Sparse Tensors and Subdivision Methods for Finding the Zero Set of Polynomial Equations

Guillaume Moroz

Sparse Tensors and Subdivision Methods for Finding the Zero Set of Polynomial Equations

Guillaume Moroz

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France,
guillaume.moroz@inria.fr

Abstract. Finding the solutions to a system of multivariate polynomial equations is a fundamental problem in mathematics and computer science. It involves evaluating the polynomials at many points, often chosen from a grid. In most current methods, such as subdivision, homotopy continuation, or marching cube algorithms, polynomial evaluation is treated as a black box, repeating the process for each point. We propose a new approach that partially evaluates the polynomials, allowing us to efficiently reuse computations across multiple points in a grid. Our method leverages the Compressed Sparse Fiber data structure to efficiently store and process subsets of grid points. We integrated our amortized evaluation scheme into a subdivision algorithm. Experimental results show that our approach is efficient in practice. Notably, our software `voxelize` can successfully enclose curves defined by two trivariate polynomial equations of degree 100, a problem that was previously intractable.

Keywords: Subdivision, sparse tensor, polynomials, root finding

1 Introduction

Subdivision algorithms are widely used to enclose the zero set of a function F ([17, 18, 21, 24, 27] among others). They roughly consist in evaluating F on boxes created along a subdivision tree. If the input function is a high degree polynomial, one of the bottlenecks of those algorithms is the time required to evaluate F. We propose a new approach that amortizes the evaluation cost over the boxes created in a subdivision algorithm. It combines on the one hand partial evaluations of the input polynomial with interval arithmetics, and on the other hand sparse tensors [5, 26] to store the boxes created during the subdivision algorithm. This approach was implemented in the software `voxelize`, and the source code is available on gitlab[1]. Experimental results show that this software can enclose the zero set of polynomial systems that were not reachable with state-of-the-art software.

After giving an overview of our main results in the introduction, we present in Section 2.1 the Compressed Sparse Fiber data structure and we show in Section 2.2 how it can be used to evaluate efficiently a polynomial on a subset

[1] `https://gitlab.inria.fr/gmoro/voxelize`

of a grid of boxes. Then in Section 3.1, we show how our new evaluation scheme yields a quasi-linear time algorithm to compute a discrete Fourier transform. We show in Section 3.2 how to integrate our evaluation scheme into a subdivision algorithm to enclose the zero set of a polynomial system. Finally, in Section 4, we present the timing results of `voxelize` on several polynomial systems, including random polynomial systems (Section 4.1), and systems coming from applications (Section 4.2).

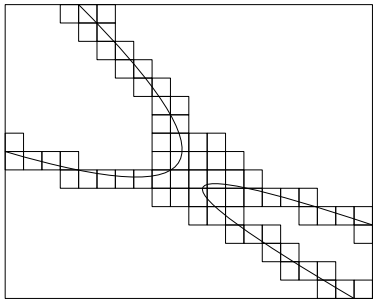

Fig. 1. Boxes on the same level of the subdivision tree

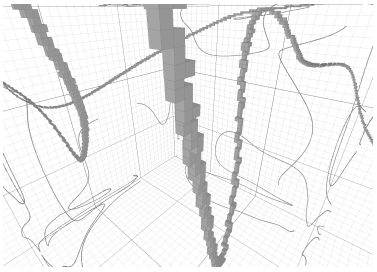

Fig. 2. Enclosing of a curve defined by 2 trivariate polynomials of degree 100

1.1 Amortized Evaluation on a Grid of Boxes

The first idea to reduce the evaluation redundancies is to use partial evaluation. Assume that $F(x_1, x_2)$ is a bivariate polynomial of degree d. Moreover, let $(I_i)_{0\leq i<n}$ and $(J_i)_{0\leq j<n}$ be two sequences of real intervals. Using the Hörner scheme, evaluating F on a box requires $O(d^2)$ arithmetic operations, and evaluating F on all the boxes $I_i \times J_j$ for $0 \leq i, j < n$ requires $O(d^2n^2)$ arithmetic operations. By reorganizing the operations using partial evaluations, the number of arithmetic operations can be reduced to $O(dn(d+n))$. This idea is well known and was used for example to speed up the multiplication of polynomials [23]. It is also currently implemented in the well-spread library `NumPy` to evaluate polynomials in 2 and 3 variables [9].

More precisely the operations are reordered as follows. For a given I_i, the partial evaluation of F in I_i results in a univariate polynomial f_i of degree d. This step requires $O(d^2)$ arithmetic operations. Then evaluating f_i on n intervals requires $O(dn)$ arithmetic operations. Finally, repeating these operations for all the n intervals I_i, this allows us to evaluate F on all the boxes of the grid with a total number of arithmetic operations in $O(dn(d+n))$. More generally, for higher dimensions, this leads to the following result.

Property 1 ([23]). Let F be a polynomial in k variables and of degree at most $d-1$ in each variable. Let $X_1, \ldots, X_k$ be k sets of n real intervals each. Then it is

possible to evaluate F on all the boxes of $X_1 \times \cdots \times X_k$ in $O(kdn \max(n,d)^{k-1})$ arithmetic operations.

In the case where $n > d$, this approach results in a significant speedup since the amortized number of arithmetic operations to evaluate F on each box of the grid is $O(kd)$ instead of $O(d^k)$.

1.2 Amortized Evaluation on a Sparse Subset of a Grid

For the simple subdivision algorithm mentioned at the beginning of the introduction, if the boxes created are never discarded, then each level of the subdivision tree forms a dense grid of boxes. In this case, the partial evaluation approach shown in the previous section can be applied directly to reduce the total number of arithmetic operations required to evaluate F on each box with interval methods. In the general case though, many boxes are discarded, and the boxes appearing in a given level of the subdivision tree form a subset of a grid, as shown in Figure 1. The boxes created in the subdivision algorithm can be handled in different orders. Using a breadth-first walk on the subdivision, the boxes on the same level are a subset of a grid. In this case, we need to evaluate a polynomial on a sparse subset of a grid.

To evaluate a polynomial on a general set of points, the case of a univariate polynomial is well understood [6, 10, 16, 20]. For multivariate polynomials, there are fewer results that are efficient in practice when the points are not arranged as a grid. A breakthrough, that was recently improved, is a quasi-linear algorithm to evaluate a polynomial of degree d in k variables on d^k points in a finite field [1,2,12,19,28]. For multipoint evaluation with real numbers, the only subquadratic algorithms are for bivariate polynomials [22], or require precomputation more than quadratic in the number of points [13, 14]. Finally, a recent work addresses the case of approximate numerical evaluation [7]. Unfortunately, those approaches are not yet efficient in practice. Our main result is a practical improvement to amortize multipoint evaluations in the case were the points or boxes that we consider are a sparse subset of a grid.

Boxes in a sparse subset of a grid can be gathered and stored as a sparse tensor in the Compressed Sparse Fiber (CSF) format [5, 26]. The CSF is a generalization of the Compressed Row Format used to store the entries of a sparse matrix. Then, F can be evaluated efficiently on these boxes (see Section 2.2 for more details). This approach was implemented in the library `voxelize`. Figure 2 shows the output boxes of the software `voxelize` enclosing an algebraic curve defined by two polynomial equations of degree 100, where the coefficients are randomly drawn from a normal law centered at zero. Performing the partial evaluation approach on a set of boxes in a CSF format leads to Theorem 1.

1.3 Notations

For a set E, we denote by $|E|$ its number of elements. Then, we define the notations for the size of the projection of a subset E of a grid. In particular, the

size of the projection is smaller when the elements of E are aligned within the grid.

Definition 1. *Given a finite set $E \subset \mathbb{N}^k$, and an integer i between 1 and k, we denote by $N_i(E)$ (resp. $\widetilde{N}_i(E)$) the number of elements in the projection E on the first (resp. last) i coordinates, counting repeated projections only once.*

Even though this definition holds for a set of integer tuples, it can be naturally extended for multivariate polynomials. Indeed, for each monomial, we can associate its vector of exponents. If F is a polynomial in k variables, for a given integer i, we can define $N_i(F)$ (resp. $\widetilde{N}_i(F)$) as the size of the projections of the set of vectors of exponents of F to their first (resp. last) i coordinates.

For S a set of points or boxes that is a subset of a grid, we can also extend the definition of N_i by simply indexing the elements of S by their integer positions in the grid. Letting S_{ind} be the set of integer indices of the boxes of S, we can define $N_i(S)$ by $N_i(S_{ind})$.

1.4 Main Result

We can now state our main theorem to evaluate a multivariate polynomial on a set of boxes that is a sparse subset of a grid of boxes G that is the Cartesian product of k sets of intervals $X_1 \times \cdots \times X_k$.

Theorem 1. *Let F be a polynomial in k variables, and S be a subset of boxes of G. It is possible to evaluate F on all the boxes of S in $O(\sum_{i=0}^{k-1} \widetilde{N}_{k-i}(F)N_{i+1}(S))$ arithmetic operations.*

When the set of boxes enclose a variety of dimension j,the projection of S on the first j coordinates is often a dense grid. In this case, we have the following corollary.

Corollary 1. *For $1 \leq j \leq k-1$, assume that the projection of S on the first j coordinates is:*

i. a dense grid, denoted by $X_1 \times \cdots \times X_j$
ii. $|X_i| > d$ for all $1 \leq i \leq j$.

Then we can evaluate each box of S in $O(j(d+1)^{k-j+1})$ arithmetic operations on average, instead of $O((d+1)^k)$ operations.

Proof (Corollary 1). First, if F has degree at most d in each variable, then $\widetilde{N}_{k-i}(F)$ is less than d^{k-i} for all non-negative integers less or equal to k.

For $0 \leq i < j$, Assumption i implies that $N_{i+1}(S) = N_i(S)|X_{i+1}|$. Then we deduce with Assumption ii. that $(d+1)N_i(S) \leq N_{i+1}(S)$. This implies that:

$$\begin{aligned}\widetilde{N}_{k-i}(F)N_{i+1}(S) &\leq (d+1)^{k-i}N_{i+1}(S)\\ &\leq (d+1)^{k+1-j}\\ N_j(S) &\leq (d+1)^{k+1-j}|S|.\end{aligned}$$

For $i \geq j$ we have $\widetilde{N}_{k-i}(F) \leq (d+1)^{k-i}$, such that $\widetilde{N}_{k-i}(F)N_{i+1}(S) \leq (d+1)^{k-i}|S|$. Thus, the evaluation of F on all the boxes of S is in

$$O(j(d+1)^{k-j+1}|S| + \sum_{i=j}^{k-1}(d+1)^{k-i}|S|) = O(j(d+1)^{k-j+1}).$$

In particular, the amortized cost of evaluating each box is in $O(jd^{k-j+1})$ arithmetic operations instead of $O(d^k)$ with a direct algorithm.

2 Evaluating Polynomials with Compressed Sparse Fibers

2.1 Sparse Tensor Data Structure

The main data structure used in our algorithms is the Compressed Sparse Fiber, as described in [5,26]. This data structure is well suited to store a subset of a grid in high dimension. It can be seen as a generalization of the classical Compressed Sparse Row data structure used to store the entries of a sparse matrix as in Figure 3.

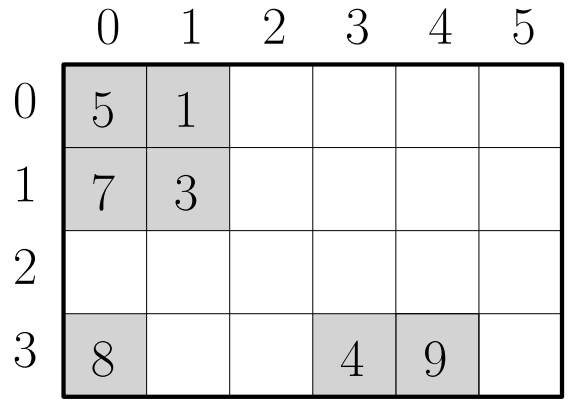

Fig. 3. Numbers stored in a sparse matrix

For a subset of a 2D grid, the data structure is a labeled tree that stores the positions of the non-empty rows in the children of the root node, and then in each row, the position of the non-empty entries are stored in the children of the corresponding node (Figure 4). In higher dimension k, this idea is applied recursively. Let E be a subset of points in $\mathbb{N}^k$. For $t = (t_1, \ldots, t_\ell) \subset \mathbb{N}^\ell$ a tuple of size $\ell < k$, we denote by $\pi_t(E)$ the subset of $\mathbb{N}$ defined by:

$$\pi_t(E) = \{i \in \mathbb{N} \mid \exists y_{\ell+2}, \ldots y_k \in \mathbb{N} \text{ such that } (t_1, \ldots, t_\ell, i, y_{\ell+2}, \ldots, y_k) \in E\}.$$

Then the Compressed Sparse Fiber (or CSF) data structure associated to E is a labeled tree of depth k defined recursively as follows. The root of the tree

is at depth 0 and its children are the nodes labeled by the elements of $\pi_\emptyset(E)$, where $\emptyset$ denotes the empty tuple. Consider now a node N of the tree at depth $1 \leq \ell < k$. Let $t(N)$ be the tuple of size ℓ, where the i-th coordinate is the label of the i-th node on the path from the root to N. Then the children of N are the nodes labeled by the elements of $\pi_{t(N)}(E)$. Finally, for a node N at depth k, it is possible to add a leaf that can be labeled with the value of the entry associated to the tuple $t(N)$. Given a CSF data structure, the corresponding set of tuple E is unique and is called its *support*.

As an example, using the compressed sparse data structure to store the sparse matrix given in Figure 3, we get the tree shown in Figure 4, and its support is $\{(0,0),(0,1),(1,0),(1,1),(3,0),(3,3),(3,4)\}$.

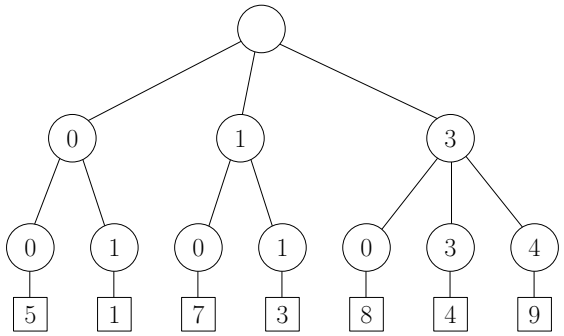

Fig. 4. Compressed Sparse Fiber associated to the sparse matrix

Remark 1. Given T a CSF data structure associated to E, remark the size of the projection on the first i coordinates is the number of nodes of depth i in T. In particular, we have $N_i(E) = \sum_{t \in \mathbb{N}^{i-1}} |\pi_t(E)|$.

Representing a Multivariate Polynomial. A natural application of the Compressed Sparse Fiber data structure is to encode the monomials of a sparse polynomial. Given a polynomial in k variables, the exponents of each monomial can be represented as a k-tuple of integers in $\mathbb{N}^k$, and the coefficients can be represented as the entries associated to each tuple. Using this representation, it can be directly encoded in a CSF data structure. Given a polynomial $F(x_1, \ldots, x_k)$, we denote by T_F the CSF tree associated to F. By extension of Definition 1, we define the size $N_i(F)$ as the size $N_i(T_f)$ of its corresponding CSF tree truncated to depth i.

For example the following polynomial in two variables would be encoded with the CSF tree in Figure 4:

$$5 + x_2 + 7x_1 + 3x_1x_2 + 8x_1^3 + 4x_1^3x_2^3 + 9x_1^3x_2^4$$

Representing a Set of Boxes. A sparse subset of a grid of boxes can also be represented with the Compressed Sparse Fiber data structure, by applying it to the indexes of the boxes within the grid. Without loss of generality, consider a subdivision of the unit box $[0,1]^k$ into n^k smaller boxes, where each box is a product of intervals of the form $\prod_{i=1}^{k}[a_i, b_i]$, where a_i and b_i are real numbers. In the case where the subdivision is uniform, let G_n be the set of these n^k boxes. Each cube from G_n can be indexed by a k-tuple of integers in $\mathbb{N}^k$. In particular, for a sparse subset of G_n, we can associate the set B of the indices of its boxes. Then, B can be encoded in a CSF data structure. In this case, the tree we construct won't have leaves since there is no entry associated to each box.

2.2 Evaluation Algorithm

One Variable. A classical way to evaluate a univariate polynomial on a point is the Hörner algorithm that we recall in Algorithm 1 for the evaluation of a sparse polynomial on an interval.

Algorithm 1: Hörner algorithm

Input: An interval I and a polynomial $F(x) = a_0 x^{e_0} + \cdots + a_\ell x^{e_\ell}$ where:
$e_0 < \cdots < e_l$ are integers
$a_0, \ldots, a_\ell$ are real numbers or intervals.
Output: The interval obtained by evaluating F on I with the Hörner scheme.

```
J ← a_l
for j from ℓ − 1 to 0 do
    J ← J × I^(e_{j+1} − e_j) + a_j
return J
```

Several Variables. For multivariate polynomials F, we can use the Hörner scheme recursively. Moreover, if we want to evaluate F on a set of boxes, Algorithm 2 generalizes the Hörner scheme to the case where F and the boxes are stored in a CSF data structure. The key idea in Algorithm 2 is that for boxes that share the same coordinate, we only evaluate the polynomial partially on those coordinates. Then we reuse those partially evaluated polynomials to evaluate the boxes on the remaining coordinates.

The advantage of using the approach in Algorithm 2 is that it allows us to amortize the cost of the evaluation when several boxes have the same projection. In the following, we will prove that the complexity of Algorithm 2 is in $O(\sum_{i=0}^{k-1} \widetilde{N}_{k-i}(F) N_{i+1}(S))$ arithmetic operations, which will prove Theorem 1.

Proof (Theorem 1). Since Algorithm 2 is recursive, we will prove its complexity by recurrence. Algorithm 2 is a loop over the nodes of the root of T. In particular,

Algorithm 2: Evaluation on a set of boxes

Input: F a polynomial in k variables
T a CSF tree representing the indices of a subset S of boxes of a grid $X_1 \times \cdots \times X_k$, where X_i is a set of intervals.

Output: A CSF data structure representing the evaluation of the polynomial represented by F on all the boxes in S.

```
1  Function EvaluationCSF(F,T):
2      X_1 ← the list of intervals of the first coordinate in the grid G
3      L ← empty list
4      for i in π_∅(T) do
5          I ← the interval of index i in X_1
6          F_I ← F(I, x_2, ..., x_n)
7          if F is univariate
8              Append F_I to L
9          else
10             T_i ← the subtree of T rooted at the node at depth 1 with label i
11             L_i ← EvaluationCSF(F_I, T_i)
12             Append L_i to L
13     return L
```

this loop will be called $N_1(S)$. In each loop, the dominating complexities are in line 6 and 11. In line 6, the complexity of evaluating partially F in one variable x_1 is $\widetilde{N}_k(F)$. Thus, the total complexity carried by line 6 is in $O(\widetilde{N}_k(F)N_1(S))$. And if F is univariate, the complexity of Algorithm 2 is in $O(\widetilde{N}_1(F)N_1(S))$.

Then, if F is a polynomial in k variables with $k > 1$, the number of operations is again carried by lines 6 and 11. Let S_I be the set of boxes represented by the tree T_I. By recurrence the number of operations in line 11 is in

$$O\left(\sum_{i=0}^{k-2} \widetilde{N}_{k-1-i}(F_I)N_{i+1}(S_I)\right).$$

In particular, remark that $\widetilde{N}_{k-1-i}(F_I) = \widetilde{N}_{k-1-i}(F)$. And using Remark 1, the sum of the $N_{i+1}(S_I)$ on all the intervals I children of the root of T is equal to $N_{i+2}(S)$. Thus, the complexity of Algorithm 2 carried by line 11 is $O\left(\sum_{i=0}^{k-2} \widetilde{N}_{k-1-i}(F)N_{i+2}(S)\right)$. By changing the index of the sum, this complexity becomes

$$O\left(\sum_{i=1}^{k-1} \widetilde{N}_{k-i}(F)N_{i+1}(S)\right).$$

Since the complexity carried by line 6 is $O(\widetilde{N}_k(F)N_1(S))$, this concludes the proof.

3 Applications

3.1 The Fast Fourier Transform Revisited

Given a vector u of $d+1$ complex numbers $u_0, \ldots, u_d$, its discrete Fourier Transform is the vector v of $d+1$ complex numbers $v_0, \ldots, v_d$ such that:

$$v_k = \sum_{j=0}^{d} u_j e^{-i2\pi \frac{k}{d+1} j} \tag{1}$$

The fast Fourier Transform algorithm returns the vector v using $O(d \log d)$ arithmetic operations. If we reinterpret Equation (1) as the evaluation of a multivariate polynomial on a set of points stored with a CSF tree data structure, we can use Algorithm 2 to compute the discrete Fourier transform in $O(d \log d)$ arithmetic operations.

Without restriction of generality, assume that there exists an integer k such that $d+1 = 2^k$ is a power of two. Let F be the polynomial in k variables defined by:

$$F = \sum_{(i_1,\ldots,i_k)\in\{0,1\}^k} u_{i_1+\cdots+i_k 2^{k-1}} x_1^{i_1} \cdots x_k^{i_k}$$

Moreover, let w be the $(d+1)$-th root of unity $e^{-i2\pi/(d+1)}$. For $1 \leq j \leq k$, let $X_j = \{1, w^{2^{k-j}}\}$, and let G be the grid of points $g_{i_1,\ldots,i_k}$ in $\mathbb{C}^k$ for $(i_1, \ldots, i_k) \in \{0,1\}^k$, defined by:

$$g_{i_1,\ldots,i_k} = (w^{i_1 2^{k-1}}, \ldots, w^{i_k}) \in X_1 \times \cdots \times X_k$$

Then, using the notations of Equations (1), for an integer $j = i_1 2^{k-1} + \cdots + i_k$ we have $v_j = F(g_{i_1,\ldots,i_k})$. The polynomial F has a degree at most 1 in each variable and the set of points on which F is evaluated is a Cartesian product $X_1 \times \cdots \times X_k$ where X_j has size 2 for all $1 \leq j \leq k$. Then, using Claim 1, this evaluation can be done using $O(k2^k)$, that is $O(d \log d)$ arithmetic operations.

3.2 Subdivision Algorithm

A classical approach to find the zero locus of a set of a polynomial equation is to use a subdivision algorithm. Given a polynomial equation F and a box B, assume that we have two criteria $C_0(F, B)$ and $C_1(F, B)$ such that:

- if $C_0(F, B)$ is true, then F doesn't vanish in B
- if $C_1(F, B)$ is true, then F vanishes in B

The idea of a subdivision algorithm is to start with a set of boxes, and to bisect them recursively until the criterion C_0 is true, or C_1 is true and the size is smaller than a given threshold. Recall that the grid G_n is the set of n^k boxes obtained by subdividing uniformly $[0, 1]^k$ in n boxes in all the directions. Given a box B from the grid G_n, if we bisect it uniformly in 2 in all the directions, we

end up with a set of 2^k boxes, all of them included in G_{2n}. In particular, if we bisect a set of boxes in G_n, we end up with a set of boxes in G_{2n}. Moreover, if the criteria C_0 and C_1 are based on polynomial evaluations, we can use Algorithm 2 to amortize the evaluation. This leads to Algorithm 3, that computes a set of boxes that enclose the zero-set of a polynomial equation. If we want to compute the zero set of a system of polynomial equations and inequalities, Algorithm 3 can be used unchanged, and the criteria C_1 and C_0 can be easily adapted to detect if a system of equations has solutions or not in a given box. To ensure that Algorithm 3 terminates, it is necessary that for boxes small enough, either criterion C_0 or C_1 succeed.

Algorithm 3: Simple subdivision algorithm to enclose the zero locus of a polynomial equation

Input: F a multivariate polynomial
ε a positive threshold real number
Output: A CSF data structure representing the boxes of size at most ε, such that F vanishes in all the boxes, and doesn't vanish outside the boxes.

```
1  S ← {[0,1]^k}
2  R ← {}
3  size ← 1
4  while S is not empty do
5  |  S ← set of boxes B in S not satisfying C0(F,B)
6  |  if size < ε
7  |  |  R ← R union the set of boxes B in S satisfying C1(F,B)
8  |  |  S ← set of boxes B in S not satisfying C1(F,B)
9  |  S ← set of boxes bisected from the boxes in S
10 |  size ← size/2
11 return R
```

Criteria for Exclusion and Inclusion

Exclusion Criterion. A simple exclusion criterion $C_0(F, B)$ consists in evaluating F on B using interval arithmetic. Interval arithmetic is the generalization of standard arithmetic operations to the case where numbers are replaced by intervals. If $[a, b]$ and $[c, d]$ are two intervals, the result of $[a, b] + [c, d]$ is the interval $[a + c, b + d]$. If F is a polynomial in k variables and B is a product of k intervals, we denote by $\square F(B)$ the interval returned when F is evaluated on B using interval arithmetic. The main property of interval arithmetic is that the interval $\square F(B)$ satisfies $\{F(x) \mid x \in B\} \subset \square F(B)$. In particular, if $0 \notin \square F(B)$, then F does not vanish in B. Thus, we can define $C_0(F, B)$ as the predicate $0 \notin \square F(B)$.

The exclusion criterion can also be computed using other schemes to evaluate F on B, such as the Taylor form, which can reduce the overesetimation near the zeros of F [11, §3.5].

Definition 2 (Taylor Form [25, Definition 3.3], [15]). *If c is the middle point of B, for a given integer m, the Taylor form of order m of the polynomial F in k variables is defined by:*

$$T_m(F, x) = F(c) + \cdots + \frac{F^{(m-1)}(c)}{(m-1)!}(x-c)^{m-1} + \frac{\square F^{(m)}(B)}{m!}(x-c)^m$$

where $x = (x_1, \ldots, x_k)$ is a tuple of symbolic variables.

This evaluation scheme satisfies the property $\{F(x) \mid x \in B\} \subset T_m(F, B)$, such that $0 \notin T_m(F, B)$ implies that F does not vanish in B. In the case of a system of several equations, we can simply test if any of the input polynomial does not contain 0.

Inclusion Criterion. For the inclusion criterion $C_1(F, B)$ to detect if F vanishes in B, a simple test consists in evaluating F on all the vertices of B and returning `true` if two of them have different signs, and `false` if all the signs are the same. Remark that the set of all the vertices of all the boxes are a subset of a grid, and thus we can also use Algorithm 2 to amortize the cost of their evaluation.

The inclusion criterion $C_1(F, B)$ can also be based on the Taylor form if we computed it with order m, where m is an integer greater or equal to 2. Let $\ell(x)$ be the linear part of $T_m(F, x)$. Let V_{min} be a vertex of B that minimizes ℓ and V_{max} one that maximizes ℓ. Then we can reduce the evaluation of F to the vertices V_{min} and V_{max}. We can also use the Taylor form to evaluate lower and upper bounds of the values of F at V_{min} and V_{max}. In this case, our predicate will return `true` if the lower bound on $F(V_{max})$ is positive and the upper bound on $F(V_{min})$ is negative.

If F is a vector of multiple polynomials, and if we want to test if they vanish simultaneously inside a box, we can use a criterion C_1 derived from the Newton Interval criterion [8, 21]. First, when the number of input equations $F_1 = 0, \ldots, F_k = 0$ is equal to the number of variables, the Newton Interval criterion can be seen as a fixed-point theorem. Letting S be the $k \times k$ matrix defined by

$$S_{ij} = \begin{cases} \frac{F_i(x_1,\ldots,x_{j-1},x_j,c_{j+1},\ldots,c_k) - F_i(x_1,\ldots,x_{j-1},c_j,c_{j+1},\ldots,c_k)}{x_j - c_j} & \text{if } x_j \neq c_j \\ \frac{dF}{dx_j}(x_1, \ldots, x_{j-1}, c_j, \ldots, c_k) & \text{if } x_j = c_j \end{cases},$$

and c be the center of the box B, we define the formula $N(x) = c - S(x)^{-1}F(c)$. If $N(B) \subset B$, the fixed-point theorem ensures that there exists a point x_0 in B such that $N(x_0) = x_0$, which is equivalent to $F_1(x_0) = 0, \ldots, F_k(x_0) = 0$. Otherwise, when the number of polynomial equations is less than the number of variables, we can intersect the box with the linear space spanned by the gradient

vectors of the input polynomial at the center of the box B. Then we can use the Newton Interval criterion on the resulting system that has as many equations as variables.

4 Experiments

Algorithm 2 and 3 have been implemented in `C++` in the software `voxelize`. This software can take as input a list of polynomial equations and polynomial inequalities, and it returns a list of boxes enclosing the set of points where the input system has solutions. Furthermore, if the input is a single polynomial equation, then it is guaranteed to vanish in each box returned by `voxelize` that are larger than a threshold given by the user. The software can be used as a standalone program, taking one file per polynomial, or it can be used through a python interface.

The criterion C_0 used to exclude boxes is based on the Taylor form evaluation scheme described in Definition 2. The criterion C_1 is implemented in the case where the input is a single polynomial equation, and it follows the approach based on the Taylor form detailed at the end of Section 3.2. In the case of multiple input polynomial equations, the subdivision process stops when the boxes are smaller than a threshold given by the user.

4.1 Random Polynomials

In Table 1, we show the time to enclose the zero-set of polynomial equations in k variables where k is either $2, 3$ or 4. In each case, we consider three cases: a hypersurface defined by one equation, a curve defined by $k-1$ equations, points defined by k equations. And for each case, we generated random polynomials of total degree either 20 or 100, except for $k = 4$ where `voxelize` could not handle polynomials in 4 variables and total degree 100. The random coefficients are floating-point numbers with double precision uniformly sampled between -10 and 10.

The computation have been done on a laptop with a 1.9GHz CPU and 16G of RAM. The tests have been done with one thread, for easier comparison with other single-thread programs. Note that `voxelize` is also implemented with the multi-thread library `openmp` and it can distribute the computations on several threads. Up to our knowledge, `voxelize` is the only available software that can handle the systems with polynomials of degree 100 in 3 variables presented in Table 1.

4.2 Polynomials Coming from Applications

We also used the software on two polynomial systems coming from robotics and automatic applications. In these cases, we compared our software with the state-of-the-art subdivision software `ibex`. The `ibex` software is a general subdivision software including a specific feature called contractors [3]. A contractor is an

Table 1. Timing in seconds for computing enclosing boxes in the cube $[-2,2]^k$. For points and curves, the subdivsion process stopped for boxes smaller than $2^{-8} \simeq 0.004$. For hypersurfaces, the subdivision process stopped when either the criterion C_0 or C_1 was satisfied on all the boxes, and the boxes had a size smaller than $2^{-5} \simeq 0.03$.

dimension k	**2D**		**3D**		**4D**
degree d	**20**	**100**	**20**	**100**	**20**
points[2] (k equations)	0.006	0.32	0.5	273	56
curves[2] ($k-1$ equations)	0.062	0.31	1.3	270	91
hypersurfaces (1 equation)	0.062	0.31	1.1	412	373

[2] Only the exclusion criterion was implemented for this case, and not the inclusion criterion.

operator that takes as input a function F and a box B, and that returns a smaller box B' such that the intersection of B' with the zero set Z of F is the same as the intersection of B with Z.

Robotics. In robotics, a classical problem is to compute the parallel singularities of a robot. That is the set of control parameters around which the robot can be assembled in two nearby configurations. In particular, the following set of equations defines the singularities in the orientation space of the 3-PPPS manipulator [4]. The orientation space is modeled with 4 quaternion variables, commonly used to parametrize the rotation matrices in $3D$. The sum of the squares of the quaternion variables is constrained to be 1.

$$(R)\begin{cases} 0 &= -6\,{Q_2}^2 Q_3 Q_1 + 6\,Q_2 {Q_3}^2 Q_4 + 3\sqrt{3} {Q_2}^2 Q_3 Q_4 \\ &\quad - 6\,Q_2 {Q_1}^2 Q_4 + 6\,Q_1 {Q_4}^2 Q_3 \\ &\quad - 3\sqrt{3} Q_2 Q_1 {Q_4}^2 + 3\sqrt{3} Q_2 {Q_3}^2 Q_1 \\ &\quad - 3\sqrt{3} Q_3 {Q_1}^2 Q_4 + \sqrt{3} {Q_2}^3 Q_1 \\ &\quad - \sqrt{3} Q_2 {Q_1}^3 + Q_4 \sqrt{3} {Q_3}^3 - Q_3 \sqrt{3} {Q_4}^3 \\ 1 &= Q_1^2 + Q_2^2 + Q_3^2 + Q_4^2 \end{cases}$$

Automatic. In control theory, a common problem is to decide if it is possible to add a controller to a dynamic system such that it becomes stable. In some case, this problem can be reduced to decide if a polynomial system does not vanish on complex numbers of modulus less than one. For example, the following system in 3 complex variables was communicated by Thomas Cluzeau and Alban Quadrat. If it has no solution where z_1, z_2 and z_3 have a modulus less than 1, then it is possible to design a stable controller for the corresponding dynamic system.

$$(A)\begin{cases}
|z_1| \leq 1\\
|z_2| \leq 1\\
|z_3| \leq 1\\
0 = z_1z_2^2 - z_1z_3 - 2\\
0 = 12z_2^3z_3^3 - 2z_1^2z_2z_3^2 + z_2^3z_3^2 - 2z_2^2z_3^3 - 12z_2z_3^4 + 2z_1^2z_3^2 - z_2z_3^3\\
\quad -2z_1z_2z_3 - 7z_2^3 - 10z_2^2z_3 + 14z_1z_3 - 8z_2^2 + 9z_2z_3 + 12z_3^2 + 30z_3 + 2\\
0 = z_1^3z_3^3 + z_1z_2z_3^4 - z_1^3z_3^2 + z_1z_3^4 - 12z_2^2z_3^3 - 6z_1^2z_2z_3 + 3z_1^2z_3^2 - z_1z_2z_3^2\\
\quad -z_2^2z_3^2 - 10z_2z_3^3 - 7z_1^2z_3 - 12z_1z_2z_3 - 2z_1z_3^2 - z_2z_3^2 + 2z_3^3 - z_1z_2\\
\quad -9z_1z_3 + 7z_2^2 + 10z_2z_3 - z_1 + 15z_2 + 8z_3 + 8\\
0 = z_1^3z_2z_3^2 - z_1^3z_3^2 + z_1z_3^4 + z_1^2z_2z_3 - 12z_2z_3^3 - 7z_1^2z_3 - z_1z_2z_3 - z_1z_3^2\\
\quad -z_2z_3^2 + 2z_3^3 - 11z_1z_3 - z_1 + 7z_2 + 10z_3 + 8
\end{cases}$$

By using the change of variable $z_j = x_j + iy_j$, we get 8 polynomial equations in 6 variables, with the additional inequalities $x_i^2 + y_1^2 \leq 1$.

Experiences. We used `voxelize` and `ibex` on those two system of polynomial equations and inequalities. The timings and the number of boxes returned for the two software are presented in Table 2.

Table 2. Subdivision solvers used to enclose the zero-set of the systems (R) and (A). For the system (R), the subdivision process was stopped when boxes were smaller than $2^{-4} \simeq 0.06$, both in `ibexsolve` and `voxelize`.

Software	**Robotics** (R)		**Automatic** (A)	
	Time	**Number of boxes**	**Time**	**Number of boxes**
`ibexsolve`	103s	29871	2.5s	0
`voxelize`	0.1s	7228	1.2s	0

We can see that both solvers could detect that the system (A) has no complex solutions of moduli less than 1. In both cases, `voxelize` was faster than `ibexsolve`, and significantly faster for the system (R). This shows that the amortized evaluation scheme based on the CSF data structure is efficient not only in theory, but also in practice. On the other hand, `ibexsolve` and `voxelize` solve the system (A) with a time within the same order of magnitude, despite the fact that `ibexsolve` does not used amortized evaluations. This might be due to the fact that the contractors used by `ibexsolve` work well for this system. Remark that it could be possible to combine contractors and amortized evaluation scheme. The main issue is that after applying a contractor, the boxes are not anymore aligned on a grid. This could be solved by snapping the boxes to expanded boxes from a refined grid after applying the contractors.

Acknowledgments. The author wishes to thank Luc Jaulin, Thomas Cluzeau and Alban Quadrat for their insightful remarks and examples discussed in this article.

References

1. Bhargava, V., Ghosh, S., Guo, Z., Kumar, M., Umans, C.: Fast multivariate multipoint evaluation over all finite fields. In: 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS). pp. 221–232. IEEE Computer Society, Los Alamitos, CA, USA (nov 2022). https://doi.org/10.1109/FOCS54457.2022.00028, https://doi.ieeecomputersociety.org/10.1109/FOCS54457.2022.00028
2. Bhargava, V., Ghosh, S., Kumar, M., Mohapatra, C.K.: Fast, algebraic multivariate multipoint evaluation in small characteristic and applications. In: Proceedings of the 54th annual ACM SIGACT symposium on theory of computing, STOC '22, Rome, Italy June 20–24, 2022, pp. 403–415. New York, NY: Association for Computing Machinery (ACM) (2022). https://doi.org/10.1145/3519935.3519968
3. Chabert, G., Jaulin, L.: Contractor programming. Artif. Intell. **173**(11), 1079–1100 (2009). https://doi.org/10.1016/j.artint.2009.03.002
4. Chen, C., Gayral, T., Caro, S., Chablat, D., Moroz, G., Abeywardena, S.: A six-dof epicyclic-parallel manipulator. Journal of Mechanisms and Robotics **4**(4) (Apr 2012). https://doi.org/10.1115/1.4007489, https://hal.science/hal-00684803, https://hal.science/hal-00684803/file/MEPaM-JMR-FINAL.pdf
5. Chou, S., Kjolstad, F., Amarasinghe, S.: Format abstraction for sparse tensor algebra compilers. Proc. ACM Program. Lang. **2**(OOPSLA), 123:1–123:30 (Oct 2018). https://doi.org/10.1145/3276493
6. Fiduccia, C.M.: Polynomial evaluation via the division algorithm the fast fourier transform revisited. In: Proceedings of the Fourth Annual ACM Symposium on Theory of Computing. p. 88–93. STOC '72, Association for Computing Machinery, New York, NY, USA (1972). https://doi.org/10.1145/800152.804900, https://doi.org/10.1145/800152.804900
7. Ghosh, S., Harsha, P., Herdade, S., Kumar, M., Saptharishi, R.: Fast numerical multivariate multipoint evaluation. In: 2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS). pp. 1426–1439. IEEE Computer Society, Los Alamitos, CA, USA (nov 2023). https://doi.org/10.1109/FOCS57990.2023.00088, https://doi.ieeecomputersociety.org/10.1109/FOCS57990.2023.00088
8. Goldsztejn, A.: Comparison of the Hansen-Sengupta and the Frommer-Lang-Schnurr existence tests. Computing **79**(1), 53–60 (2007). https://doi.org/10.1007/s00607-006-0217-8
9. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (Sep 2020). https://doi.org/10.1038/s41586-020-2649-2, https://doi.org/10.1038/s41586-020-2649-2
10. van der Hoeven, J.: Fast composition of numeric power series. Tech. Rep. 2008-09, Université Paris-Sud, Orsay, France (2008)

11. van der Hoeven, J.: Reliable homotopy continuation. Research report, LIX, Ecole polytechnique (Jan 2015), https://hal.science/hal-00589948
12. van der Hoeven, J., Lecerf, G.: Fast multivariate multi-point evaluation revisited. J. Complexity **56**, 38 (2020). https://doi.org/10.1016/j.jco.2019.04.001, id/No 101405
13. van der Hoeven, J., Lecerf, G.: Fast amortized multi-point evaluation. J. Complexity **67**, 15 (2021). https://doi.org/10.1016/j.jco.2021.101574, id/No 101574
14. van der Hoeven, J., Lecerf, G.: Amortized multi-point evaluation of multivariate polynomials. J. Complexity **74**, 17 (2023). https://doi.org/10.1016/j.jco.2022.101693, id/No 101693
15. Hormann, K., Kania, L., Yap, C.: Novel range functions via taylor expansions and recursive lagrange interpolation with application to real root isolation. In: Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation. p. 193–200. ISSAC '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3452143.3465532, https://doi.org/10.1145/3452143.3465532
16. Imbach, R., Moroz, G.: Fast evaluation and root finding for polynomials with floating-point coefficients. In: Proceedings of the 48th international symposium on symbolic and algebraic computation, ISSAC, Tromsø, Norway, July 24–27, 2023, pp. 325–334. New York, NY: Association for Computing Machinery (ACM) (2023). https://doi.org/10.1145/3597066.3597112
17. Jaulin, L., Kieffer, M., Didrit, O., Walter, E., Jaulin, L., Kieffer, M., Didrit, O., Walter, É.: Interval analysis. Springer (2001)
18. Kearfott, R.B.: Rigorous global search: continuous problems. Nonconvex optimization and its applications, Kluwer Academic Publishers, Dordrecht, Boston (1996), http://opac.inria.fr/record=b1092397
19. Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM J. Comput. **40**(6), 1767–1802 (2011). https://doi.org/10.1137/08073408X
20. Moroz, G.: New data structure for univariate polynomial approximation and applications to root isolation, numerical multipoint evaluation, and other problems. In: 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS). pp. 1090–1099. IEEE (2022)
21. Neumaier, A.: Interval methods for systems of equations. Cambridge University Press (1990). https://doi.org/10.1017/CBO9780511526473
22. Nüsken, M., Ziegler, M.: Fast multipoint evaluation of bivariate polynomials. In: Algorithms – ESA 2004. 12th annual European symposium, Bergen, Norway, September 14–17, 2004. Proceedings., pp. 544–555. Berlin: Springer (2004). https://doi.org/10.1007/b100428
23. Pan, V.Y.: Simple multivariate polynomial multiplication. J. Symb. Comput. **18**(3), 183–186 (1994). https://doi.org/10.1006/jsco.1994.1042
24. Plantinga, S., Vegter, G.: Isotopic approximation of implicit curves and surfaces. In: SGP '04: Eurographics/ACM SIGGRAPH Symposium on Geometry Processing. pp. 245–254 (2004). https://doi.org/http://doi.acm.org/10.1145/1057432.1057465
25. Ratschek, H., Rokne, J.: Computer methods for the range of functions. Ellis Horwood Series in Mathematics and Its Applications. Chichester: Ellis Horwood Limited; New York etc.: Halsted Press: a Division of John Wiley & Sons. 168 p. £ 16.95 (1984). (1984)
26. Smith, S., Karypis, G.: Tensor-matrix products with a compressed sparse tensor. In: Proceedings of the 5th Workshop on Irregular Applications: Architectures

and Algorithms. pp. 5:1–5:7. IA3 '15, ACM (2015). https://doi.org/10.1145/2833179.2833183
27. Snyder, J.M.: Interval analysis for computer graphics. In: Proceedings of the 19th annual conference on Computer graphics and interactive techniques. pp. 121–130. SIGGRAPH '92, ACM, New York, NY, USA (1992). https://doi.org/10.1145/133994.134024, http://doi.acm.org/10.1145/133994.134024
28. Umans, C.: Fast polynomial factorization and modular composition in small characteristic. In: Proceedings of the 40th annual ACM symposium on theory of computing, STOC 2008. Victoria, Canada, May 17–20, 2008, pp. 481–490. New York, NY: Association for Computing Machinery (ACM) (2008)