md5=6cfeca1fca13ab45088a72636c704b14.

[13] W.P.M.H. Heemels, J.M. Schumacher, and S. Weiland. "Linear complementarity systems". In: *SIAM Journal on Applied Mathematics* 60.4 (2000), pp. 1234–1269. DOI: 10.1137/S0036139997325199. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0033685377&doi=10.1137%2fS0036139997325199&partnerID=40&md5=90f266d7e9fb1d91bfea425f73949d4b.

[14] A.B. Hempel, P.J. Goulart, and J. Lygeros. "Every continuous piecewise affine function can be obtained by solving a parametric linear program". In: *2013 European Control Conference, ECC 2013* (2013), pp. 2657–2662. DOI: 10.23919/ecc.2013.6669386. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84893295893&doi=10.23919%2fecc.2013.6669386&partnerID=40&md5=ca46d1dad6b1c436fe4fe21fa2f72e25.

[15] A.B. Hempel, P.J. Goulart, and J. Lygeros. "Inverse Parametric Optimization With an Application to Hybrid System Control". In: *IEEE Transactions on Automatic Control* 60.4 (2015), pp. 1064–1069. DOI: 10.1109/TAC.2014.2336992. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84926363945&doi=10.1109%2fTAC.2014.2336992&partnerID=40&md5=9552b0b5f39453fe41905a2a964aac6a.

[16] Martin Herceg, Michal Kvasnica, Colin N Jones, and Manfred Morari. "Multi-parametric toolbox 3.0". In: *2013 European control conference (ECC)*. IEEE. 2013, pp. 502–510.

[17] Jeanne W. Kerr and John E. Wetzel. "Platonic Divisions of Space". In: *Mathematics Magazine* 51.4 (1978), pp. 229–234. DOI: 10.1080/0025570X.1978.11976718. eprint: https://doi.org/10.1080/0025570X.1978.11976718. URL: https://doi.org/10.1080/0025570X.1978.11976718.

[18] Leonid Khachiyan, Endre Boros, Konrad Borys, Vladimir Gurvich, and Khaled Elbassioni. "Generating all vertices of a polyhedron is hard". In: *Twentieth Anniversary Volume:* Springer, 2009, pp. 1–17.

[19] A. Kripfganz and R. Schulze. "Piecewise affine functions as a difference of two convex functions". In: *Optimization* 18.1 (1987), pp. 23–29. DOI: 10.1080/02331938708843210. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84860675504&doi=10.1080%2f02331938708843210&partnerID=40&md5=a08f87d6ef8fbd2976e99604ff818f39.

[20] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. "On the number of linear regions of deep neural networks". In: vol. 4. January. Cited by: 537. 2014, pp. 2924–2932. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84930634427&partnerID=40&md5=a39f5440d237faa81163972a4ab1d349.

[21] S. Ovchinnikov. "Max-min representation of piecewise linear functions". In: *Beitrage zur Algebra und Geometrie* 43.1 (2002), pp. 297–302. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-2642577838&partnerID=40&md5=94b36b210b43c7c573a7332799a5

[22] Samuel Roberts. "On the Figures formed by the Intercepts of a System of Straight Lines in a, Plane, and on analogous relations in Space of Three Dimensions". In: *Proceedings of the London Mathematical Society* 1.1 (1887), pp. 405–422.

[23] L Schläfli. "Ueber das Verhalten linearer Kontinua zu einander". In: *Theorie der vielfachen Kontinuität*. Springer, 1901, pp. 29–38.

[24] N. Schlüter and M.S. Darup. "Novel convex decomposition of piecewise affine functions". In: *arXiv preprint arXiv:2108.03950* (2021).

[25] Richard P Stanley et al. "An introduction to hyperplane arrangements". In: *Geometric combinatorics* 13.389-496 (2004), p. 24.

[26] J.M. Tarela and M.V. Martínez. "Region configurations for realizability of Lattice Piecewise-Linear models". In: *Mathematical and Computer Modelling* 30.11-12 (1999), pp. 17–27. DOI: 10.1016/S0895-7177(99)00195-8. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0033486282&doi=10.1016%2fS0895-7177%2899%2900195-8&partnerID=40&md5=dfe50c0740908f7974a7bd9a907a314b.

[27] T.J.J. van den Boom and B. De Schutter. "Modelling and control of discrete event systems using switching max-plus-linear systems". In: *Control Engineering Practice* 14.10 (2006), pp. 1199–1211. DOI: 10.1016/j.conengprac.2006.02.006. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-33744907504&doi=10.1016%2fj.conengprac.2006.02.006&partnerID=40&md5=cd456581611bed820a30a82c06c20952.

[28] T.J.J. van den Boom and B. De Schutter. "Properties of MPC for max-plus-linear systems". In: *European Journal of Control* 8.5 (2002), pp. 453–462. DOI: 10.3166/ejc.8.453-462. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-1542332153&doi=10.3166%2fejc.8.453-462&partnerID=40&md5=dfbc9b51a980a091eb6659461ce4a924.

[29] T.J.J. van den Boom, B. De Schutter, and A. Gupta. "Max-Min-Plus-Scaling Discrete-Event Systems: Modeling, Control and Scheduling". In: *Internal Report* (2022).

[30] S. Wang. "General constructive representations for continuous piecewise-linear functions". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 51.9 (2004), pp. 1889–1896. DOI: 10.1109/TCSI.2004.834521. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-4744368326&doi=10.1109%2fTCSI.2004.834521&partnerID=40&md5=237f52c9270ff79598d01f2280c0ee0d.

[31] C. Wen, X. Ma, and B. Erik Ydstie. "Analytical expression of explicit MPC solution via lattice piecewise-affine function". In: *Automatica* 45.4 (2009), pp. 910–917. DOI: 10.1016/j.automatica.2008.11.023. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-61849134460&doi=10.1016%2fj.automatica.2008.11.023&partnerID=40&md5=9c72d0db22414bf8503cb4a340e0893d.

[32] R.H. Wilkinson. "A Method of Generating Functions of Several Variables Using Analog Diode Logic". In: *IEEE Transactions on Electronic Computers* EC-12.2 (1963), pp. 112–129. DOI: 10.1109/PGEC.1963.263420. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0009556152&doi=10.1109%2fPGEC.1963.263420&partnerID=40&md5=514d0c283a7767d648e73cfa3d45e353.

[33] J. Xu, T.J.J. van den Boom, L. Busoniu, and B. De Schutter. "Model predictive control for continuous piecewise affine systems using optimistic optimization". In: *Proceedings of the American Control Conference* 2016-July (2016), pp. 4482–4487. DOI: 10.1109/ACC.2016.7526058. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84992166027&doi=10.1109%2fACC.2016.7526058&partnerID=40&md5=e1b750c674128dba93dc08c89886a0af.

[34] J. Xu, T.J.J. van den Boom, and B. De Schutter. "Optimistic optimization for continuous nonconvex piecewise affine functions". In: *Automatica* 125 (2021). DOI: 10.1016/j.automatica.2020.109476. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85099459512&doi=10.1016%2fj.automatica.2020.109476&partnerID=40&md5=52300b4891c6dd6365ed877e72453186.

[35] J. Xu, T.J.J. van den Boom, B. De Schutter, and S. Wang. "Irredundant lattice representations of continuous piecewise affine functions". In: *Automatica* 70 (2016), pp. 109–120. DOI: 10.1016/j.automatica.2016.03.018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84963684554&doi=10.1016%2fj.automatica.2016.03.018&partnerID=40&md5=c7a07d0cac6b713c27f300959d52a75a.

[36] Thomas Zaslavsky. *Facing up to arrangements: Face-count formulas for partitions of space by hyperplanes: Face-count formulas for partitions of space by hyperplanes*. Vol. 154. American Mathematical Soc., 1975.

[37] Seth Zimmerman. "Slicing Space". In: *The College Mathematics Journal* 32.2 (2001), pp. 126–128. ISSN: 07468342, 19311346. URL: http://www.jstor.org/stable/2687119 (visited on 10/06/2022).

Glossary

List of Acronyms

MMPS	Max-Min-Plus-Scaling
PWA	Piecewise Affine
FBR	Finest Base Region
MPC	Model Predictive Control
LC	Linear Complementarity
ELC	Extended Linear Complementarity
MLD	Mixed Logical Dynamical
DES	Discrete Event Systems
OL	Ordered Lattice

List of Symbols

Abbreviations

γ	Modified structure matrix
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_\top$	Set of real numbers including ∞
$\mathbb{R}_\varepsilon$	Set of real numbers including $-\infty$
$\mathbb{R}_c$	Set of real numbers including $-\infty$ and ∞
$\mathscr{R}$	Anyone of three sets $\mathbb{R}_\top$, $\mathbb{R}_c$ or $\mathbb{R}_\varepsilon$
Ω	Convex polyhedral domain
Φ	Finest Base Region
ϕ	Parameter matrix
ψ	Structure matrix
H	Hyperplane
N_*	Total number of the subscript object $*$