Image computation for polynomial dynamical systems using the Bernstein expansion

Thao Dang and David Salinas

VERIMAG,
Centre Equation,
2 avenue de Vignate, 38610 Gières, France

Abstract. This paper is concerned with the problem of computing the image of a set by a polynomial function. Such image computations constitute a crucial component in typical tools for set-based analysis of hybrid systems and embedded software with polynomial dynamics, which found applications in various engineering domains. One typical example is the computation of all states reachable from a given set in one step by a continuous dynamics described by a differential or difference equation. We propose a new algorithm for over-approximating such images based on the Bernstein representation of polynomial functions. The images are stored using template polyhedra. Using a prototype implementation, the performance of the algorithm was demonstrated on two practical systems as well as a number of randomly generated examples.

1 Introduction

Hybrid systems, that is, systems exhibiting both continuous and discrete dynamics, have been an active research domain, thanks to their numerous applications in chemical process control, avionics, robotics, and most recently in molecular biology. Due to the safety critical features of many such applications, formal analysis is a topic of particular interest. A major component in any verification algorithm for hybrid systems is an efficient method to compute the reachable set of their continuous dynamics described by differential or difference equations. Well-known properties of affine systems and other simpler systems can be exploited to design relatively efficient methods[1] (such as a recently-developed method can handle continuous systems of 100 and more variables [14]). Nevertheless, nonlinear systems are much more difficult to analyze.

In this work, we address the following image computation problem: given a set in $\mathbb{R}^n$, compute its image by a polynomial. This problem typically arises when we need to deal with a dynamical system of the form $x_{k+1} = \pi(x_k)$ where π is a multivariate polynomial. Such dynamical systems could result from a numerical approximation of a continuous or hybrid system. Many existing reachability computation methods for continuous systems can be seen as an extension of numerical integration. For reachability analysis which requires considering all possible solutions (for example, due to non-determinism in initial conditions), one has to solve the above equation with sets, that is x_k and x_{k+1} in the equation are subsets of $\mathbb{R}^n$ (while they are points if we only need a single solution, as in numerical integration). In addition, similar equations can arise in embedded control systems, such

[1] The hybrid systems reachability computation literature is vast. The reader is referred to the recent proceedings of the conference HSCC.

as some physical system controlled by a computer program, which is the implementation of some continuous (or possibly hybrid) controller using appropriate discretization.

One reason for which we are interested in the image computation problem for polynomials is that such systems can be used to model a variety of physical phenomena in engineering, economy and bio-chemical networks. This problem was previously considered in [7], where a method using Bézier techniques from Computer Aided Geometric Design (CADG) was proposed. The drawback of this method is that it requires expensive convex hull and triangulation computation, which restricts its application to systems of dimensions not higher than $3, 4$. The essence of the new method we propose in this paper can be summarized as follows. By using a special class of polyhedra together with optimization, we are able to reduce the complexity of the required polyhedral manipulation. Furthermore, by exploiting a technique from CADG, namely the Bernstein expansion, we only need to solve linear programming problems instead of polynomial optimization problems.

Our method is similar to a number of existing methods for continuous and hybrid systems in the use of linear approximation. Its novelty resides in the efficient way of computing linear approximations. Indeed, a common method to approximate a non-linear function by a piecewise linear one, as in the hybridization approach [2] for hybrid systems, requires non-linear optimization. Our method exploits the Bernstein expansion of polynomials to replace expensive polynomial programming by linear programming. Besides constrained global optimization, other important applications of the Bernstein expansion include various control problems [10] (in particular, robust control). The approximation of the range of a multivariate polynomial over a box is also used in program analysis and optimization (for example [26, 5]). In the hybrid systems verification, polynomial optimization is used to compute barrier certificates [22], and the algebraic properties of polynomials are used to compute polynomial invariants [27].

The paper is organized as follows. In Section 2 we introduce the notion of template polyhedra and the Bernstein expansion of polynomials. We then formally state our problem and describe an optimization based solution. In order to transform the polynomial optimization problem to a linear programming problem, a method for computing bound functions for polynomials is then presented. Section 4 describes an algorithm summarizing the main steps of our method. Some experimental results, in particular the analysis of a control and a biological systems, are reported in Section 6.

2 Preliminaries

Notation. Let $\mathbb{R}$ denote the set of reals. Throughout the paper, vectors is often written using bold letters. Exceptionally, scalar elements of multi-indices, introduced later, are written using bold letters. Given a vector $\mathbf{x}$, x_i denotes its i^{th} component. Capital letters, such as A, B, X, Y denote matrices or sets. If A is a matrix, A^i denotes the i^{th} row of A. An affine function is thus represented as $\mathbf{c}^T\mathbf{x} + \mathbf{d}$.

We use B_u to denote the unit box anchored at the origin, that is $B_u = [0, 1]^n$. We use π to denote a vector of n functions such that for all $i \in \{1, \dots, n\}$, π_i is an n-variate polynomial of the form $\pi_i : \mathbb{R}^n \to \mathbb{R}$. In the remainder of the paper, we sometimes refer to π simply as "a polynomial".

To discuss the Bernstein expansion of polynomials, we use multi-indices of the form $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_n)$ where each $\mathbf{i}_j$ is a non-negative integer. Given two multi-indices $\mathbf{i}$ and $\mathbf{d}$, we write $\mathbf{i} \leq \mathbf{d}$ if for all $j \in \{1, \dots, n\}$, $\mathbf{i}_j \leq \mathbf{d}_j$. Also, we write $\frac{\mathbf{i}}{\mathbf{d}}$ for $(\mathbf{i}_1/\mathbf{d}_1, \mathbf{i}_2/\mathbf{d}_2, \dots, \mathbf{i}_n/\mathbf{d}_n)$ and $\binom{\mathbf{i}}{\mathbf{d}}$ for $\binom{\mathbf{i}_1}{\mathbf{d}_1}\binom{\mathbf{i}_2}{\mathbf{d}_2}\dots\binom{\mathbf{i}_n}{\mathbf{d}_n}$.

2.1 Template polyhedra

A convex polyhedron is a conjunction of a finite number of linear inequalities described as $A\mathbf{x} \leq \mathbf{b}$, where A is a $m \times n$ matrix, $\mathbf{b}$ is a column vector of size m. Template polyhedra are commonly used in static analysis of programs for computing invariants. Ranges [6] and the octagon domains [20] are special template polyhedra. General template polyhedra are used as an abstract domain to represent sets of states in [23, 4].

We now briefly recall template polyhedra and algorithms for their manipulation. The reader is referred to [23] for a thorough description of template polyhedra. A template is a set of linear functions over $\mathbf{x} = (x_1, \ldots, x_n)$. We denote a template by an $m \times n$ matrix H, such that each row H^i corresponds to the linear function $H^i\mathbf{x}$. Given such a template H and a real-valued vector $\mathbf{d} \in \mathbb{R}^m$, a template polyhedron is defined by considering the conjunction of the linear inequalities of the form $\bigwedge_{i=1,\ldots,m} H^i\mathbf{x} \leq d_i$. We denote this polyhedron by $\langle H, \mathbf{d} \rangle$.

By changing the values of the elements of $\mathbf{d}$, one can define a family of template polyhedra corresponding to the template H. We call $\mathbf{d}$ a *polyhedral coefficient vector*. Given $\mathbf{d}, \mathbf{d}' \in \mathbb{R}^m$, if $\forall i \in \{1, \ldots, m\} \ : \ d_i \leq d'_i$, we write $\mathbf{d} \preceq \mathbf{d}'$. Given an $m \times n$ template H and two polyhedral coefficient vectors $\mathbf{d}, \mathbf{d}' \in \mathbb{R}^m$, if $\mathbf{d} \preceq \mathbf{d}'$ then the inclusion relation $\langle H, \mathbf{d} \rangle \subseteq \langle H, \mathbf{d}' \rangle$ holds, and we say that $\langle H, \mathbf{d} \rangle$ is not larger than $\langle H, \mathbf{d}' \rangle$.

The advantage of template polyhedra over general convex polyhedra is that the Boolean operations (union, intersection) and common geometric operations can be performed more efficiently [23].

2.2 Bernstein expansion

We consider an n-variate polynomial $\pi : \mathbb{R}^n \to \mathbb{R}^n$ defined as: $\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I_{\mathbf{d}}} \mathbf{a_i x^i}$ where $\mathbf{a_i}$ is a vector in $\mathbb{R}^n$; $\mathbf{i}$ and $\mathbf{d}$ are two multi-indices of size n such that $\mathbf{i} \leq \mathbf{d}$; $I_{\mathbf{d}}$ is the set of all multi-indices $\mathbf{i} \leq \mathbf{d}$, that is $I_{\mathbf{d}} = \{\mathbf{i} \mid \mathbf{i} \leq \mathbf{d}\}$. The multi-index $\mathbf{d}$ is called the *degree* of π.

Given a set $X \subset \mathbb{R}^n$, the image of X by π, denoted by $\pi(X)$, is defined as follows:

$$\pi(X) = \{(\pi_1(\mathbf{x}), \ldots, \pi_n(\mathbf{x})) \mid \mathbf{x} \in \mathbb{R}^n\}.$$

In order to explain the Bernstein expansion of the polynomial π, we first introduce Bernstein polynomials. For $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the $\mathbf{i}^{th}$ Bernstein polynomial of degree $\mathbf{d}$ is: $\mathcal{B}_{\mathbf{d},\mathbf{i}}(\mathbf{x}) = \beta_{\mathbf{d}_1,\mathbf{i}_1}(x_1) \ldots \beta_{\mathbf{d}_n,\mathbf{i}_n}(x_n)$ where for a real number y, $\beta_{\mathbf{d}_j,\mathbf{i}_j}(y) = \binom{\mathbf{d}_j}{\mathbf{i}_j} y^{\mathbf{i}_j}(1 - y^{\mathbf{d}_j - \mathbf{i}_j})$. Then, for all $\mathbf{x} \in B_u = [0,1]^n$, the polynomial π can be written using the Bernstein expansion as follows: $\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I_{\mathbf{d}}} \mathbf{b_i} \mathcal{B}_{\mathbf{d},\mathbf{i}}(\mathbf{x})$ where for each $\mathbf{i} \in I_{\mathbf{d}}$ the Bernstein coefficient $\mathbf{b_i}$ is defined as:

$$\mathbf{b_i} = \sum_{\mathbf{j} \leq \mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}} \mathbf{a_j}. \tag{1}$$

The following properties of the Bernstein coefficients are of particular interest.

1. Convex-hull property: $Conv\{(\mathbf{x}, \pi(\mathbf{x})) \ : \ \mathbf{x} \in B_u\} \subseteq Conv\{(\mathbf{i}/\mathbf{d}, \mathbf{b_i}) \mid \mathbf{i} \in I_{\mathbf{d}}\}$. The points $\mathbf{b_i}$ are called the control points of π.
2. The above enclosure yields: $\forall \mathbf{x} \in B_u \ : \ \pi(\mathbf{x})) \in \Box(\{\mathbf{b_i} \mid \mathbf{i} \in I_{\mathbf{d}}\})$ where $\Box$ denotes the bounding box of a point set.

3. Sharpness of some special coefficients: $\forall \mathbf{i} \in I^0_{\mathbf{d}} : \ \mathbf{b_i} = \pi(\mathbf{i}/\mathbf{d})$ where $I^0_{\mathbf{d}}$ is the set of all the vertices of $[0, \mathbf{d}_1] \times [0, \mathbf{d}_2] \ldots [0, \mathbf{d}_n]$.

Let us return to the main problem of the paper, which is computing the image of a set by a polynomial. Using the above convex-hull property, we can use the coefficients of the Bernstein expansion to over-approximate the image of the unit box B_u by the polynomial π. To compute the image of a general convex polyhedron, one can over-approximate the polyhedron by a box and then transform it to the unit box via some affine transformation. A similar idea, which involves using the coefficients of the Bézier simplex representation, was used in [7] to compute the image of a convex polyhedron. However, the convex hull computation is expensive especially in high dimensions, which poses a major problem in continuous and hybrid systems verification approaches using polyhedral representations.

In this work, we propose a new method which can avoid complex convex hull operations over general convex polyhedra as follows. First, we use template polyhedra to over-approximate the images. Second, the problem of computing such template polyhedra can be formulated as a polynomial optimization problem. This optimization problem is computationally difficult, despite recent progress in the development of methods and tools for polynomial programming (see for example [28, 17, 9] and references therein). We therefore seek their affine bound functions for polynomials, in order to transform the polynomial optimization problem to a linear programming one, which can be solved more efficiently (in polynomial time) using well-developed techniques, such as Simplex [8] and interior point techniques [3]. Indeed, the above-described Bernstein expansion is used to compute these affine bound functions. This is discussed in the next section.

2.3 Bound functions

To compute bound functions, we use the method using the Bernstein expansion, published in [12]. Computing convex lower bound functions for polynomials is a problem of great interest, especially in global optimization. The reader is referred to [11–13] for more detailed descriptions of this method and its variants.

Before continuing, it is important to note that the method described in this section only works for the case where the variable domain is the unit box B_u. We however want to compute the images of more general sets, in particular polyhedra. An extension of this method to such cases will be developed in Section 3.2.

A simple affine lower bound function is a constant function, which can be deduced from the second property of the Bernstein expansion: $x_i \leq \min\{\mathbf{b_i} \mid \mathbf{i} \in I_{\mathbf{d}}\} = \mathbf{b}_{\mathbf{i}^0} = \mathbf{b}^0$. The main idea of the method is as follows. We first compute the affine lower bound function whose corresponding hyperplane passes through this control point $\mathbf{b}^0$. Then, we aditionally determine $(n-1)$ hyperplanes passing through n other control points. This allows us to construct a sequence of n affine lower bound functions $l_0, l_1, \ldots l_n$. We end up with l_n, a function whose corresponding hyperplane passes through a lower facet of the convex hull spanned by these control points. A summary of the algorithm can be found in Appendix. Note that we can easily compute upper bound functions of π by computing the lower bound functions for $-\pi$ using this method and then multiply each resulting function by -1.

3 Image approximation using template polyhedra

This section is concerned with the main problem of the paper, which can be formulated as follows. We want to use a template polyhedron $\langle H, \mathbf{d} \rangle$ to over-approximate the image of a polyhedron P by the polynomial π. The template matrix H, which is of size $m \times n$ is assumed to be given; the polyhedral coefficient vector $\mathbf{d} \in \mathbb{R}^m$ is however unknown. The question is thus to find $\mathbf{d}$ such that

$$\pi(P) \subseteq \langle H, \mathbf{d} \rangle. \tag{2}$$

It is not hard to see that the following condition is sufficient for (2) to hold: $\forall \mathbf{x} \in P \ : \ H\pi(\mathbf{x}) \leq \mathbf{d}$. Therefore, to determine $\mathbf{d}$, one can formulate the following optimization problem:

$$\forall i \in \{1, \ldots, m\}, d_i = \max(\Sigma_{k=1}^{n} H_k^i \pi_k(\mathbf{x})) \text{ subj. to } \mathbf{x} \in P. \tag{3}$$

where H^i is the i^{th} row of the matrix H and H_k^i is its k^{th} element. Note that the above functions to optimize are polynomials. As mentioned earlier, polynomial optimization is expensive. Our solution is to bound these functions with affine functions, in order to transform the above optimization problem to a linear programming one. This is formalized as follows.

3.1 Optimization-based solution

In Section 2.3 we discussed lower bound functions for polynomials. Note that these bound functions are valid only then the variables $\mathbf{x}$ are inside the unit box B_u. To consider more general domains, we introduce the following definition.

Definition 1 (Upper and lower bound functions). *Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the function $\upsilon : \mathbb{R}^n \to \mathbb{R}$ is called an upper bound function of f w.r.t. a set $X \subset \mathbb{R}^n$ if $\forall \mathbf{x} \in X : \ f(\mathbf{x}) \leq \upsilon(\mathbf{x})$. A lower bound function can be defined similarly.*

The following property of upper and lower bound functions is easy to prove.

Lemma 1. *Given two set $X, Y \subseteq \mathbb{R}^n$ such that $Y \subseteq X$, if υ is an upper (lower) bound function of f w.r.t. X, then υ is also an upper (lower) bound function of f w.r.t. Y.*

For each $k \in \{1, \ldots, m\}$, let $u_k(\mathbf{x})$ and $l_k(\mathbf{x})$ respectively be an upper bound function and a lower bound function of $\pi_k(\mathbf{x})$ w.r.t. a bounded polyhedron $P \subset \mathbb{R}^n$. We consider the following optimization problem:

$$\forall i \in \{1, \ldots, m\}, d_i = \Sigma_{k=1}^{n} H_k^i \omega_k. \tag{4}$$

where the term $H_k^i \omega_k$ is defined as follows:

- If the element $H_k^i > 0$, $H_k^i \omega_k = H_k^i \max u_k(\mathbf{x})$ subj. to $\mathbf{x} \in P$;
- If the element $H_k^i \leq 0$, $H_k^i \omega_k = H_k^i \min l_k(\mathbf{x})$ subj. to $\mathbf{x} \in P$.

The following lemma is a direct result of (4).

Lemma 2. *If a polyhedral coefficient vector $\mathbf{d} \in \mathbb{R}^m$ satisfies (4). Then $\pi(P) \subseteq \langle H, \mathbf{d} \rangle$.*

Proof. It is indeed not hard to see that the solution d_i of the optimization problems (4) is greater than or equal to the solution of (3). Hence, if $\mathbf{d}$ satisfies (4), then

$$\forall i \in \{1, \dots, m\} \; \forall \mathbf{x} \in P : \; \Sigma_{k=1}^{n} H_k^i \pi_k(\mathbf{x}) \leq d_i.$$

This implies that $\forall \mathbf{x} \in P : \; H\pi(\mathbf{x}) \leq \mathbf{d}$, that is the image $\pi(P)$ is included in the template polyhedron $\langle H, \mathbf{d} \rangle$. ■

We remark that if all the bound functions in (4) are affine and P is a bounded convex polyhedron, $\mathbf{d}$ can be computed by solving $2n$ linear programming problems. It remains now to find the affine bound functions u_k and l_k for π w.r.t. a polyhedron P, which is the problem we tackle in the next section.

3.2 Computing affine bound functions over polyhedral domains

As mentioned earlier, the method to compute affine bound functions for polynomials in Section 2.3 can be applied only when the function domain is a unit box, anchored at the origin. The reason is that the expression of the control points of the Bernstein expansion in (1) is only valid for this unit box. If we over-approximate P with a box B, it is then possible to derive a formula expressing the Bernstein coefficients of π over B. However, this formula is complex and its representation and evaluation can become expensive.

We alternatively consider the composition of the polynomial π with an affine transformation τ that maps the unit box B_u to B. The functions resulting from this composition are still polynomials, for which we can compute their bound functions over the unit box, using the formula (1) of the Bernstein expansion. This is explained more formally in the following.

Let B be the bounding box of the polyhedron P, that is, the smallest box that includes P.

The composition $\gamma = (\pi \; o \; \tau)$ is defined as $\gamma(\mathbf{x}) = \pi(\tau(\mathbf{x}))$. The functions τ and γ can be computed symbolically, which will be discussed later.

Lemma 3. *Let $\gamma = \pi \; o \; \tau$. Then, $\pi(P) \subseteq \gamma(B_u)$.*

Proof. By the definition of the composition γ, $\gamma(B_u) = \{\pi(\tau(\mathbf{x})) \mid \mathbf{x} \in B_u\}$. Additionally, $\tau(B_u) = B$. Therefore, $\gamma(B_u) = \pi(B)$. Since the polyhedron P is included in its bounding box B, we thus obtain $\pi(P) \subseteq \pi(B) = \gamma(B_u)$. ■

We remark that the above proof is still valid for any affine function τ. This means that instead of an axis-aligned bounding box, we can over-approximate P more precisely with an oriented (i.e. non-axis-aligned) bounding box. This can be done using the following method.

Computing an oriented bounding box

The directions of an oriented bounding box can be computed using Principal Component Analysis [15]. For a thorough description of the Principal Component Analysis (PCA), the reader is referred to [18].

We first choose a set $S = \{\mathbf{s}^1, \mathbf{s}^2, \dots, \mathbf{s}^m\}$ of m points[2] in the polyhedron P, such that $m \geq n$. We defer a discussion on how this point set is selected to the end of this section. PCA is used to find an

[2] By abuse of notation we use m to denote both the number of template constraints and the number of points here.

orthogonal basis that best represents the point set S. More concretely, we use $\bar{\mathbf{s}}$ to be the mean of S, that is $\bar{\mathbf{s}} = \sum_{i=1}^{m} \mathbf{s}^i$ and we denote $\tilde{\mathbf{s}}_{i,j} = \mathbf{s}_i^j - \bar{\mathbf{s}}_i$. For two points $\mathbf{s}^i$ and $\mathbf{s}^j$ in S, the covariance of their translated points is: $cov(\mathbf{s}_i, \mathbf{s}_j) = \frac{1}{m-1} \sum_{k=1}^{m} \tilde{\mathbf{s}}_{k,i} \tilde{\mathbf{s}}_{k,j}$. Then, we define the co-variance matrix as follows:

$$C = \begin{pmatrix} cov(\mathbf{s}^1, \mathbf{s}^1) & cov(\mathbf{s}^1, \mathbf{s}^2) \ldots cov(\mathbf{s}^1, \mathbf{s}^m) \\ cov(\mathbf{s}^2, \mathbf{s}^1) & cov(\mathbf{s}^2, \mathbf{s}^2) \ldots cov(\mathbf{s}^2, \mathbf{s}^m) \\ & \ldots \\ cov(\mathbf{s}^m, \mathbf{s}^1) & cov(\mathbf{s}^k, \mathbf{s}^2) \ldots cov(\mathbf{s}^m, \mathbf{s}^m) \end{pmatrix}.$$

The n largest singular values of C provide the orientation of the bounding box. More concretely, since C is symetric, by singular value decomposition, we have $C = U \Lambda U^T$ where Λ is the matrix of singular values. The axes of the bounding box are hence determined by the first n columns of the matrix U, and its centroid is $\bar{\mathbf{s}}$.

We now discuss how to select the set S. When the vertices of P are available, we can include them in the set. However, if P is given as a template polyhedron, this requires computing the vertices which is expensive. Moreover, using only the vertices, when their distribution do not represent the geometric form of the polyhedron, may cause a large approximation error, since the resulting principal directions are not the ones along which the points inside P are mostly distributed. To remedy this, we sample points inside P as follows. First, we compute an axis-aligned bounding box of P (this can be done by solving $2n$ linear programming problems). We then uniformly sample points inside this bounding box and keep only the points that satisfy the constraints of P. Uniform sampling on the boundary of P in general enables a better precision. More detail on this can be found in [8].

4 Image computation algorithm

The following algorithm summarizes the main steps of our method for over-approximating the image of a bounded polyhedron $P \subset \mathbb{R}^n$ by the polynomial π. The templates are an input of the algorithm. In the current implementation of the algorithm, the templates can be fixed by the user, or the templates forming regular sets are used.

Algorithm 1 Over-approximating $\pi(P)$

```
/* Inputs: convex polyhedron $P$, polynomial $\pi$, templates $H$ */

$B = PCA(P)$                              /* Compute an oriented bounding box */
$\tau = UnitBoxMap(B)$                    /* Compute the function mapping the unit box $B_u$ to $B$ */
$\gamma = \pi \; o \; \tau$
$(u, l) = BoundFunctions(\gamma)$         /* Compute the affine bound functions */
$\bar{\mathbf{d}} = PolyApp(u, l, H)$     /* Compute the coefficient vector $\mathbf{d}$ */
$Q = \langle H, \bar{\mathbf{d}} \rangle$ /* Form the template polyhedron and return it */
Return($Q$)
```

The role of the procedure PCA is to compute an oriented bounding box B that encloses P. The procedure $UnitBoxMap$ is then used to determine the affine function τ that maps the unit box B_u

at the origin to B. This affine function is composed with the polynomial π, the result of which is the polynomials γ. The affine lower and upper bound functions l and u of γ are then computed, using the Bernstein expansion. The function *PolyApp* determines the polyhedral coefficient vector $\mathbf{d}$ by solving the linear programs in (4) with u, l and the optimization domain is B_u. The polyhedral coefficient vector $\mathbf{d}$ are then used to define a template polyhedron Q, which is the result to be returned.

Based on the analysis so far, we can state the correctness of Algorithm 1.

Theorem 1. *Let $\langle H, \bar{\mathbf{d}} \rangle$ be the template polyhedron returned by Algorithm 1. Then $\pi(P) \subseteq \langle H, \bar{\mathbf{d}} \rangle$.*

We remark that u and l are upper and lower bound functions of γ with respect to B_u. It is not hard to see that $\tau^{-1}(P) \subseteq B_u$ where τ^{-1} is the inverse of τ. Using the property of bound functions, u and l are also bound functions of γ with respect to $\tau^{-1}(P)$. Hence, if we solve the optimization problem over the domain $\tau^{-1}(P)$ (which is often smaller than B_u), using Lemma 1, the resulting polyhedron is still an over-approximation of $\pi(P)$. This remark can be used to obtain more accurate results.

5 Approximation error and computation cost

We finish this section by briefly discussing the precision and complexity of the method. A more detailed analysis can be found in [8]. The approximation error are caused by the bound functions, the oriented bounding box approximation, and the use of template polyhedra.

The following lemma states an important property of the Bernstein expansion.

Lemma 4. *Let $C_{\pi,B}$ be the piecewise linear function defined by the Bernstein control points of π with respect to the box B. Then, for all $x \in B$, $|\pi(x) - C_{\pi,B}(x)| \leq K\delta^2(B)$ where $|\cdot|$ is the infinity norm on $\mathbb{R}^n$, $\delta(B)$ is the box size (i.e. its largest side length), $K_k = max_{x \in B; i,j \in \{1,\dots,n\}} |\partial_i \partial_j \pi_k(x)|$, $K = max_{k \in \{1,\dots,n\}} K_k$.*

From this result it can be proven that in one dimensional case, the error between the bound functions computed using the Bernstein expansion and the original polynomial is quadratic in the length of box domains. This quadratic convergence seems to hold for higher dimensional cases in practice, as shown in [12]. We conjecture that there exists a subdivision method of the box B which allows a quadratic convergence of the error. This subdivision method is similar to the one used for finding roots of a polynomial with quadratic convergence [21].

On the other hand, a polyhedron can be approximated by a set of non-overlapping oriented boxes with arbitrarily small error. Then, for each box, we compute a bounding function, with which we then compute a coefficient for each template. Finally, for each template, we take the largest coefficient to define the template polyhedron. Since the boxes are smaller, the bounding functions are more precise, we can thus improve the coefficients as much as desired.

Concerning the error inherent to the approximation by template polyhedra, it can be controlled by fine-tuning the number of template constraints. If using this method with a sufficient number of templates to assure the same precision as the convex hull in our previous Bézier method [7], then the convergence of both methods are quadratic. However the Bezier method requires expensive convex-hull and triangulation operations, and geometric complexity of resulting sets may grow step after step. Combining template polyhedra and bounding functions allows a good accuracy-cost compromise.

Concerning complexity issues, the computation of bound functions and PCA only requires manipulating matrices and linear equations. Linear programming can be solved in polynomial time.

Hence, one can expect that our method depends polynomially on the dimensions. When iterating this method to compute the reachable set of a polynomial dynamical system, if the number of template constraints is constant, the complexity depends linearly on the number of iterations.

6 Experimental results

We have implemented our method in a prototype tool. The implementaion uses the template polyhedral library developed by S. Sankaranarayanan [24] and the library **lpsolve** for linear programming. The tool can be used to analyze hybrid systems with polynomial continuous dynamics. In the following, we demonstrate the method with two examples: a control system (modelled as a hybrid system) and a biological system (modelled as a continuous system). The time efficiency of the tool is also evaluated by considering using a number of randomly generated polynomials.

6.1 A control system

The control system we consider is the Duffing oscillator taken from [19, 9]. This is a nonlinear oscillator of second order, the continuous-time dynamics is described by $\ddot{y}(t) + 2\zeta\dot{y}(t) + y(t) + y(t)^3 = u(t)$, where $y \in \mathbb{R}$ is the state variable and $u \in \mathbb{R}$ is the control input. The damping coefficient $\zeta = 0.3$. In [9], using a forward difference approximation with a sampling period $h = 0.05$ time units, this system is approximated by the following discrete-time model:

$$\begin{aligned} x_1(k+1) &= x_1(k) + hx_2(k) \\ x_2(k+1) &= -hx_1(k) + (1-2\zeta h)x_2(k) + hu)k) - hx_1(k)^3 \end{aligned}$$

In [9], an optimal predictive control law $u(k)$ was computed by solving a parametric polynomial optimization problem. In Figure 6.1 one can see the phase portrait of the system under this control law and without it (i.e. $\forall kge0\ u(k) = 0$) is shown . We model this control law by the following switching law with 3 modes:

$$\begin{array}{lll} u(k) & = 0.5 * k & \text{if } 0 \le k \le 10 \\ u(k) & = 5 - 0.5 * (k-10)/3 & \text{if } 10 < k \le 40 \\ u(k) & = 0 & \text{if } k > 40 \end{array}$$

The controlled system is thus modelled as a hybrid automaton [1] with 3 discrete states. The result obtained using our tool on this system is shown in Figure 6.1, which is coherent with the phase portrait in [9]. The initial set is a ball with radius $1e-04$. The number of template constraints is 100. In addition to the reachable set after 120 steps (computed after $3s$), in Figure 6.1, we also illustrate the approximation error by visualizing the template polyhedron after the first step and a cloud of exact points (obtained by sampling the initial set and applying the polynomial to the sampled points).

6.2 A biological system

The second example is the well-known Michaelis-Menten enzyme kinetics, taken from [16]. The kinetic reaction of this signal transduction pathway is represented in Figure 3, where E is the concentration

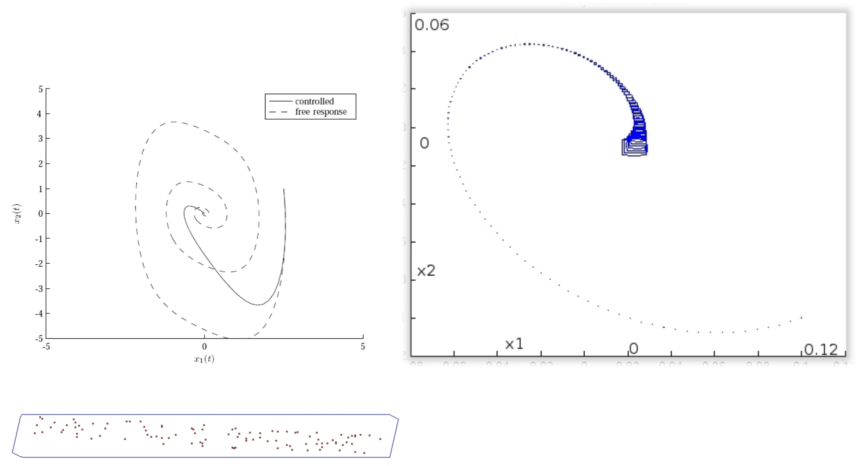


Fig. 1. The Duffing oscillator: phase portrait, the reachable set, and the reachable set after the first step.

$$S + E \underset{\theta_2}{\overset{\theta_1}{\rightleftarrows}} ES \xrightarrow{\theta_3} E + P$$

Fig. 2. Michaelis-Menten enzyme kinetics

of an enzyme that combines with a substrate S to form an enzyme substrate complex ES. In the next step, the complex can be dissociated into E and S or it can further proceed to form a product P. This pathway kinetics can be described by the following ODEs where x_1, x_2, x_3 and x_4 are the concentrations of S, E, ES and P:

$$\begin{aligned}
\dot{x}_1 &= -\theta_1 x_1 x_2 + \theta_2 x_3 \\
\dot{x}_2 &= -\theta_1 x_1 x_2 + (\theta_2 + \theta_3) x_3 \\
\dot{x}_3 &= \theta_1 x_1 x_2 + (\theta_2 + \theta_3) x_3 \\
\dot{x}_4 &= \theta_3 x_3
\end{aligned}$$

Using a second order Runge Kutta discretization with time step 0.3, we obtain the following 4-variate polynomial system:

$$
\begin{aligned}
\pi_1(\mathbf{x}) &= x_1 - 0.053838 x_1 x_2 + 0.001458 x_1^2 x_2 + 0.001458 x_1 x_2^2 + \\
&\quad -3.9366e - 05.x_1^2 x_2^2 + 0.005775 x_3 - 0.002025 x_1 x_3 - 0.000162 x_2 x_3 + \\
&\quad 5.9049e - 05 x_1 x_2 x_3 - 6.075e - 06 x_3^2 \\
\pi_2(\mathbf{x}) &= x_2 - 0.051975 x_1 x_2 + 0.001458 x_1^2 x_2 + 0.001458 x_1 x_2^2 \\
&\quad -3.9366e - 05 x_1^2 x_2^2 + 0.0721875 x_3 - 0.002025 x_1 x_3 - 0.000162 x_2 x_3 + \\
&\quad 5.9049e - 05 x_1 x_2 x_3 - 6.075e - 06 x_3^2 \\
\pi_3(\mathbf{x}) &= 0.051975 x_1 x_2 - 0.001458.x1^2 x_2 - 0.001458 x_1 x_2^2 + \\
&\quad 3.9366e - 05 x1^2 x_2^2 + 0.927812 x_3 + 0.002025 x_1 x_3 + 0.000162 x_2 x_3 \\
&\quad -5.9049e - 05 x_1 x_2 x_3 + 6.075e - 06 x_3^2 \\
\pi_4(\mathbf{x}) &= 0.001863 x_1 x_2 + 0.0664125 x_3 + x_4.
\end{aligned}
$$

The reachable set computed for all the initial states inside a ball centered at $(12, 12, 0, 0)$ with radius $1e - 0.4$ is shown in Figure 3. The number of template constraints is 60. In order to compare with the result in [16], the figures depicts the evolution of each variable for the first 10 steps (the horizontal axis is time). In the vertical axis, the minimal and maximal values of the variables are shown. This result is conherent with the simulation result in [16]. The computation time for 20 steps is $3.7s$.

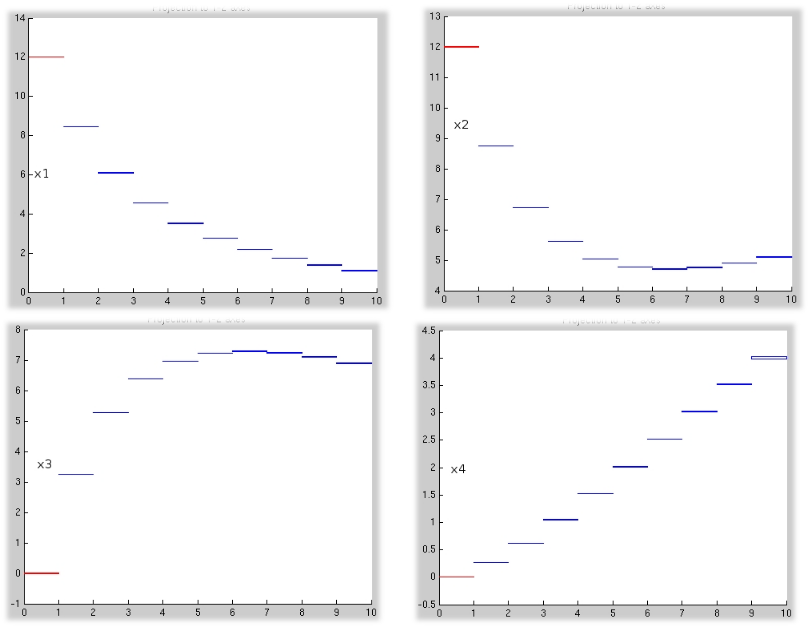


Fig. 3. Michaelis-Menten enzyme kinetics. The evolution of the reachable set after 10 steps

6.3 Randomly generated systems

In order to evaluate the performance of our method, we tested it on a number of randomly generated polynomials in various dimensions and maximal degrees (the maximal degree is the largest degree for all variables). For a fixed dimension and degree, we generated different examples to estimate an average computation time. In the current implementation, polynomial composition is done symbolically, and we do not yet exploit the possibility of sparsity of polynomials (in terms of the number of monomials). The computation time shown in the table in Figure 4 does not include the time for polynomial composition. Note that the computation time for 7-variate polynomials of degree 3 is significant, because the generated polynomials have a large number of monomials; however, practical systems often have a much smaller number of monomials.

As expected, the computation time grows linearly w.r.t. the number of steps. This can be explained by the use of template polyhedra where the number of constraints can be chosen according to required precisions and thus control better the complexity of the polyhedral operations, compared to general convex polyhedra. Indeed, when using general polyhedra, the operations such as convex hull may increase their geometric complexity (roughly described by the number of vertices and constraints).

7 Conclusion

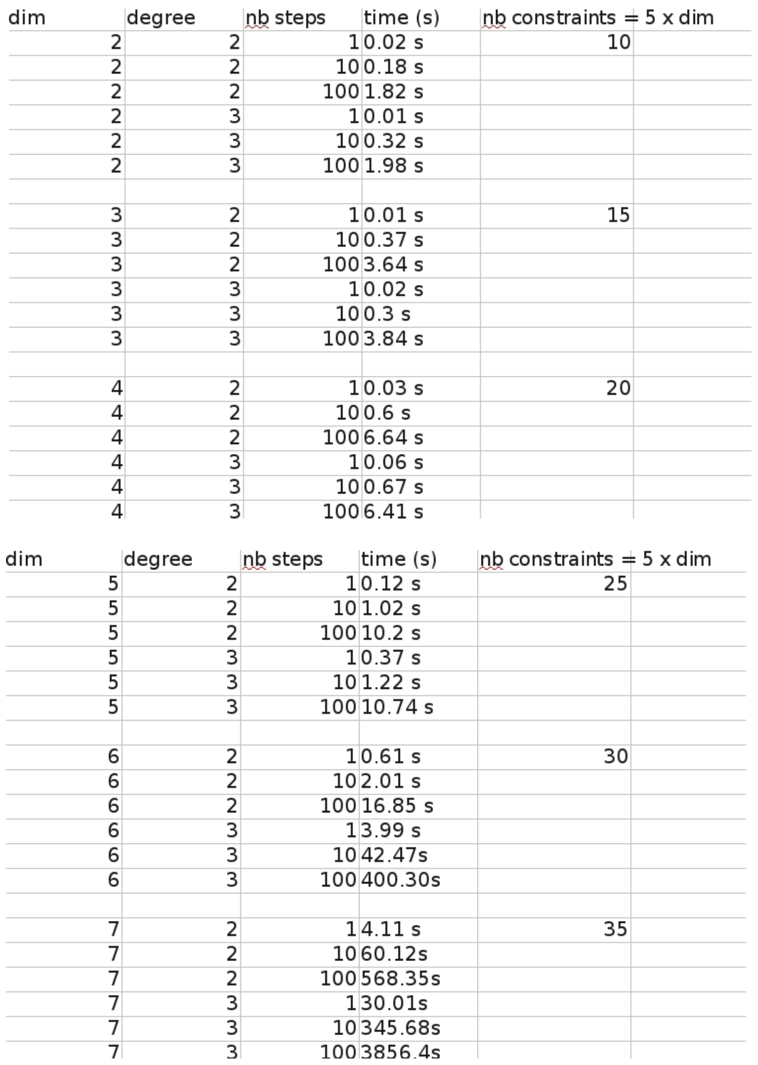
In this paper we propose a new method to compute images of polynomials. This method combines the ideas from optimization and the Bernstein expansion. This result can be readily applicable to solve the reachability analysis problem for hybrid systems with polynomial continuous dynamics.

The performance of the method was demonstrated using a number of randomly generated examples, which shows an improvement in efficiency compared to our previously developed method using Bézier techniques [7]. These encouraging results also show an important advantage of the method: thanks to the use of template polyhedra, the complexity and precision of the method are more controllable than those using polyhedra as symbolic set representations.

There are a number interesting directions to explore. Indeed, different tools from geometric modeling could be exploited to improve the efficiency of the method. For example, polynomial composition can be done for sparse polynomials more efficiently using the blossoming technique [25]. In addition to more experimentation on other hybris systems case studies, we intend to explore a new application domain, which is verification of embedded control software. In fact, multivariate polynomials arise in many situations when analyzing programs that are automatically generated from practical embedded controllers.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica.*, 43(7):451–476, 2007.
3. S. Boyd and S. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.
4. L. Chen, A. Miné, and P. Cousot. A sound floating-point polyhedra abstract domain. In *Proc. of the Sixth Asian Symposium on Programming Languages and Systems (APLAS'08)*, volume 5356 of *LNCS*, pages 3–18, Bangalore, India, December 2008. Springer.



dim	degree	nb steps	time (s)	nb constraints = 5 x dim
2	2	1	0.02 s	10
2	2	10	0.18 s	
2	2	100	1.82 s	
2	3	1	0.01 s	
2	3	10	0.32 s	
2	3	100	1.98 s	
3	2	1	0.01 s	15
3	2	10	0.37 s	
3	2	100	3.64 s	
3	3	1	0.02 s	
3	3	10	0.3 s	
3	3	100	3.84 s	
4	2	1	0.03 s	20
4	2	10	0.6 s	
4	2	100	6.64 s	
4	3	1	0.06 s	
4	3	10	0.67 s	
4	3	100	6.41 s	

dim	degree	nb steps	time (s)	nb constraints = 5 x dim
5	2	1	0.12 s	25
5	2	10	1.02 s	
5	2	100	10.2 s	
5	3	1	0.37 s	
5	3	10	1.22 s	
5	3	100	10.74 s	
6	2	1	0.61 s	30
6	2	10	2.01 s	
6	2	100	16.85 s	
6	3	1	3.99 s	
6	3	10	42.47s	
6	3	100	400.30s	
7	2	1	4.11 s	35
7	2	10	60.12s	
7	2	100	568.35s	
7	3	1	30.01s	
7	3	10	345.68s	
7	3	100	3856.4s	

Fig. 4. Computation time for randomly generated polynomial systems in various dimensions and degrees

5. F. Clauss and I.Yu. Chupaeva. Application of symbolic approach to the bernstein expansion for program analysis and optimization. *Program. Comput. Softw.*, 30(3):164–172, 2004.
6. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming, Dunod, Paris*, pages 106–130, 1976.
7. T. Dang. Approximate reachability computation for polynomial systems. In *HSCC*, pages 138–152, 2006.
8. T. Dang and D. Salinas. Computing set images of polynomias. Technical report, VERIMAG, June 2008.
9. I.A. Fotiou, P. Rostalski, P.A. Parrilo, and M. Morari. Parametric optimization and optimal control using algebraic geometriy methods. *International Journal of Control*, 79(11):1340–1358, 2006.
10. J. Garloff. Application of bernstein expansion to the solution of control problems. In *University of Girona*, pages 421–430, 1999.
11. J. Garloff, C. Jansson, and A.P. Smith. Lower bound functions for polynomials. *Journal of Computational and Applied Mathematics*, 157:207–225, 2003.
12. J. Garloff and A.P. Smith. An improved method for the computation of affine lower bound functions for polynomials. In C. A. Floudas and P. M. Pardalos, editors, *Frontiers in Global Optimization*, Series Nonconvex Optimization and Its Applications, pages 135–144. Kluwer Academic Publ.,Boston, Dordrecht, New York, London, 2004.
13. J. Garloff and A.P. Smith. A comparison of methods for the computation of affine lower bound functions for polynomials. In C. Jermann, A. Neumaier, and D. Sam, editors, *Global Optimization and Constraint Satisfaction*, LNCS, pages 71–85. Springer, 2005.
14. A. Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, pages 291–305, 2005.
15. S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Computer Graphics SIGGRAPH '96*, pages 171–180, 1996.
16. F. He, L. F. Yeung, and M. Brown. Discrete-time model representation for biochemicql pathway systems. *IAENG International Journal of Computer Science*, 34(1), 2007.
17. D. Henrion and J.B. Lasserre. Gloptipoly: Global optimization over polynomials with matlab and sedumi. In *Proceedings of the IEEE Conference on Decision and Control*, 2002.
18. I. T. Jolliffe. *Principal Component Analysis.* Springer, 2002.
19. D.W. Jordan and P. Smith. *Nonlinear Ordinary Differential Equations.* Oxford Applied Mathematics and Computer Science. Oxford University Press, 1987.
20. A. Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO II*, LNCS 2053, pages 155–172. Springer, 2001.
21. B. Mourrain and J. P. Pavone. Subdivision methods for solving polynomial equations. Technical report, INRIA Research report, 5658, August 2005.
22. Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004.
23. S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *Verification, Model-Checking and Abstract-Interpretation (VMCAI 2005)*, LNCS 3385. Springer, 2005.
24. Sriram Sankaranarayanan. Mathematical analysis of programs. Technical report, Standford, 2005. PhD thesis.
25. H.-P. Seidel. Polar forms and triangular B-spline surfaces. In *Blossoming: The New Polar-Form Approach to Spline Curves and Surfaces, SIGGRAPH '91 Course Notes 26, ACM SIGGRAPH*, pages 8.1–8.52, 1991.
26. I. Tchoupaeva. A symbolic approach to bernstein expansion for program analysis and optimization. In *In 13th International Conference on Compiler Construction, CC 2004*, pages 120–133. Springer, 2004.
27. Ashish Tiwari and Gaurav Khanna. Nonlinear systems: Approximating reach sets. In *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 600–614. Springer, 2004.
28. S. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

A Computing affine bound functions using the Bernstein expansion

We describe the algorithm for computing these functions published in [12]. Let us consider a polynomial $\pi_k(\mathbf{x})$, which is the k^{th} component of $\pi(\mathbf{x})$ and for simplicity we denote it simply by $p(\mathbf{x})$. The Bernstein coefficient of p is denoted by the scalars $b_{\mathbf{i}}$. We shall compute an affine lower bound function denoted by $l(\mathbf{x})$.

– Iteration 1.
  - Define the direction $\mathbf{u}^1 = (1, 0, \ldots, 0)$.
  - Compute the slopes from each $b_{\mathbf{i}}$ to b^0 in the direction $\mathbf{u}^1$:

$$\forall \mathbf{i} \in I_{\mathbf{d}} : \mathbf{i}[1] \neq \mathbf{i}^0[1],\ g^1_{\mathbf{i}} = \frac{b_{\mathbf{i}} - b^0}{\mathbf{i}[1]/\mathbf{d}[1] - \mathbf{i}^0[1]/\mathbf{d}[1]}$$

  - Let $\mathbf{i}^1$ be the multi-index with the smallest absolute value of $g^1_{\mathbf{i}}$. Define the lower bound function:

$$l^1(\mathbf{x}) = b^0 + g^1_{\mathbf{i}^1}\mathbf{u}^1(\mathbf{x} - \mathbf{i}^0/\mathbf{d}).$$

– Iteration $j = 2, \ldots, n$.
  - Compute the direction $\bar{\mathbf{u}}^j = (\beta_1, \ldots, \beta_{j-1}, 0, \ldots, 0)$ such that $\bar{\mathbf{u}}^j \frac{\mathbf{i}^k - \mathbf{i}^0}{d} = 0$ for all $k = 1, \ldots, j-1$. This requires solving a system of $j-1$ linear equations with $j-1$ unknown variables. Then normalize $\mathbf{u}^j = \bar{\mathbf{u}}^j / ||\bar{\mathbf{u}}^j||$.
  - Compute the slopes from each $\mathbf{b}_{\mathbf{i}}$ to $\mathbf{b}^0$ in the direction $\mathbf{u}^j$:

$$\forall \mathbf{i} \in I_{\mathbf{d}} : \frac{\mathbf{i}[1] - \mathbf{i}^0[1]}{\mathbf{d}}\mathbf{u}^j \neq 0, g^j_{\mathbf{i}} = \frac{b_{\mathbf{i}} - l^{j-1}(\mathbf{i}/\mathbf{d})}{(\mathbf{i}/\mathbf{d} - \mathbf{i}^0/\mathbf{d})\mathbf{u}^j}$$

  - Let $\mathbf{i}^j$ be the multiindex with the smallest absolute value of $g^j_{\mathbf{i}}$. Define the lower bound function:

$$l^j(\mathbf{x}) = l^{j-1}(\mathbf{x}) + g^j_{\mathbf{i}^j}\mathbf{u}^j(\mathbf{x} - \mathbf{i}^0/\mathbf{d}).$$