Approximate reachability computation for polynomial systems

Thao Dang[1]

VERIMAG,
Centre Equation,
2 avenue de Vignate,
38610 Gières, France
Thao.Dang@imag.fr

Abstract. In this paper we propose an algorithm for approximating the reachable sets of systems defined by polynomial differential equations. Such systems can be used to model a variety of physical phenomena. We first derive an integration scheme that approximates the state reachable in one time step by applying some polynomial map to the current state. In order to use this scheme to compute all the states reachable by the system starting from some initial set, we then consider the problem of computing the image of a set by a multivariate polynomial. We propose a method to do so using the Bézier control net of the polynomial map and the blossoming technique to compute this control net. We also prove that our overall method is of order 2. In addition, we have successfully applied our reachability algorithm to two models of a biological system.

1 Introduction

Reachability analysis is an important problem in formal verification of hybrid systems. A major ingredient in designing a reachability analysis algorithm for hybrid systems is an efficient method to handle their continuous dynamics described by differential equations (since their discrete dynamics can be handled using existing discrete verification methods). Reachability computation methods for a special class of systems with constant derivatives are also well-developed. On the other hand, while many well-known properties of linear differential equations can be exploited to design relatively efficient methods, non-linear systems are much more difficult to analyze. Numerical integration is a common method to solve non-linear differential equations. Its goal is to derive a scheme to approximate the solution at each time step based on the solution at one or several previous steps. In general, a typical numerical integration scheme can be written as: $\mathbf{x}_{k+1} = \mathcal{Y}_k(f, h, \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k)$ where f is the derivative and h is the step size. Nevertheless, while this approach is concerned with computing a single solution at a time and each $\mathbf{x}_k$ here is a point, in reachability analysis one has to deal with sets of all possible solutions (due to non-determinism in initial conditions and in the dynamics of the system). Therefore, wishing to exploit the numerical integration idea for reachable set computation purposes, a question that arises is how to perform such schemes with sets, that is, when each $\mathbf{x}_k$ is a set of points. The essence behind the approach we propose in this paper can be described as extending traditional numerical integration to set integration. In particular, we are interested in systems defined by *polynomial differential equations*. Such systems can be used to model a variety of physical phenomena, in particular, the dynamics of many analog components can be modeled or approximated by polynomial equations.

We first derive an integration scheme that approximates the reachable state $\mathbf{x}_{k+1}$ by applying some polynomial map to $\mathbf{x}_k$. In order to use this scheme to approximate the reachable set, we then consider the problem of computing the image of a set by a multivariate polynomial. To do so, we employ the techniques from computer aided geometric design, in particular the Bézier techniques and the blossoming principle. We also prove that our overall method is of order 2. Although this paper only focuses on continuous systems, but the proposed method can be extended to hybrid systems, since reachable sets are represented by convex polyhedra, and Boolean operations (required to deal with discrete transitions) over such polyhedra can be computed using a variety of existing algorithms.

Before continuing, we present a brief review of related work. The reachability problem for continuous systems described by differential equations has motivated much research both for theoretical problems, such as computability (see for example [2]), and for the development of computation methods and tools. If the goal is to compute exactly the reachable set or approximate it as accurately as possible, one can use a variety of methods for tracking the evolution of the reachable set under the continuous flows using some set represention (such as polyhedra, ellipsoids, level sets) [20, 10, 8, 21, 3, 30, 23, 19]. Since high quality approximations are hard to compute, other methods seek approximations that are sufficiently good to prove the property of interest[1] (such as barrier certificates [24], polynomial invariants [29]). Abstraction methods for hybrid systems are also close in spirit to these methods. Indeed, their main idea is to approximate the original system with a simpler system (that one can handle more efficiently) and refine it if the analysis result obtained for the approximate system is too conservative (see for example [28, 1, 9, 5, 4]).

The paper is organized as follows. In Section 2, after stating our problem, we describe an integration scheme for polynomial differential equations. This scheme requires computing the image of a set by a polynomial map, the problem we discuss in Section 3. We then present our reachability algorithm and some experimental results obtained using the algorithm on a hybrid systems benchmark.

[1] It should be noted that reachable set computations can also be used for controller synthesis where the accuracy criterion is important.

2 Reachability analysis of polynomial systems

Throughout the paper, vectors are often written using bold letters. Given a vector $\mathbf{x}$, $\mathbf{x}[i]$ denotes its i^{th} component.

We consider a polynomial system:

$$\dot{\mathbf{x}}(t) = g(\mathbf{x}(t)). \tag{1}$$

We first rewrite the dynamics of the system as the sum of a linear part $A\mathbf{x}(t)$ and a non-linear part $f(\mathbf{x}(t))$, that is,

$$\dot{\mathbf{x}}(t) = g(\mathbf{x}(t)) = A\mathbf{x}(t) + f(\mathbf{x}(t)). \tag{2}$$

We then consider the non-linear term as independent input. In other words, the system is treated as a linear system with input $f(\mathbf{x}(t))$. This trick is to separate the linear part for which we can derive the exact closed-form solution. The interest in doing so will become clearer when we discuss the approximation error. We now develop a numerical solution for (2). Let $h > 0$ be a time step and $t_k = kh$ where $k = 0, 1, 2, \ldots$. Then, we have

$$\mathbf{x}(t_{k+1}) = e^{Ah}\mathbf{x}(t_k) + \int_0^h e^{A(h-\tau)} f(\mathbf{x}(t_k + \tau))\, d\tau. \tag{3}$$

The idea is to approximate $\mathbf{x}(t_k+\tau)$ in the above equation by its Taylor expansion around t_k to the first order, that is $\alpha(t_k + \tau) = \mathbf{x}(t_k) + g(\mathbf{x}(t_k))\tau$. Denoting $\mathbf{x}(t_k) = \mathbf{x}_k$, $g(\mathbf{x}(t_k)) = g_k$, and $f(\mathbf{x}(t_k)) = f_k$, we have $\alpha(t_k + \tau) = \mathbf{x}_k + g_k\tau = \mathbf{x}_k + (Ax_k + f_k)\tau$. Replacing $\mathbf{x}(t_k+\tau)$ with $\alpha(t_k+\tau)$, we obtain an approximation $\bar{\mathbf{x}}_{k+1}$ of the exact solution $\mathbf{x}_{k+1}$:

$$\bar{\mathbf{x}}_{k+1} = e^{Ah}\mathbf{x}_k + \int_0^h e^{A(h-\tau)} f(\alpha(t_k + \tau))\, d\tau. \tag{4}$$

The integral in the above equation is a function of $\mathbf{x}_k$, and we denote it by $Q(\mathbf{x}_k) = \int_0^h e^{A(h-\tau)} f(\alpha(t_k + \tau))\, d\tau$.

Proposition 1. *The map $Q(\mathbf{x}_k)$ in the equation (4) can be written as a polynomial in $\mathbf{x}_k$.*

Proof. The proof of the proposition is straightforward, however we present it here for the clarity of the presentation that follows. It is easy to see that if the total degree of $f(\mathbf{x}_k)$ is d, then $\alpha(t_k + \tau)$ is a multivariate polynomial of total degree d in $\mathbf{x}_k$ and therefore $f(\alpha(t_k + \tau))$ is a polynomial of degree d in τ. We denote $\Gamma_l = \int_0^h e^{A(h-\tau)}\tau^l\, d\tau$, which can be written in a closed form. Thus, we have $\int_0^h e^{A(h-\tau)} f(\alpha(t_k + \tau))\, d\tau = \sum_{l=0}^{d} \Gamma_l \psi_l(\mathbf{x}_k)$, where for every $l \in \{0, 1, \ldots, d\}$, ψ_l is a polynomial in $\mathbf{x}_k$. □

The resulting integration scheme to approximate the solution of (1) is:

$$\begin{cases} \bar{\mathbf{x}}_{k+1} = e^{Ah}\bar{\mathbf{x}}_k + Q(\bar{\mathbf{x}}_k) = P(\bar{\mathbf{x}}_k), \\ \bar{\mathbf{x}}_0 \quad = \mathbf{x}(0). \end{cases}$$

We call $P(\mathbf{x}_k)$ the *integration map.*

Example of multi-affine systems. Let us illustrate the proof with a simple case where $g(\mathbf{x})$ is a multi-affine function of degree 2. This is the case of a biological model we study in Section 6. The function $f(\mathbf{x})$ can be written as: $f(\mathbf{x}) = \sum_{i,j\in\{1,\dots,n\}, i\neq j} \mathbf{x}[i]\mathbf{x}[j]\boldsymbol{c}_{ij}$ with $\boldsymbol{c}_{ij} \in \mathbb{R}^n$. Then, replacing $\mathbf{x}(t_k+\tau)$ with $\alpha(t_k+\tau) = \mathbf{x}_k + g_k\tau$, we have

$$f(\alpha(t_k+\tau)) = \sum_{i\neq j\in\{1,\dots,n\}} (\ g_k[i]g_k[j]\tau^2 + (\mathbf{x}_k[i]g_k[j] + g_k[i]\mathbf{x}_k[j])\tau + \mathbf{x}_k[i]\mathbf{x}_k[j]\ )\boldsymbol{c}_{ij}$$

Therefore, the equation (4) becomes:

$$\bar{\mathbf{x}}_{k+1} = P(\mathbf{x}_k) = \varPhi\mathbf{x}_k + \sum_{i\neq j\in\{1,\dots,n\}} (\gamma_2\varGamma_2 + \gamma_1\varGamma_1 + \gamma_0\varGamma_0)\boldsymbol{c}_{ij}. \tag{5}$$

where $\varPhi = e^{Ah}$ and $\gamma_2 = g_k[i]g_k[j]$, $\gamma_1 = g_k[i]\mathbf{x}_k[j] + \mathbf{x}_k[i]g_k[j]$, $\gamma_0 = \mathbf{x}_k[i]\mathbf{x}_k[j]$. After straightforward calculations, we obtain:

$$\varGamma_l = l!\sum_{i=0}^{\infty} \frac{A^i h^{i+l+1}}{(i+l+1)!} \tag{6}$$

It is thus easy to see that, due to the term γ_2, $P(\mathbf{x}_k)$ in (5) is a polynomial of degree 4 in $\mathbf{x}_k$. The equation (5) can be readily used as a scheme specialized for multi-affine systems of degree 2.

Convergence. A bound on the error in our approximation is given in the following theorem.

Theorem 1. *Let $\bar{\mathbf{x}}(t_{k+1})$ be the approximate solution at time t_{k+1} (computed by (4)) and $\mathbf{x}(t_{k+1})$ be the corresponding exact solution such that $\bar{\mathbf{x}}(t_k) = \mathbf{x}(t_k)$. Then, a bound on the local error is given by:*

$$||\bar{\mathbf{x}}(t_{k+1}) - \mathbf{x}(t_{k+1})|| = \mathcal{O}(h^3).$$

The proof of this result is presented in Appendix. This theorem shows that the equation (4) is a *second order scheme*. In addition. we can show that the global error is also convergent. As one can see from the proof, the error bound depends on the Lipschitz constant of the non-linear function f. So now we can see the interest in separating the linear part since the Lipschitz constant of f is smaller than that of g.

Higher order integration schemes. Note that we have used an approximation of the exact solution $\mathbf{x}(t_k+\tau)$ by the its first order Taylor expansion around t_k. To obtain better convergence orders, we can use higher order expansions which results in integration schemes involving high order derivatives of $f(\mathbf{x})$. The derivation of such schemes is similar to the above development, but the degree of the resulting integration map $P(\mathbf{x}_k)$ can be higher. In the other direction, if we use a simpler approximation $\alpha(t_k + \tau) = \mathbf{x}_k$ for all $\tau \in [t_k, t_{k+1})$, then $Q(\mathbf{x}_k) = \Gamma_0 f(\mathbf{x}_k)$ and we obtain the classic Euler scheme for the non-linear part. The advantage of this scheme is that the resulting polynomial $Q(\mathbf{x}_k)$ has the same degree as $f(\mathbf{x})$. As we shall see later, the degree of the integration map is one of the factors determining the complexity of the reachability algorithm. It remains to compute the polynomial map $Q(\mathbf{x}_k)$, the problem we tackle in the next section.

3 Computing polynomial maps

The problem we are interested in can be formally stated as follows. Given a polynomial map $\pi : \mathbb{R}^n \to \mathbb{R}^n$ of total degree d and a bounded set $X \subset \mathbb{R}^n$, we want to compute the image $\pi(X)$ defined as: $\pi(X) = \{\pi(\mathbf{x}) \mid \mathbf{x} \in X\}$. We shall focus on the case where X is a simplex in $\mathbb{R}^n$.

3.1 Bézier simplices

To determine the image of a simplex by a polynomial map, we use the results on *Bézier simplices* [16]. We need to introduce first some notation.

A multi-index $\mathbf{i} = (\mathbf{i}[1], \ldots, \mathbf{i}[n+1])$ is a vector of $(n+1)$ non-negative integers. We define the norm of $\mathbf{i}$ by $||\mathbf{i}|| = \sum_{j=1}^{n+1} \mathbf{i}[j]$ and let $\mathcal{I}_n^d$ denote the set of all multi-indices $\mathbf{i} = (\mathbf{i}[1], \ldots, \mathbf{i}[n+1])$ with $||\mathbf{i}|| = d$. We define two special multi-indices: $\mathbf{e}_k$ is a multi-index that has all the components equal to 0 except for the k^{th} component which is equal to 1, and $\mathbf{o}$ is a multi-index that has all the components equal to 0. We call $\mathbf{o}$ the zero multi-index.

Let Δ be a full-dimensional simplex in $\mathbb{R}^n$ with vertices $\{\mathbf{v}_1, \ldots, \mathbf{v}_{n+1}\}$. Given a point $\mathbf{x} \in \Delta$, let $\lambda(\mathbf{x}) = (\lambda_1(\mathbf{x}), \ldots, \lambda_{n+1}(\mathbf{x}))$ be the function that gives the barycentric coordinates of $\mathbf{x}$ with respect to the vertices of Δ, that is, $\mathbf{x} = \sum_{k=1}^{n+1} \lambda_k(\mathbf{x})\mathbf{v}_k$ and $\sum_{k=1}^{n+1} \lambda_k(\mathbf{x}) = 1$. A *Bézier simplex* of degree d of the form $\pi : \mathbb{R}^n \to \mathbb{R}^n$ is defined as[2]:

$$\pi(\mathbf{x}) = \sum_{||\mathbf{i}||=d} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^d(\lambda_1(\mathbf{x}), \ldots, \lambda_{n+1}(\mathbf{x})) \tag{7}$$

where for a given multi-index $\mathbf{i}$, $\mathbf{b}_{\mathbf{i}}$ is a vector in $\mathbb{R}^n$ and $B_{\mathbf{i}}^d : \mathbb{R}^n \to \mathbb{R}$ is a *Bernstein polynomial* of degree d defined as:

$$B_{\mathbf{i}}^d(y_1, \ldots, y_{n+1}) = \binom{d}{\mathbf{i}} y_1^{\mathbf{i}[1]} y_2^{\mathbf{i}[2]} \ldots y_{n+1}^{\mathbf{i}[n+1]} \tag{8}$$

with the multimonial coefficient

$$\binom{d}{\mathbf{i}} = \frac{d!}{\mathbf{i}[1]!\mathbf{i}[2]!\ldots\mathbf{i}[n+1]!}.$$

[2] The definition holds for more general polynomials of the form $\pi : \mathbb{R}^n \to \mathbb{R}^m$.

In the above formula (7), each vector $\mathbf{b_i}$ is called a *Bézier control point* and the set of all such $\mathbf{b_i}$ form the *Bézier control net* of π with respect to Δ.

Any polynomial can be written in form of a Bézier simplex, as in formula (7). This form is a popular way to write polynomials in computer aided geometric design (see [16] and references therein). The following properties of Bernstein polynomials are well-known. The Bernstein polynomials form a partition of unity, that is, $\sum_{||\mathbf{i}||=d} B^d_{\mathbf{i}}(y_1, \ldots, y_{n+1}) = 1$, and they are non-negative, that is, $B^d_{\mathbf{i}}(y_1, \ldots, y_{n+1}) \geq 0$ for all $0 \leq y_1, \ldots, y_{n+1} \leq 1$.

These properties of Bernstein polynomials imply the following *shape properties* of Bézier simplices, which we shall use for reachability computation purposes.

Lemma 1. *Given an arbitrary point $\mathbf{x} \in \Delta$,*

1. **[Convex hull property]** *the point $\pi(\mathbf{x})$ lies inside the convex hull of the control net, that is $\pi(\mathbf{x}) \in conv\{\mathbf{b_i} \mid \mathbf{i} \in \mathcal{I}^d_n\}$.*
2. **[End-point interpolation property]** *π interpolates the control net at the corner control points specified by $\mathbf{b}_{d\mathbf{e}_k}$ for all $k \in \{1, \ldots, n+1\}$.*

Note that the number of multi-indices in $\mathcal{I}^d_k$ is $\binom{d+n}{n}$; therefore, the number of points $\mathbf{b_i}$ is exactly $\binom{d+n}{n} = \frac{(d+n)!}{d!\,n!}$. We denote this number by $\beta(n, d)$.

These shape properties can be used to approximate polynomial maps. Indeed, the *convex hull property* in Lemma 1 shows that one can over-approximate $\pi(\Delta)$ by taking the convex hull of the Bézier control net of π with respect to Δ. In addition, this approximation is tight due to the above *end-point interpolation property.* In the rest of this section we focus on the problem of computing the Bézier control net of the polynomial π. To avoid confusion, it is worthy to emphasize that for reachability computation purposes, we are dealing with the systems whose vector fields are given in monomial form (i.e. sums of monomials), hence the integration map is also defined in this form. To compute the control points of a polynomial given in monomial forms, we shall exploit the techniques for approximating and designing polynomial curves and surfaces. However, it is important to mention that most of such existing tools deal with univariate or bivariate polynomials (often expressed in terms of control points), their application to solve our problem requires an adaptation to multivariate polynomials as well as geometric manipulation in general dimension.

3.2 Computing the Bézier control net

Our goal is to obtain the Bézier control net of a polynomial π given in monomial form. By the definition (7), the most natural approach is to solve the following interpolation problem. Let S be a set of $\beta(n, d)$ points in Δ. For each $\mathbf{x} \in S$, we evaluate $\pi(\mathbf{x})$ and use (7) to obtain a system of linear equations with the coordinates of the Bézier control points $\mathbf{b_i}$ as unknown variables. One can choose the set S such that the unique solution to these linear equations exists [11]. Although this method is conceptually simple, it may require solving a large linear system[3] (which is of size $n*\beta(n,d)$). We shall use a more efficient approach based on the blossoming principle, which is summarized in the following theorem. A thorough description of this principle and its various applications can be found in [25, 27].

Theorem 2 (Blossoming principle). *For any polynomial $\pi : \mathbb{R}^n \to \mathbb{R}^n$ of degree d, there is a unique symmetric d-affine map $p : (\mathbb{R}^n)^d \to \mathbb{R}^n$ such that for all $\mathbf{x} \in \mathbb{R}^n$ $p(\mathbf{x}, \ldots, \mathbf{x}) = \pi(\mathbf{x})$. The map p is called the* blossom *or the* polar form *of π.*

We recall that a map $q(\mathbf{x}_1, \ldots, \mathbf{x}_d)$ is called *d-affine* if it is affine when all but one of its arguments are kept fixed; it is said to be symmetric if its value does not depend on the ordering of the arguments, that is, for any permutation $(\mathbf{y}_1, \ldots, \mathbf{y}_d)$ of $(\mathbf{x}_1, \ldots, \mathbf{x}_d)$ we have $q(\mathbf{y}_1, \ldots, \mathbf{y}_d) = q(\mathbf{x}_1, \ldots, \mathbf{x}_d)$. Given a polynomial π, the connection between its Bézier control net relative to a simplex Δ and its blossom p is described by the following lemma.

Lemma 2. *For all $\mathbf{i} \in \mathcal{I}_n^d$, $\mathbf{b}_\mathbf{i} = p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \underbrace{\mathbf{v}_2, \ldots, \mathbf{v}_2}_{\mathbf{i}[2]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]})$ where $\{\mathbf{v}_1, \ldots, \mathbf{v}_{n+1}\}$ are the vertices of Δ.*

This fact is also well-known [25], and we present its proof in Appendix, which can facilitate understanding subsequent development.

Computing the blossom. We have seen that the Bézier control points can be computed by evaluating the blossom values at some particular points shown in Lemma 2. To compute them, we first derive an analytic expression of the polar form and then show how to compute this expression efficiently. We do so by extending the results for bivariate polynomial surfaces [18] to multivariate polynomials.

Before proceeding, we mention that the problem of computing the Bézier control net can be formulated as a problem of changing from the monomial basis to the Bézier basis, which can be solved using the algorithms proposed in [15, 22]. These algorithms also make use of the blossoming principle. The idea is to express the coordinates of the new basis vectors in the old basis, and then apply the transformation matrix to the old coefficients. However, when the polynomial representation is "sparse", that is it contains many zero coefficients, this sparsity is not exploited. The method discussed in the following deals better with such sparsity since it considers only the monomials with non-null coefficients. More precisely, by "sparse polynomial representations" we mean those where the number of monomials (with non-null coefficients) is much smaller than the number of all combinations of coordinate variables up to degree d. The sparse case happens in many practical applications we have encountered.

[3] The Gaussian elimination algorithm to solve a linear system of size $m \times m$ has the time complexity $O(m^3)$.

We now show how to compute the blossom of monomials which are products of only two variables, such as $\mathbf{x}[i]^h\mathbf{x}[j]^k$. Similar treatment can be used for monomials involving more variables, but due to the length of the involved formulas we do not detail it here. On the other hand, using linearity, we can obtain the blossom of any polynomial expressed as a sum of monomials.

The blossom of degree d of the monomial $(\mathbf{x}[i])^h(\mathbf{x}[j])^k$ is given by:

$$p^d_{h,k}(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d) = \frac{1}{\binom{d}{h}\binom{d-h}{k}} \sum_{\substack{I \cup J \subset \{1, \ldots, d\}, \\ |I| = h, |J| = k, I \cap J = \emptyset}} \prod_{r \in I} \mathbf{u}_r[i] \prod_{s \in J} \mathbf{u}_s[j].$$

To prove this, it suffices to check that the right hand side is a symmetric multi-affine function, and moreover $p^d_{h,k}(\mathbf{u}, \mathbf{u}, \ldots, \mathbf{u}) = (\mathbf{u}[i])^h(\mathbf{u}[j])^k$. □

To compute the blossom values using the above expression, we make use of a recurrence equation on p, as proposed in [18]. We first denote

$$\sigma^d_{h,k} = \frac{1}{\binom{d}{h}\binom{d-h}{k}} p^d_{h,k}(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d).$$

The function σ is symmetric and has the following interpretation: this function is computed by choosing h i^{th} coordinates of the argument points and k i^{th} coordinates and forming their product, then summing these products over all possible choices. We can thus derive the following recurrence formula:

$$\begin{cases} \sigma^d_{h,k} = \sigma^{d-1}_{h,k} + \mathbf{u}_d[i]\sigma^{d-1}_{h-1,k} + \mathbf{u}_d[j]\sigma^{d-1}_{h,k-1} & \text{if } h, k \geq 0 \text{ and } h + k \geq 1, \\ \sigma^d_{0,0} = 0 \end{cases} \tag{9}$$

This means that to compute the required blossom value $p^d_{h,k}(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d)$ we compute all the intermediate values $p^{d'}_{h',k'}(\mathbf{u}_1, \ldots, \mathbf{u}_{d'})$ with $d' \leq d$, $h' + k' \leq d'$. This computation can be done in time $O(d^3)$.

3.3 Approximation error and subdivision

We now estimate an error bound for the approximation of the polynomial map π by its the Bézier control points.

Theorem 3. *For each Bézier control point* $\mathbf{b_i}$ *there exists a point* $\mathbf{y} \in \pi(\Delta)$ *such that* $||\mathbf{b_i} - \mathbf{y}|| \leq K\rho^2$ *where* ρ *be the maximal side length of* Δ *and* K *is some constant not depending on* Δ.

The proof of this theorem can be found in Appendix.

Consequently, when the simplicial domain Δ is large, to achieve the desired accuracy we may need to subdivide it into smaller simplices. This subdivision creates new Bézier bases and therefore new control points. However, due to the properties of multi-affine maps, one can compute the new control nets in a clever way which reuses the computations performed for the original simplex.

The essence behind this idea is as follows. Suppose that we want to partition the simplex Δ by adding a point $\mathbf{x} \in \Delta$ and forming $(n+1)$ new smaller simplices. Then, we can use de Catesljau algorithm [13, 16] to compute the value of the polynomial π at $\mathbf{x}$. It turns out that this computation also produces the control net for the new simplices. Note that this algorithm can only be applied when the Bézier control points of the polynomial are known.

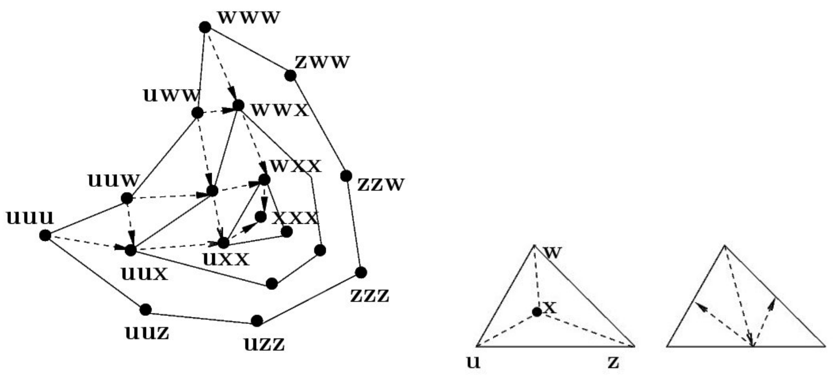

Fig. 1. Subdividing a Bézier control net

We denote $\mathbf{b}_{\mathbf{i}}^{l} = p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]}, \underbrace{\mathbf{x}_1, \ldots, \mathbf{x}_l}_{l})$ with $\mathbf{i}[1] + \ldots + \mathbf{i}[n+1] + l = d$. Since p is symmetric and multi-affine, we have:

$$\mathbf{b}_{\mathbf{i}}^{l} = \lambda_1(\mathbf{x}_l)\mathbf{b}_{\mathbf{i}+\mathbf{e}_1}^{l-1} + \ldots + \lambda_n(\mathbf{x}_l)\mathbf{b}_{\mathbf{i}+\mathbf{e}_n}^{l-1} \tag{10}$$

Note that $\mathbf{b}_{\mathbf{o}}^{n} = p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where $\mathbf{o}$ is the zero multi-index. In addition, with $l = 0$, $\mathbf{b}_{\mathbf{i}}^{0}$ are exactly the Bézier control points of the polynomial. Therefore, by running the above recursion starting from $l = 0$ until $l = n$ we obtain the blossom value at $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. If all the argument points of the blossom are equal to $\mathbf{x}$, the result of the algorithm is $\pi(\mathbf{x})$. The de Catesljau algorithm is illustrated with a 2-dimensional example in Figure 1 where each node is annotated with the arguments of the blossom to evaluate. The nodes on the outermost layer correspond to the control points for the original triangle **uzw**. The incoming arrows of **uux** show that the blossom value at this point is computed from the blossom values at **uuu** and **uuw**. As mentioned earlier, we can see that the computation of $\pi(\mathbf{x})$ indeed produces the Bézier control points for the sub-simplices. Figure 1 shows the values $p(\underbrace{\mathbf{u}, \ldots, \mathbf{u}}_{\mathbf{i}[1]}, \underbrace{\mathbf{x}, \ldots, \mathbf{x}}_{\mathbf{i}[2]}, \underbrace{\mathbf{w}, \ldots, \mathbf{w}}_{\mathbf{i}[3]})$ which are the Bézier control points for the triangle **uxw**.

One important remark is that the subdivision at the center of the simplex does not reduce the maximal side length of the simplices. By Theorem 3 this means that the convergence of the Bézier control net towards the polynomial is not guaranteed. However, one can repeat the bisection at the mid-point of the logest edge, as shown in Figure 1 to achieve the desired accuracy. More generally, the subdivision of a simplex can be defined as follows. For each barycentric coordinate $\lambda_i(\mathbf{x}) > 0$ of a point $\mathbf{x} \in \Delta$ we define a simplex Δ_i obtained from Δ by replacing the vertex $\mathbf{v}_i$ with $\mathbf{x}$. Hence, when the point $\mathbf{x}$ is the mid-point of an edge we obtain a bisection. It was proved in [26] that using the bisection at the mid-point of the longest edge, after n steps (where n is the dimension of the simplex) the simplex diameter is reduced at least by $\sqrt{3}/2$ times.

In two dimensions, another method of subdivision via all the mid-points of the edges was discussed in [18]. This method is however more complex to implement for dimensions higher than 2.

4 Reachability algorithm

Let us summarize our development so far. In Section 2, we presented a scheme to approximate the successor in one time step by applying a polynomial, called the integration map, to the current state. We then showed in Section 3 how to over-approximate the image of a simplex by a polynomial map using the Bézier control net. The result of this approximation is in general a polyhedron.

We are now ready to describe our reachability algorithm for polynomial systems. In Algorithm 1, X_0 is the initial set which is assumed to be a convex polyhedron in $\mathbb{R}^n$, each R_k is a set of convex polyhedra. The function *Bez* over-approximates the image of a simplex Δ by the integration map P, using the method presented in Section 3. The goal of the function *triangulation* triangulates a set of convex polyhedra and returns the set of all simplices of the triangulation. To do so, we collect all the vertices of the polyhedra and compute a triangulation of this set. We then exclude all the simplices in the triagulation whose interior does not intersect with R_k. Let us briefly discuss the precision of

Algorithm 1 Reachable set computation

```
$R_0 = X_0$, $k = 0$
**repeat**
  $S_\Delta = triangulation(R_k)$
  $C = \emptyset$
  **for all** $\Delta \in S_\Delta$ **do**
    $C = C \cup Bez(\Delta)$
  **end for**
  $R_{k+1} = C$
  $k = k + 1$
**until** $R_{k+1} = R_k$
```

the algorithm. We suppose that ρ is the maximal size of the simplices that are produced by the function *triangulation* and h is the integration time step. If the integration map P can be exactly computed, using Theorem 1, we know that the integration error is $\mathcal{O}(h^3)$. In addition, Theorem 3 shows that our approximation of the integration map P induces an error $\mathcal{O}(\rho^2)$. By the triangle inequality, the total error in each iteration of Algorithm 1 is bounded by $(\mathcal{O}(h^3) + \mathcal{O}(\rho^2))$. Therefore, by choosing appropriate value ρ in function of h, we can guarantee that Algorithm 1 is a second order method.

We now discuss some computation issues. The first remark is that the total number of the Bézier control points is $\beta(n, d)$, but the actual number of vertices of their convex hull is often much smaller, depending on the geometric structure of the polynomial map P. On the other hand, in order to speed up the computation (at the price of less precise results), one can approximate C by its convex hull or even by a simplex. Algorithms for doing so have been developed and some algorithms can compute a minimal volume enclosing simplex (such as, [31, 17]).

Let us now briefly discuss the relation between our new algorithm and the reachability algorithm based on hybridization, proposed in [5]. The latter first approximates the (general) non-linear dynamics by a piecewise linear dynamics, using a simplicial decomposition of the state space. Hence, for the approximate system, one can indeed compute the reachable set of each linear dynamics more accurately. However, the treatement of discrete transitions (i.e. the dynamics changes) makes the overall computation very expensive due to the geometric complexity of the intersection between the reachable set and the switching hyperplanes. In the algorithm of this paper, the one-step computation for polynomial systems is in general more costly than that for linear systems, but discrete transitions are avoided. Nevertheless, more experimentation is needed to draw conclusions about the advantages and inconvenients of these two approaches.

5 Approximation using box splines

We now describe another method for approximating the image of a polynomial map. This method is similar with the above method using the Bezier techniques in the expression of the polynomial of interest in another basis, which is a box spline basis. For clarity, let us briefly recall our problem and summa.

Given a multivariate polynomial π, and a set $X \subset \mathbb{R}^n$, we want to compute $Y = \pi(X)$. To this end, we represent the polynomial in question using a box spline basis. As in the case of Bézier basis, the coefficients of this representation allow to approximate the image $Y = \pi(X)$ when the set X is a zonotope. The approximation of $\pi(X)$ thus consists of 2 steps:

1. The set X is first over-approximated by a zonotope. The generators of this zonotope determine the box spline basis functions.
2. We then establish the corresponding box-spline representation of the polynomial π. As we will see later, the convex hull of its coefficients provide an over-approximation of the image of the zonotope.

In the following, we first present a brief introduction of box splines and some of their properties. This summary is necessary for the development that follows.

Then, we show how to approximate the convex hull of a set of points by a zonotope, which is the operation of the step 1. This is particularly necessary if we need to perform the image computation iteratively.

5.1 Box splines: background

We present two definitions of box splines: inductive one and another which is more geometric. Other definitions (such as inductive definition, or by recursion) can be found in [12].

Inductive definition Let n be the dimension (or the number of variables). Given an integer $k \geq n$, let $V = \{v_1, \dots, v_k\}$ be a set of k vectors in $\mathbb{R}^n$ where $v_1, v_2, \dots, v_n$ are linearly independent. Each such vector is called *direction*. We denote by $M = [v_1\, v_2\, \dots, v_k]$ the $n \times k$ matrix the columns of which are the vectors in V_k. It could be assumed that the first n columns of M form the n-dimensional identity matrix I. The inductive definition of a box spline B_M associated with V is as follows.

We start with the base case where the degree is $s = 0$, then $M_s = I$ and $B_{M_s}(x) = 1$ if $x \in [0,1)^s$ and 0 otherwise. Note that this function is piecewise constant and has degree $s = 0$. If we add a column in M denoted by $M \cup v$, then the box spline associated with the new matrix $M_s = M_{s-1} \cup v$ is defined as:

$$B_{M_s \cup v} = \int_0^1 B_{M_{s-1}}(x - tv)dt. \tag{11}$$

Thus, each convolution in another direction v increases the degree by 1, and when $s = k - n$, B_M is a piecewise polynomial of degree $k - n$.

Geometric definition Given a vector $u = (u_1, \dots, u_n, \dots, u_k) \in \mathbb{R}^k$, we define an orthogonal projection $\pi : \mathbb{R}^k \to \mathbb{R}^n$ as: $\pi(u) = (u_1, \dots, u_n)$.

For each $1 \leq i \leq k$, let $\{u_1, u_2, \dots, u_k\}$ where each $u_j \in \mathbb{R}^k$ such that $v_j = \pi(u_j)$ and $\{u_1, u_2, \dots, u_k\}$ are linearly independent. We denote by $M' = [u_1\, u_2\, \dots\, u_k]$, the matrix the columns of which are the vectors u_i. Then, the parallelpiped $\beta = [u_1 u_2 \dots u_k][0,1)^k$ is the image of the k-dimensional box $[0,1)^k$ by the linear function associated with M'.

We define a map $\mathcal{C} : \mathbb{R}^n \to \mathbb{R}^k$, called *cylindrification*, as follows: given $x \in \mathbb{R}^n$, $\mathcal{C}_\beta(x) = \{y \in \beta \mid x = \pi(y)\}$. Intuitively, $\mathcal{C}(x)$ is the set of points inside the parallelpiped $\beta \subset \mathbb{R}^k$ that have x the projection on $\mathbb{R}^n$.

The box spline function $B_M(x)$ associated with the vectors $V = \{v_1, \dots, v_k\}$ is defined as:

$$B_M(x) = \frac{vol_{k-s}(\mathcal{C}(x))}{vol_k(\beta)} \tag{12}$$

Basic properties The box splines have the following properties which follow directly from their definition.

1. For all $x \in [v_1\, v_2\, \ldots v_k][0,1)^k$, $B_M(x) > 0$.
2. The support of $B_M(x)$ is
$$Z = [v_1 v_2 \ldots v_k][0,1]^k. \tag{13}$$
that is the sum of the columns in V, or *the zonotope with V as the set of its generators.*
3. The box spline B_M is symmetric with respect to the center of its support.
4. The box spline B_M is *piecewise polynomial.* Indeed, it is a polynomial of degree *at most* $(k-n)$ within each element of the simplicial mesh defined by V over the support of B_M.
5. The box spline B_M is $(\rho-2)$ times continuously differentiable, where ρ is the minimal number of vectors that need to be removed from V so that they do not span $\mathbb{R}^n$.
6. The function B_M *reproduces all polynomials of degree* $(\rho-1)$ with ρ defined as above.
7. The integer shifts of B_M sum to 1, that is,
$$\sum_{\mathbf{i}\in\mathbb{Z}^n} B_M(x-\mathbf{i}) = 1.$$

Again, we assume that the vectors in $V = \{v_1, \ldots, v_k\}$ span $\mathbb{R}^n$. We define an integer shift of the box spline $B_M(x-\mathbf{i})$ with $\mathbf{j} \in \mathbb{Z}^n$. Since $B_M(x)$ is non-negative and the sum of all the integer shifts of $B_M(x)$ is 1, the integer shifts of any box spline $B_M(x)$ form a *partition of unity.*

We are now interested in the space of polynomials Π spanned by all shifts of B_M. We define the *index set* for each $x \in \mathbb{R}^n$ as follows:
$$\mathcal{I}(x) = \{\mathbf{i} \in \mathbb{Z}^n \mid B_M(x-\mathbf{i}) \neq 0\}.$$
Note that this set is finite for any x; therefore, for convenience of notation we write infinite linear combinations of the integer shifts while keeping in mind that the combinations are only over the associated index set.

Lemma 3. *If the box spline B_M is r times continuously differentiable. Then, for any polynomial c of degree $(r+1)$, the function*
$$s(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n} c(\mathbf{i}) B_M(x-\mathbf{i}). \tag{14}$$
is a polynomial of degree $(r+1)$, and any polynomial can be represented as in (14) by choosing M appropriately. The coefficients $c(\mathbf{i})$ are called the control points.

Convex-hull property. It is not hard to see that, given a point x, $s(x)$ lies in the convex hull of the control points corresponding to the index set $\mathcal{I}(x)$:
$$s(x) = conv\{c(\mathbf{i}) \mid \mathbf{i} \in \mathcal{I}(x)\}.$$
This property, called the *convex-hull property*, will be used for our problem of approximating the image of a set by a polynomial map.

5.2 Computing the box spline representation

In this section, we focus on the problem of the Step 2, that is computing the control points $c(\mathbf{i})$ in the representation (14), provided that the box spline B_M must be of appropriate degree. To this end, we derive a symbolic expression of the 'weight' function c, and then determine the index set, that is the set of integer points at which the integer shifts in the box spline representation (14) is non-null.

Control points Any polynomial can be decomposed to a linear combination of monomials, it is possible to compute the control points for each monomial and then combine them. More concretely, if the polynomial π is a linear combination of two monomials m_1 and m_2, i.e. $\pi = km + k'm'$. Then, if $m(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n} c(\mathbf{i})B_M(x-\mathbf{i})$ and $m'(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n} c'(\mathbf{i})B_M(x-\mathbf{i})$, then $\pi(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n}(c(\mathbf{i}) + c'(\mathbf{i})B_M(x-\mathbf{i})$.

We use the extension of the Marsden identity to derive an analytic expression of c. We consider a monomial of the form $x_1^{r_1}\ldots x_n^{r_n}$ where each r_i are non-negative integers, and we denote it as $m_r(x) = x^r$ where $r = (r_1,\ldots,r_n)$ is called the multi-index. We define $r' \prec r$ iff $r \neq r'$ and $\forall i \in \{1,\ldots,n\} : r'_i \leq r_i$. Similarly, the difference $r - r'$ can be defined componentwise, that is $r - r' = (r_1 - r'_1, \ldots, r_n - r'_n)$. Given a monomial m_r, the goal is to determine the weight function c_r such that the monomial can be represented using the box spline B_M, that is, $x^r = \sum_{\mathbf{i}} c(\mathbf{i})B_M(x-\mathbf{i})$. We define the operator

$$\mu : f \to \sum_{\mathbf{i}} B_M(\mathbf{i})f(-\mathbf{i}). \tag{15}$$

The (symbolic) computation of the function c can be done using the following recurrence (see Chapter *Quasi-interpolants and approximation power* of the book [12]):

$$\begin{cases} c_r &= m_r - \sum_{r'\prec r}\mu(m_{r-r'})\,c_{r'} \\ c_{(0,\ldots,0)} &= 1. \end{cases} \tag{16}$$

For a fixed r, we need to compute the expresions of $c_{r'}$ for all $r' \prec r$, and this requires computing the application of the operator μ on them. To do so, as shown in (15) we need to determine the values of the box spline B_M at all the integer points inside its support.

The generation of $\mu(m_{r'})$ for all $r' \prec r$ can be done before starting the recursion (16). The following recursive method for evaluating B_M [**?**] can be used. For a point x which can be represented as $x = Mt$ and let t_v denote the component of t corresponding to the column v of the matrix M. Then,

$$(n-s)B_M(x) = \sum_{v\in V} t_v B_{M\setminus\{v\}}(x) + (1-t_v)B_{M\setminus\{v\}}(x-v) \tag{17}$$

where $M \setminus \{v\}$ is the matrix resulting from removing the column v from M. The base case for the above recurrence corresponds to the square matrix $M =$

$(v_1, \ldots, v_n)$, and in this case

$$B_M(x) = \frac{1}{|detM|} \chi_{M[0,1]^k}(x)$$

where χ is the characteristic function of the set $M[0,1]^k$. Since we only need to evaluate B_M at integer points, and if we additionally choose the matrix M to have integer elements, then the computation of the characteristic function is accurate, and so is the evaluation of the box spline. In other words, for our purposes, the above scheme does not suffer from the numerical instability as if x is a real-valued vector.

Index set As mentioned earlier, when writing the infinite sum, it indeed suffices to consider the integer points in the index set. Given x, the index set $\mathcal{I}(x)$ associated with the box spline B_M is $\mathcal{I}(x) = \{\mathbf{i} \in \mathbb{Z}^n \mid B_M(x - \mathbf{i}) \neq 0\}$. It can be proven that [12]

$$\mathcal{I}(x) = \mathbb{Z}^n \cap (x - M[0,1]^k). \tag{18}$$

The index set for all x inside some set $X \subset \mathbb{R}^n$, denoted by $\mathcal{I}(X)$, is defined as follows:

$$\begin{aligned} \mathcal{I}(X) &= \{\mathbf{i} \in \mathbb{Z}^n \mid \exists x \in X : B_M(x - \mathbf{i}) \neq 0\} \\ &= \mathbb{Z}^n \cap (X \oplus (-M[0,1]^k)). \end{aligned}$$

where $\oplus$ denotes the Minkowski sum. It is not hard to see that if X is a zonotope, then the set $X \oplus (-M[0,1]^k)$ is a zonotope. Since the Minkowski sum of zonotopes, unlike that of polytopes, can be efficiently, it is convenient to over-approximate X by a zonotope, and then the computation of the index set $\mathcal{I}(X)$ amounts to enumerating all the points with integer coordinates inside a zonotope.

5.3 Image Computation Algorithm

We are now ready to apply the above image computation method to compute the following iteration:

$$Y^i = \pi(X^i)$$

where π is a polynomial map and $x_0 \in X_0$. We assume that we have chosen an appropriate matrix M, that is the condition on M so that the associated box spline is of appropriate degree is satisfied. In the following algorithm, the initial set X_0 is assumed to be a polytope. In each iteration, we compute a set of points, the convex hull of which is an over-approximation of the reachable set. It is important to emphasize that in this algorithm we do not need to compute this convex hull, which is computationally expensive.

The algorithm consists of three main steps:

– We first compute a zonotope Z_P over-approximating the current point set P^i using Algorithm 3 which we shall describe in the next section.

Algorithm 2 Reachable set computation

```
$P^0 = vertices(X_0)$
$i = 0$
**repeat**
  $Z_P = zonotopeBound(P^i)$                    /* using Algorithm 3 */
  $Z = Z_P \oplus (-M[0,1]^k)$
  $I = \mathcal{I}(Z)$                          /* computing the index set */
  $P^{i+1} = \emptyset$
  **forall**($\mathbf{i} \in I$)
    $p = c(\mathbf{i})$                         /* computing the control points */
    $P^{i+1} = P^i \cup \{p\}$
  **endforall**
  $i = i + 1$
**until** $i > K_{max}$
```

– To determine the index set associated with Z_P, we first compute the Minkowski sum of Z_P and $M[0,1]^k$ (the latter is the support of B_M). Then, the index set is the set of all the integer points in the resulting sum. We recall that the zonotopic over-approximation in the first step is used to facilitate the computattion of the Minkowski sum.
– To each integer point in the index set, we apply the weight function c to obtain the corresponding control point. By the convex-hull property of the box spline representation, the convex hull of all such control points is thus an over-approximation of the reachable set at the current iteration. Note that the weight function c for the given box spline B_M can be precomputed in a symbolic form, as shown in (16).

Approximation by zonotopes. We now show how to compute the operator *zonotopeBound* in Algorithm 3, that is over-approximating the convex hull of a point set by a zonotope. Let P be a set of a points in $\mathbb{R}^n$. First of all, using the PCA we can find an oriented hyper-rectangle R that contains P. We denote this by $R = boundPCA(P)$ and defer a description of this procedure to Appendix. Let $l_1, \ldots, l_n$ be the side length of R and fix $\eta_1, \ldots, \eta_n$ to be the axes of R. We define $R_{\boldsymbol{\lambda}}$ the rectangle resulting from scaling R by $\boldsymbol{\lambda} \in (0,1)^n$ around its center (or centroid) c.

Given a point x, we next define an special *translation operator* with respect to $R_{\boldsymbol{\lambda}}$, denoted by $\tau(x, R_{\boldsymbol{\lambda}})$. Let H_i denote the hyper-plane which has η_i as its normal and goes through c. Let $proj_i(x)$ is the projection of x on H_i. We define

$$\varDelta_i = \frac{\delta_i}{||proj_i(x) - x||}(proj_i(x) - x)$$

where $\delta_i \in [0, l_i/2]$. Then, the translation operator is defined as follows. If $x \in R_{\boldsymbol{\lambda}}$, then $\tau(x, R_{\boldsymbol{\lambda}}) = c$; otherwise, $\tau(x, R_{\boldsymbol{\lambda}}) = x + \sum_{i=1}^{n} \varDelta_i$.

We extend this operator to a set of points: $\tau(P, R_{\boldsymbol{\lambda}}) = \{\tau(\boldsymbol{p}_i, R_{\boldsymbol{\lambda}}) \mid \boldsymbol{p}_i \in P\}$. Let $(\delta_1, \ldots \delta_n)$ and $\boldsymbol{\lambda}$ be user-defined parameters, then the zonotope that over-approximates $conv\{P\}$ is computed by the following iterative algorithm.

Algorithm 3 *zonotopeBound(P)*: approximation a point set P by a zonotope

```
$Q^0 = P$, $i = 0$
**repeat**
    $R^i = boundPCA(Q^i)$;
    $Q^{i+1} = \tau(Q^i, R^i_{\boldsymbol{\lambda}})$;
    $i = i + 1$
**until** $size(R^i) \leq \varepsilon$
```

The algorithm stops when the size of R^i is sufficiently small, and all the generators associated with each rectangles R^i computed in each iteration will be used to define the over-approximating zonotope. Its center can be the center of R^0.

6 Illustrative example

We have implemented the Bézier spline based algorithm 1 and applied it to two models of a well-known hybrid system benchmark [7, 6]. These models have been developed to study the luminescence mechanism in the Vibrio Fisheri bacteria and methods to control it.

The first model corresponds to one mode of a simplified hybrid system where the continuous dynamics is described by the following multi-affine system:

$$\begin{cases} \dot{x_1} = k_2x_2 - k_1x_1x_3 + u_1 \\ \dot{x_2} = k_1x_1x_3 - k_2x_2 \\ \dot{x_3} = k_2x_2 - k_1x_1x_3 - nx_3 + nu_2 \end{cases} \tag{19}$$

The state variables $\mathbf{x} = (x_1, x_2, x_3)$ represent the cellular concentrations of different species, and the parameters k_1, k_2, n are the binding, dissociation and diffusion constants. The variables u_1 and u_2 are control variables, which respectively represent the plasmid and external source of autoinducer. In [6] the following control law for steering all the states in the rectangle $[1, 2] \times [1, 2] \times [1, 2]$ to the face $x_2 = 2$ was proposed: $u_1(\mathbf{x}) = -10(x_2 + x_1(-1 + 3) - 4x_3)$ and $u_2(\mathbf{x}) = x_1(3 + x_2(-1 + x_3)) - (-2 + x_2)x_3$. This control objective corresponds to the activation of some genes in the system. We consider two cases: with no control (i.e. $u_1 = u_2 = 0$) and with the above control law. Figure 2 shows the projection on x_2 and x_3 of the reachable sets obtained using our algorithm for polynomial systems. In [4] we have already treated this model using an abstraction method based on projection. This method approximates the multi-affine system by a lower dimensional bilinear system. Comparing with the result presented in [4], one can see that our new algorithm for polynomial systems is more accurate, and in addition we have observed that it is also more time-efficient.

The second model is taken from [7]. It is a hybrid model[4] with two modes and one additional continuous variable x_4. The continuous dynamics is $\dot{\mathbf{x}} =$

[4] The numbering of variables is different from that in [7].

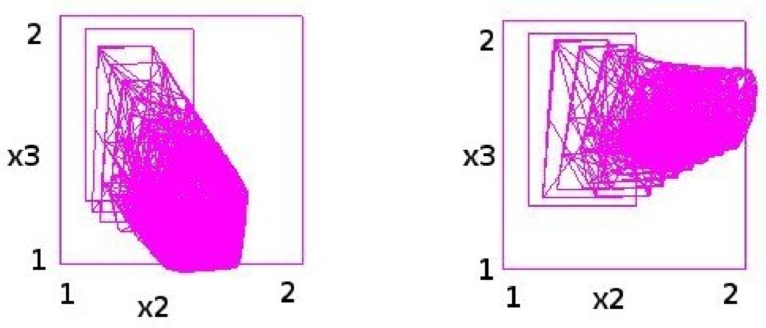

Fig. 2. Reachable sets: with $u_1 = u_2 = 0$ (left) and with the specified control law (right). The control law indeed drives the system to the face $x_2 = 2$.

$A\mathbf{x}+g(\mathbf{x})+b_{ij}$ where b_{01} and b_{10} correspond respectively to the non-luminescent and luminescent modes, and

$$A = \begin{pmatrix} \frac{-1}{H_{sp}} & 0 & 0 & r_{Co} \\ 0 & 0 & 0 & \frac{-1}{H_{sp}} - r_{Co} \\ 0 & x_0 r_{AII} & \frac{-1}{H_{AI}} & x_0 r_{Co} \\ 0 & \frac{-1}{H_{sp}} & 0 & 0 \end{pmatrix}; \; g(\mathbf{x}) = \begin{pmatrix} -1 \\ 1 \\ -x_0 \\ 0 \end{pmatrix} r_{AIR} x_1 x_3$$

We are interested in the question of how to determine the sets of states from which the system can reach the luminescent equilibrium. The condition for switching between the two modes is $x_2 = x_{2sw}$. This problem was also previously studied in [7] using the tool **d/dt**. However, in [7] the multi-affine dynamics was approximated by a 3-dimensional linear system, assuming that x_1 remains constant. Using our algorithm for polynomial systems, we can now handle the non-linearity in the dynamics. In terms of qualitative behavior, the result obtained for the 4-dimensional multi-affine model is compatible with the result for the linear approximate model in [7], that is, from the non-luminescent mode the system can reach the guard to switch to the luminescent mode and then converge to the equilibrium. However, the new result obtained for the 4-dimensional model shows a larger set of states that can reach the equilibrium. This can be explained by the fact that in this model the variable x_1 is not kept constant and can evolve in time.

7 Concluding remarks

In this paper, we presented a new approach to approximate reachability analysis of polynomial systems by combining the ideas from numerical integration and techniques from computer aided geometric design. The reachability algorithm we proposed is of order 2, and these results can be straightforwardly applied to safety verification of hybrid systems. This work opens interesting directions to explore. Indeed, different tools from geometric modeling (such as, splines) could be exploited to approximate polynomial maps more efficiently. In addition, we plan to apply these techniques to verify analog circuit benchmarks proposed in [14].

References

1. R. Alur, T. Dang, and F. Ivancic. Reachability analysis via predicate abstraction. In M. Greenstreet and C. Tomlin, editors, *Hybrid Systems: Computation and Control*, LNCS 2289. Springer-Verlag, 2002.
2. R. Alur, T.A. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 2000.
3. E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 20–31. Springer-Verlag, 2000.
4. E. Asarin and T. Dang. Abstraction by projection. In R. Alur and G. Pappas, editors, *Hybrid Systems: Computation and Control*, LNCS 2993, pages 32–47. Springer-Verlag, 2004.
5. E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, LNCS 2623, pages 20–35. Springer-Verlag, 2003.
6. C. Belta, L. C. G. J. M. Habets, and V. Kumar. Control of multi-affine systems on rectangles with an application to gene transcription control. In *Proceedings of CDC*, 2003.
7. C. Belta, J. Schug, T. Dang, V. Kumar, G.J. Pappas, H. Rubin, and P. Dunlap. Stability and reachability analysis of a hybrid model of luminescence in the marine bacterium *vibrio fisheri.* In *Proceedings of CDC*, 2001.
8. A. Chutinan and B.H. Krogh. Verification of polyhedral invariant hybrid automata using polygonal flow pipe approximations. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 76–90. Springer-Verlag, 1999.
9. Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.
10. T. Dang and O. Maler. Reachability analysis via face lifting. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, LNCS 1386, pages 96–109. Springer-Verlag, 1998.
11. O. Davyrov, M.Sommer, and H.Strauss. On almost interpolation by multivariate splines. In J.W.Schmidt G.Nürnberger and G.Walz, editors, *Multivariate Approximation and Splines*, pages 45–58. ISNM, Birkhäuser, 1997.
12. C. de Boor and K. Höllig and S. Riemenschneider. *Box Splines.* Applied Mathematical Sciences 98, Springer-Verlag, 1993.
13. P. de Casteljau. *Formes à pôles.* Hermes, Paris, 1985.

14. B. Kaminska, K. Arabi, I. Bell, P. Goteti, J.L. Huertas, B. Kim, A. Rueda, and M. Soma. Analog and Mixed-Signal Benchmark Circuits - First Release. In *IEEE International Test Conference*, Washington DC, November 1997.
15. Tony DeRose, Ronald Goldman, Hans Hagen, and Stephen Mann. Functional composition via blossoming. *ACM Transactions on Graphics*, 12(2), April 1993.
16. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.
17. D. R. Fuhrmann. A simplex shrink-wrap algorithm. In *Proceedings of SPIE*. AeroSense, 1999.
18. Jean Gallier. *Curves and surfaces in geometric modeling: theory and algorithms*. Series In Computer Graphics and Geometric Modeling. Morgan Kaufmann, 1999.
19. A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems : Computation and Control*, LNCS 3414, pages 291–305. Springer, 2005.
20. M.R. Greenstreet and I. Mitchell. Integrating projections. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, LNCS 1386, pages 159–1740. Springer-Verlag, 1998.
21. A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 202–214. Springer-Verlag, 2000.
22. S.K. Lodha and R. Goldman. Change of basis algorithms for surfaces in cagd. *Computer Aided Geometric Design*, 12:801–824, 1995.
23. Ian M. Mitchell and Jeremy A. Templeton. A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS. Springer-Verlag, 2005, to appear.
24. 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.
25. Lyle Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6:323–358, 1989.
26. M.-C. Rivara. Mesh refinement process based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21:604–613, 1984.
27. 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.
28. A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. Tomlin and M.R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, LNCS 2289, pages 465–478. Springer-Verlag, March 2002.
29. 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.
30. C. Tomlin, I. Mitchell, A. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.
31. G. Vegter and C. Yap. Minimal circumscribing simplices. In *Proceedings of the 3rd Canadian Conference on Computational Geometry, Vancouver, Canada*, pages 58–61, 1991.

Proof of Theorem 1. From (3) and (4), the local error can be written as:

$$\mathbf{x}(t_k + \tau) - \bar{\mathbf{x}}(t_k + \tau) = \int_0^h e^{A(h-\tau)}[f(\mathbf{x}(t_k + \tau)) - f(\alpha(t_k + \tau))]\,d\tau.$$

On the other hand, due to the Taylor expansion, we have $||\mathbf{x}(t_k+\tau)-\alpha(t_k+\tau)|| \leq M\tau^2$ where M is some constant. We then have $||\, f(\mathbf{x}(t_k+\tau)) - f(\alpha(t_k+\tau))\, || \leq LM\tau^2$ where L is the Lipschitz constant of f. Using the expression (6), we have $\Gamma_2 = \int_0^h e^{A(h-\tau)}\tau^2\, d\tau = \frac{A^3}{3!}h^3 + \mathcal{O}(h^4)$, it then follows that

$$||\mathbf{x}(t_k+\tau) - \bar{\mathbf{x}}(t_k+\tau)|| = \mathcal{O}(h^3).$$

This completes the proof of the theorem. □

Proof of Lemma 2. We consider $p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d)$ where each argument $\mathbf{x}_j$ can be expressed using the barycentric coordinates as: $\mathbf{x}_j = \lambda_1(\mathbf{x}_j)\mathbf{v}_1 + \ldots + \lambda_{n+1}(\mathbf{x}_j)\mathbf{v}_{n+1}$. Due to the property of multi-affine maps, replacing the first argument $\mathbf{x}_1$ with its barycentric coordinates, we have:

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d) = \lambda_1(\mathbf{x}_1)p(\mathbf{v}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n+1}) + \ldots + \lambda_{n+1}(\mathbf{x}_1)p(\mathbf{v}_{n+1}, \mathbf{x}_2, \ldots, \mathbf{x}_{n+1}).$$

We then do the same with other arguments to obtain:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_d) = \sum_{I \in \Xi} \prod_{k \in I_1} \lambda_1(\mathbf{x}_k) \ldots \prod_{k \in I_{n+1}} \lambda_{n+1}(\mathbf{x}_k) p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]}) \tag{20}$$

where Ξ is the set of all partitions of $\{1, 2, \ldots, d\}$ defined as follows. We say that $I = \{I_k\}_{k=1,2\ldots,n+1}$ is a partition of $\{1, 2, \ldots, d\}$ iff all I_k are pairwise disjoint and $\cup_{k \in \{1,\ldots,n+1\}} I_k = \{1, 2, \ldots, d\}$. We write $|I_k|$ to denote the cardinality of I_k. Then, by letting the arguments $\mathbf{x}_i$ to be equal, it is not hard to see that the equation (20) becomes:

$$p(\mathbf{x}, \ldots, \mathbf{x}) = \sum_{||\mathbf{i}||=d} \binom{d}{\mathbf{i}} \lambda_1^{\mathbf{i}[0]}(\mathbf{x})\lambda_2^{\mathbf{i}[1]}(\mathbf{x}) \ldots \lambda_n^{\mathbf{i}[n]}(\mathbf{x}) p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]})$$

Comparing the above with the definition of Bézier simplices (7), it is easy to see that all the points $p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]})$ form the control net of a π whose polar form is p. □

Proof of Theorem 3. Given a multi-index $\mathbf{i}$ with $||\mathbf{i}|| = d$, we consider a point $\mathbf{y} \in \Delta$ which is written as $\mathbf{y} = \sum_{i \in \{1,\ldots,n\}} \frac{\mathbf{i}[i]}{d}\mathbf{v}_i$. We first observe that due to symmetry, $p(\mathbf{x}, \mathbf{y}, \ldots, \mathbf{y}) = p(\mathbf{y}, \mathbf{x}, \ldots, \mathbf{y}) = \ldots = p(\mathbf{y}, \mathbf{y}, \ldots, \mathbf{x})$. Let D denote the partial derivative of these functions at $\mathbf{x} = \mathbf{y}$. Using the Taylor expansion of $p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d)$ around $(\mathbf{y}, \mathbf{y}, \ldots, \mathbf{y})$, we have:

$$\begin{aligned} \mathbf{b}_\mathbf{i} &= p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]}) \\ &= p(\mathbf{y}, \mathbf{y}, \ldots, \mathbf{y}) + \mathbf{i}[1]D(\mathbf{v}_1 - \mathbf{y}) + \ldots + \mathbf{i}[n+1]D(\mathbf{v}_{n+1} - \mathbf{y}) + O(\rho^2) \end{aligned}$$

Note that $\mathbf{i}[0](\mathbf{v}_0 - \mathbf{y}) + \ldots + \mathbf{i}[n+1](\mathbf{v}_{n+1} - \mathbf{y}) = 0$. It then follows that $\mathbf{b}_\mathbf{i} = \pi(\mathbf{y}) + O(\rho^2)$. This means that $||\mathbf{b}_\mathbf{i} - \pi(\mathbf{y})||$ is indeed of order $O(\rho^2)$. □

Computing the oriented bounding box PCA In this section we recall the method for computing an oriented bounding box of a set of points using Principal Component Analysis (PCA). For a thorough description of the PCA, the reader is referred to [**?**].

Let $P = \{p^1, p^2, \ldots, p^k\}$ be a set of k points in $\mathbb{R}^n$. We can assume that $k \geq n$. The axes of the oriented bounding box are determined as the directions along which the points are mostly distributed. More concretely, we use $\bar{p}$ to be the mean of P, that is $\bar{p} = \sum_{i=1}^{k} p^i$ and we denote $\tilde{p}_i, j = p_i^j - \bar{p}_i$. For two points p^i and p^j in P, the covariance of their translated points is:

$$cov(\tilde{p}_i, \tilde{p}_j) = \frac{1}{k-1} \sum_{m=1}^{k} \tilde{p}_i^m \tilde{p}_j^m .$$

Then, we define the co-variance matrix as follows:

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

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 the centroid of the box is $\bar{p}$.