FAST MACAULAY NULL SPACE THROUGH THE INTERSECTION OF SHIFTED NULL SPACES*

RAPHAËL WIDDERSHOVEN†, NITHIN GOVINDARAJAN†, AND LIEVEN DE LATHAUWER†

Abstract. Solving systems of multivariate polynomial equations using Macaulay-based methods requires the computation of null spaces of large Macaulay matrices, a step that dominates the overall computational complexity. We present MINI, an iterative algorithm that constructs the null space at successive degrees using only the null space from the previous step. By efficiently shifting and intersecting null spaces, MINI avoids explicitly forming Macaulay matrices (apart from the small initial matrix). This approach leverages the fact that the null space dimension of successive matrices in the Macaulay construction grows much slower than the row space dimension, enabling a significant computational reduction in complexity. The complexity for quadratic square systems in N variables is reduced from $\mathcal{O}(2^{6N})$ to $\mathcal{O}(2^{4N})$, highlighting the efficiency of operating entirely in the null space. Experiments show that similar reductions are achieved for other systems, including overdetermined systems where Macaulay-based methods are particularly relevant.

Key words. Macaulay matrix, polynomials systems, polynomial root solving, null space computation, structured matrices, intersection of subspaces

MSC codes. 65H04, 65H10, 15A18

1. Introduction. Various disciplines (e.g., robotics [31], physics [17], statistics [30]) encounter problems that can be solved by finding the roots of a system of multivariate polynomial equations. Over the past decades, a wide range of algorithms has been developed, each suited to different problem characteristics. In this article, we focus on computing all roots of zero-dimensional systems, i.e., systems with a finite number of roots, in a numerical setting.

A first family of techniques is symbolic. One of the most influential approaches is the theory of Gröbner bases [9, 14], which enable exact elimination. Its practical efficiency has improved markedly through algorithms such as Faugère's F4/F5 [19, 20] and modern implementations like `msolve` [6]. Closely related are symbolic resultant-based formulations, such as Sylvester, Bézoutian, Cayley, Dixon, and Macaulay matrices [35, 7, 11, 15, 28]. These symbolic techniques are powerful for exact computation, but are sensitive to noise and less suited when approximate solutions are required.

Homotopy continuation offers another effective approach for computing exact solutions [27, 32]. This technique tracks solution paths from a start system with known roots to the target system. While highly effective for square systems, its classical form does not directly handle overdetermined systems. A common workaround is to reduce it to a square system—typically via randomization—and then certify the solutions [16, 12, 10], which can be computationally expensive and sensitive to noise.

To address numerical settings, hybrid approaches combine symbolic structure with numerical linear algebra. A prominent example is Stetter's numerical polynomial algebra [34]. These methods construct the Macaulay matrix to encode algebraic relations among the polynomials, but extract roots using rank-revealing numerical

*Resubmitted to the editors on August 22nd, 2026.

Funding: This work was funded by (1) the Flemish Government: this research received funding under the AI Research Program. LDL, NG and RW are affiliated to Leuven.AI - KU Leuven institute for AI, B-3000, Leuven, Belgium. (2) KU Leuven Internal Funds: iBOF/23/064, C14/22/096.

†E. E. Dept. (ESAT), KU Leuven, Leuven, Belgium (raphael.widdershoven@kuleuven.be, nithin.govindarajan@kuleuven.be, lieven.delathauwer@kuleuven.be)

techniques. By solving the system numerically rather than purely symbolically, they naturally accommodate approximate coefficients, much like a pseudoinverse stabilizes a nearly singular linear system.

Several strategies exist for root extraction from the Macaulay matrix. While alternatives exist that rely on the column space of the Macaulay matrix [5, 40], the most common approach remains to use its null space [34, 2]. These Macaulay null-space methods involve two main computational steps (see section 2) (1) computing a basis for the null space of a Macaulay matrix and (2) extracting a basis of Vandermonde vectors for that null space (through a tensor decomposition) [36, 37, 41, 14]. The computational bottleneck is often the first one [37], especially if the system is overdetermined and has only few solutions.

To mitigate the cost of this null-space step, several numerical schemes have been proposed. Fast recursive orthogonalization methods exploit the block structure of Macaulay matrices [3, 39]. Relinearization alternates between (re-)imposing shift structure and (re-)linearizing [25]. Displacement-rank techniques leverage the approximate Toeplitz-block-(block-)Toeplitz structure that arises from the shift pattern [22]. These approaches reduce computation by using the available structure, with different trade-offs in accuracy and implementation complexity.

1.1. Scope. Our focus in this work is on accelerating the computation of the null space of the Macaulay matrix, which remains a key bottleneck in numerical polynomial algebra. In the experiments in section 4, the proposed method is only compared against alternative methods for computing the Macaulay null space, as it does not aim to outperform methods for different paradigms. (However, Appendix D does provide a basic comparison with Gröbner-based computation and homotopy continuation, allowing the contribution to be put in a broader perspective.) Techniques such as homotopy continuation or Gröbner bases can be faster in practice if applicable, especially for square and/or sparse systems. However, these methods are less suited to overdetermined numerical settings, where approximate coefficients and noise pose challenges.

1.2. Contributions. We propose a new approach for computing the null space of the Macaulay matrix, which will be referred to as MINI[1]. MINI progressively computes the null space of nested subblocks of increasing size, leveraging the nested nature of Macaulay matrices of increasing degree and the fact that the null space of vertically stacked subblocks equals the intersection of their individual null spaces. To enable MINI to be both fast and accurate, we introduce a method to compute the intersection that exploits the available structure.

The proposed methodology is shown to significantly improve on other Macaulay null-space methods, especially for systems in many variables. For instance, for quadratic square systems in N variables, the asymptotic complexity is reduced from $\mathcal{O}(2^{6N})$ to $\mathcal{O}(2^{4N})$ (see Table 1). This improvement is also practically significant; e.g., for a system in $N = 8$ variables, MINI computes the Macaulay null space in 3.5 s compared to 600 s using the singular value decomposition and 110 s with PNLA [3] (see Figure 2). While for overdetermined systems a similar theoretical complexity has not yet been derived, similar practical improvements have been observed (see Figures 3 and 4), e.g., going from 1100 s to 3 s for systems of $S = 30$ equations in $N = 15$ variables.

[1]The name MINI was chosen for Macaulay Iterative Null space Intersection, but also to emphasize its simplicity and speed.

1.3. Assumptions. Throughout this paper, we study only systems with a finite number of solutions. This implies that the systems are either square (as many equations as unknowns) or overdetermined (more equations than unknowns). While the proposed method is applicable to underdetermined systems as well, our analysis and experiments are restricted to systems with a finite number of solutions. For ease of exposition, we make two simplifying assumptions. First, we assume that the system is not homogeneous, i.e., not all monomials present in a polynomial system have the same degree. This affects notation and introduces degree shifts in the Macaulay construction. Still, the method is general; adapting it to homogeneous systems follows standard adjustments found in Macaulay-based approaches [13, 41]. Second, we assume that all polynomials have the same degree D and have nonzero coefficients (randomly drawn in our experiments) for all monomials of degree at most D, although the methodology as such works for any system. Degree refers to total degree, i.e., the maximum sum of exponents of any monomial with a nonzero coefficient in $\mathbf{\Sigma}$.

1.4. Notation.

Linear algebra. Scalars are denoted by lowercase or uppercase letters (e.g., a, A), vectors by bold lowercase letters (e.g., $\mathbf{a}$), and matrices by bold uppercase letters (e.g., $\mathbf{A}$). $\cdot^{\mathrm{T}}$ and $\cdot^{-1}$ denote the transpose and inverse. $\mathrm{col}(\cdot)$, $\mathrm{null}(\cdot)$, $\mathrm{null}_{\mathrm{B}}(\cdot)$ denote the column space, the (right) null space or kernel and a basis for the null space, respectively, which satisfy $\mathbf{A}\,\mathrm{col}(\mathbf{B}) = \mathrm{col}(\mathbf{AB})$, $\mathbf{A}\,\mathrm{null}(\mathbf{A}) = \emptyset$ and $\mathbf{A}\,\mathrm{null}_{\mathrm{B}}(\mathbf{A}) = \mathbf{0}$.

Algebraic geometry. Variables are represented by the letter x with indices, e.g., $x_1, x_2, \ldots$ The vector of all N variables is denoted $\mathbf{x} = \begin{bmatrix} x_1 & \ldots & x_N \end{bmatrix}^{\mathrm{T}}$. A monomial is denoted $\mu = \prod_{i=1}^{N} x_i^{\alpha_i}$, where exponent $\alpha_i \in \mathbb{N}$ is associated with variable x_i. A polynomial is denoted $p = \sum c_j \mu_j$, while a system of S polynomial equations is denoted $\mathbf{\Sigma} = \{p_1, \ldots, p_S\}$. Its coefficient matrix and corresponding monomials can be obtained as $\mathrm{coef}(\mathbf{\Sigma})$ and $\mathrm{mon}(\mathbf{\Sigma})$, respectively. When counting the number of roots of a system, we include multiplicities, complex roots, and roots at infinity [13, 24, 41].

1.5. Outline. Section 2 first introduces the Macaulay matrix and how its null space is used for solving systems of polynomial equations. Section 3 then explains our proposed method, named MINI, for efficiently computing this null space. Finally, section 4 tests the accuracy and speed of the proposed method.

2. The Macaulay matrix for root finding. The objective of this paper is to efficiently compute a basis for the null space of the Macaulay matrix. This section defines this Macaulay matrix and explains how the well-known Macaulay-based solution method, see, e.g., [26, 2, 4], uses its null space for root solving. For ease of explanation, all roots are assumed to be affine and have multiplicity one. The proposed method (see subsection 3.4) does not require these assumptions.

Given is a system of polynomial equations $\mathbf{\Sigma}$ of S equations in N variables and of degree D of which we want to find all L roots. All monomials with nonzero coefficients are collected into a multivariate Vandermonde vector of degree D, denoted $\mathbf{v}_D$.

Example 2.1. Throughout this section we will use the following system [41]:

$$\begin{cases} -3 - x_1 - 2x_2 + 4x_1^2 + 6x_1x_2 + 7x_2^2 = 0 \\ -2 - x_1 + x_2 + 3x_1^2 - 7x_1x_2 + 5x_2^2 = 0 \\ 1 + 7x_1 + x_2 - 8x_1^2 + 3x_1x_2 + x_2^2 = 0 \end{cases}. \tag{2.1}$$

This system of $S = 3$ polynomials of degree $D = 2$ in $N = 2$ variables has $L = 1$ root: $\mathbf{x}^{(1)} = \begin{bmatrix} 1 & 0 \end{bmatrix}^{\mathrm{T}}$. Its monomials are collected in $\mathbf{v}_2(\mathbf{x}) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}^{\mathrm{T}}$.

2.1. The Macaulay matrix. Evaluating a system of polynomials $\boldsymbol{\Sigma}$ in one of its roots $\mathbf{x}^{(l)}$ gives a zero vector by definition, i.e., $\mathbf{C}\mathbf{v}_D^{(l)} = \mathbf{0}$ where $\mathbf{C}$ is the coefficient matrix of system $\boldsymbol{\Sigma}$ with corresponding monomials forming the entries of $\mathbf{v}_D(\mathbf{x})$ and $\mathbf{v}_D^{(l)} = \mathbf{v}_D(\mathbf{x}^{(l)})$. This means that the vectors $\{\mathbf{v}_D^{(l)}\}_{l=1}^L$ all lie in the null space of the coefficient matrix $\mathbf{C}$.

The other way around, it makes sense to try to obtain the roots from the null space of the coefficient matrix. We will explain how multivariate Vandermonde vectors can be extracted from an estimate of the null space in subsection 2.2. However, this requires that all vectors $\{\mathbf{v}^{(l)}\}_{l=1}^L$ are linearly independent and that they form a basis for the null space. The null space of the coefficient matrix does not always satisfy this though because (1) the multivariate Vandermonde vectors might not be linearly independent and (2) there may be other vectors, linearly independent of the Vandermonde vectors $\{\mathbf{v}^{(l)}\}_{l=1}^L$ that also lie in the null space. We will refer to the latter as spurious vectors.

For *Theorem* 2.1, the vector $\mathbf{v}_2^{(1)} \triangleq \mathbf{v}_2(\mathbf{x}^{(1)}) = \begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0\end{bmatrix}^{\mathrm{T}}$ with $\mathbf{x}^{(1)} = \begin{bmatrix}1 & 0\end{bmatrix}^{\mathrm{T}}$ and $\mathbf{v}_2 = \begin{bmatrix}1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2\end{bmatrix}^{\mathrm{T}}$ indeed satisfies $\mathbf{C}\mathbf{v}_2^{(1)} = \mathbf{0}$ with

$$\mathbf{C} = \left[\begin{array}{c|cc|ccc} -3 & -1 & -2 & 4 & 6 & 7 \\ -2 & -1 & 1 & 3 & -7 & 5 \\ 1 & 7 & 1 & -8 & 3 & 1 \end{array}\right], \tag{2.2}$$

but the coefficient matrix $\mathbf{C}$ has a null space of dimension three while the system only has one root, indicating the presence of spurious vectors.

The Macaulay construction solves these problems, making sure the null space of the coefficient matrix is spanned by independent Vandermonde vectors in the different roots. As all roots of a polynomial p are also roots of μp for any monomial μ (e.g., all roots of $p = x_1^2 + x_2^2 - 1$ are also roots of $x_1 p = x_1^3 + x_1x_2^2 - x_1$), $\boldsymbol{\Sigma} = \{p_1, \ldots, p_S\}$ has the same roots as the system $\boldsymbol{\Sigma}_E$, which contains all polynomials of $\boldsymbol{\Sigma}$ multiplied with all monomials of degree at most $\Delta = E - D$:

$$\boldsymbol{\Sigma}_E = \{\mu p_1, \ldots, \mu p_S\}_{\mu \in \mathbf{v}_\Delta(\mathbf{x})}. \tag{2.3}$$

The Macaulay matrix of degree E is the coefficient matrix of $\boldsymbol{\Sigma}_E$, i.e.,

$$\mathbf{M}_E = \mathrm{coef}(\boldsymbol{\Sigma}_E). \tag{2.4}$$

It can be shown that if $E = D + \Delta$ is above a certain degree E^* (known as the degree of regularity), no more spurious vectors are present and the Vandermonde vectors in the L roots $\{\mathbf{v}_E^{(l)}\}_{l=1}^L$ form a basis for the L-dimensional null space of the Macaulay matrix $\mathbf{M}_E$:

$$\mathrm{null}(\mathbf{M}_E) = \mathrm{col}\Big(\begin{bmatrix}\mathbf{v}_E^{(1)} & \ldots & \mathbf{v}_E^{(L)}\end{bmatrix}\Big). \tag{2.5}$$

For *Theorem* 2.1, degree $\Delta = 1$ is large enough, meaning we multiply all $S = 3$

polynomials with $\mathbf{v}_1 = \begin{bmatrix}1 & x_1 & x_2\end{bmatrix}^{\mathrm{T}}$ yielding the set of polynomials

$$\mathbf{\Sigma}_3 = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \hline x_1p_1 \\ x_1p_2 \\ x_1p_3 \\ \hline x_2p_1 \\ x_2p_2 \\ x_2p_3 \end{bmatrix} = \left[\begin{array}{ccc|ccc|cccc} -3 & -1 & -2 & 4 & 6 & 7 & 0 & 0 & 0 & 0 \\ -2 & -1 & 1 & 3 & -7 & 5 & 0 & 0 & 0 & 0 \\ 1 & 7 & 1 & -8 & 3 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & -3 & 0 & -1 & -2 & 0 & 4 & 6 & 7 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 & 3 & -7 & 5 & 0 \\ 0 & 1 & 0 & 7 & 1 & 0 & -8 & 3 & 1 & 0 \\ \hline 0 & 0 & -3 & 0 & -1 & -2 & 0 & 4 & 6 & 7 \\ 0 & 0 & -2 & 0 & -1 & 1 & 0 & 3 & -7 & 5 \\ 0 & 0 & 1 & 0 & 7 & 1 & 0 & -8 & 3 & 1 \end{array}\right] \left[\begin{array}{c} 1 \\ \hline x_1 \\ x_2 \\ \hline x_1^2 \\ x_1x_2 \\ x_2^2 \\ \hline x_1^3 \\ x_1^2x_2 \\ x_1x_2^2 \\ x_2^3 \end{array}\right]. \tag{2.6}$$

Its coefficient matrix (the Macaulay matrix $\mathbf{M}_3$) has a 1-dimensional null space. That null space is spanned by $\mathbf{v}_3^{(1)} = \begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$, a multivariate Vandermonde vector in the only root ($\mathbf{x}^{(1)} = \begin{bmatrix}1 & 0\end{bmatrix}^{\mathrm{T}}$).

2.2. Root recovery from the Macaulay null space. One can numerically compute a basis $\mathbf{K}_E$ for the null space of the Macaulay matrix $\mathbf{M}_E$ above the degree of regularity E^*, but this basis will be a linear combination of the basis vectors $\mathbf{V}_E = \begin{bmatrix}\mathbf{v}_E^{(1)} & \dots & \mathbf{v}_E^{(L)}\end{bmatrix}$ we are looking for, i.e., for some full rank matrix $\mathbf{G} \in \mathbb{C}^{L\times L}$

$$\mathrm{null}_{\mathrm{B}}(\mathbf{M}_E) = \mathbf{K}_E = \mathbf{V}_E\mathbf{G}. \tag{2.7}$$

To recover the desired $\mathbf{V}_E$ from the numerically obtained $\mathbf{K}_E$, the additional structure in $\mathbf{V}_E$ can be exploited, namely that $\mathbf{V}_E$ is a multivariate Vandermonde matrix. For any multivariate Vandermonde vector $\mathbf{v}_E$, it holds that

$$\mathbf{S}_n\mathbf{v}_E = x_n\mathbf{v}_{E-1}, \text{ for } n = 1, \dots, N \tag{2.8}$$

where $\mathbf{S}_n$ is a selection matrix that selects all rows of $\mathbf{v}_E$ where variable x_n has a power of at least one. Similarly, for a multivariate Vandermonde matrix $\mathbf{V}_E$, it holds that

$$\mathbf{S}_n\mathbf{V}_E = \mathbf{V}_{E-1}\mathbf{D}_n, \text{ for } n = 1, \dots, N \tag{2.9}$$

with $\mathbf{D}_n = \mathrm{diag}\Big(\begin{bmatrix}x_n^{(1)} & \dots & x_n^{(L)}\end{bmatrix}\Big)$.

For *Theorem* 2.1, (2.8) with $n = 1$ gives

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \\ x_1^3 \\ x_1^2x_2 \\ x_1x_2^2 \\ x_2^3 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}. \tag{2.10}$$

Combining (2.7) and (2.9) gives

$$\mathbf{S}_n\mathbf{K}_E\mathbf{G}^{-1} = \mathbf{V}_{E-1}\mathbf{D}_n, \qquad \text{for } n = 1, \dots, N \tag{2.11}$$

$$\iff \qquad \mathbf{S}_n\mathbf{K}_E = \mathbf{V}_{E-1}\mathbf{D}_n\mathbf{G}, \qquad \text{for } n = 1, \dots, N \tag{2.12}$$

where the left side $\mathbf{S}_n\mathbf{K}_E$ is known. This is a simultaneous diagonalization problem, which can be solved by computing a canonical polyadic decomposition (CPD) of rank L [36, 41]. A CPD writes a tensor as a (minimal) sum of rank-1 tensors, which are tensors that can be written as the outer product of nonzero vectors. Here we have

$$\mathbf{S}_n\mathbf{K}_E = \sum_{l=1}^{L} x_n^{(l)} \mathbf{v}_{E-1}^{(l)} \otimes \mathbf{g}^{(l)}, \qquad \text{for } n = 1, \ldots, N \tag{2.13}$$

$$\iff \quad \mathrm{cat}_3(\mathbf{S}_0\mathbf{K}_E, \ldots, \mathbf{S}_N\mathbf{K}_E) = \sum_{l=1}^{L} \mathbf{v}_{E-1}^{(l)} \otimes \mathbf{g}^{(l)} \otimes \mathbf{x}^{(l)}, \tag{2.14}$$

where $\mathbf{G} = \begin{bmatrix} \mathbf{g}^{(1)} & \ldots & \mathbf{g}^{(L)} \end{bmatrix}^{\mathrm{T}}$, $\otimes$ denotes the outer product and $\mathrm{cat}_3(\cdot)$ stacks a sequence of matrices along a new third mode to form a third-order tensor. The tensor $\mathrm{cat}_3(\mathbf{S}_1\mathbf{K}_E, \ldots, \mathbf{S}_N\mathbf{K}_E)$ is expected to have a unique[2] rank-L CPD[3], which is given in (2.14) [36, 41]. Thus, by computing the CPD, one can obtain the roots $\{\mathbf{x}^{(l)}\}_{l=1}^{L}$ of the system of polynomial equations.

The CPD can be computed using a generalized eigenvalue decomposition (GEVD) of two slices of this tensor (see e.g. [2]). However, incorporating all available slices can improve the accuracy, especially in the presence of noise, albeit at a slightly higher computational cost [41]. This can be achieved by combining results from multiple GEVDs (the so-called generalized eigenspace decomposition (GESD)) [18, 5] and/or by refining the solution using optimization-based methods [33].

3. The MINI algorithm. Throughout this section, we will use the following system of equations to explain the proposed method:

$$\begin{cases} p_1 = 1 + 2x_1 + 3x_2 + 1x_1^2 + 5x_1x_2 + 6x_2^2 = 0 \\ p_2 = 2 + 1x_1 + 7x_2 + 8x_1^2 + 1x_1x_2 + 9x_2^2 = 0 \end{cases}. \tag{3.1}$$

[2]Unless $L > \binom{N+E}{N} - N - 1$ (which appears to occur only when $D = N = 2$, i.e., for quadratic bivariate systems), the CPD is generically unique up to scaling and permutation indeterminacies [36]. While the permutation indeterminacy only affects the order of the roots, the scaling indeterminacy can arbitrarily scale the solution vectors. To resolve this, one can impose an additional constraint on the obtained null space, namely $\mathbf{S}_0\mathbf{V}_E = \mathbf{V}_{E-1}\mathbf{D}_0$, where $\mathbf{D}_0$ is an identity matrix and $\mathbf{S}_0$ selects all monomials of degree at most $D-1$. Combining it with (2.7) gives $\mathbf{S}_0\mathbf{K}_E = \mathbf{V}_{E-1}\mathbf{D}_0\mathbf{G}$, following the same reasoning used to derive (2.12). This effectively adds a "1" entry to the solution vector $\mathbf{x}^{(l)}$ in (2.14), thereby fixing the scaling indeterminacy.

[3]When roots with multiplicity higher than one are present, a more general decomposition is needed, namely a block term decomposition [37].

Its Macaulay matrix of degree four, $\mathbf{M}_4$, is the coefficient matrix of

(3.2)

$$\begin{bmatrix} p_1 \\ p_2 \\ \hline x_1\,p_1 \\ x_1\,p_2 \\ x_2\,p_1 \\ x_2\,p_2 \\ \hline x_1^2\,p_1 \\ x_1^2\,p_2 \\ x_1x_2\,p_1 \\ x_1x_2\,p_2 \\ x_2^2\,p_1 \\ x_2^2\,p_2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 8 & 1 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ \hline x_1 \\ x_2 \\ \hline x_1^2 \\ x_1x_2 \\ x_2^2 \\ \hline x_1^3 \\ x_1^2x_2 \\ x_1x_2^2 \\ x_2^3 \\ \hline x_1^4 \\ x_1^3x_2 \\ x_1^2x_2^2 \\ x_1x_2^3 \\ x_2^4 \end{bmatrix}.$$

Note that the Macaulay matrices of degree 2 ($\mathbf{M}_2$) and 3 ($\mathbf{M}_3$) are also indicated. The objective is to efficiently compute the null space of Macaulay matrices, such as $\mathbf{M}_4$.

Using this example, the idea behind the proposed algorithm, MINI, is first explained in subsection 3.1. A formal definition is given in subsection 3.4, but to understand the algorithm we first need to know how to shift in the null space and compute null space intersections. These operations are first introduced in subsection 3.2 for our example and then formalized in subsection 3.3. After the formal explanation of the method in subsection 3.4, subsection 3.5 shows a way of reducing the number of intersections. Subsection 3.6 discusses some of the considerations that impact the accuracy. Finally, subsection 3.7 analyzes the computational complexity of MINI.

3.1. The idea behind MINI. We start by computing the null space of the smallest Macaulay matrix possible, which is relatively cheap thanks to the small dimensions:

$$\text{null}(\mathbf{M}_2) = \text{null}\left(\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 5 & 6 \\ 2 & 1 & 7 & 8 & 1 & 9 \end{array}\right]\right). \tag{3.3}$$

This null space can be reused to compute null($\mathbf{M}_3$), which is the intersection of three null spaces, corresponding to matrices formed by inserting zero columns in $\mathbf{M}_2$ at different positions:

$$\text{null}(\mathbf{M}_3) = \text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 1 & 2 & 3 & 1 & 5 & 6 & 0 & 0 & 0 \\ 2 & 1 & 7 & 8 & 1 & 9 & 0 & 0 & 0 \end{array}\right]\right) \tag{3.4}$$

$$\cap\,\text{null}\left(\left[\begin{array}{c|ccc|ccc|c} 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 \\ 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 \end{array}\right]\right) \tag{3.5}$$

$$\cap\,\text{null}\left(\left[\begin{array}{cc|ccc|ccc} 0 & 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 \\ 0 & 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 \end{array}\right]\right). \tag{3.6}$$

These three matrices are actually the coefficient matrices of $\begin{bmatrix} p_1 & p_2 \end{bmatrix}$, $\begin{bmatrix} x_1p_1 & x_1p_2 \end{bmatrix}$ and $\begin{bmatrix} x_2p_1 & x_2p_2 \end{bmatrix}$. We thus shift the polynomial system with 1, x_1 and x_2, respectively, and intersect the null spaces of the coefficient matrices of the resulting systems.

Similarly, to compute null($\mathbf{M}_4$), the system $\begin{bmatrix} p_1 & p_2 & x_1p_1 & x_1p_2 & x_2p_1 & x_2p_2 \end{bmatrix}$ (of which $\mathbf{M}_3$ is the coefficient matrix) is shifted with 1, x_1 and x_2. By intersecting the null spaces of the coefficient matrices of the resulting systems, we obtain null($\mathbf{M}_4$):

$$\text{null}(\mathbf{M}_4) = \text{null}\left(\left[\begin{array}{cc|ccc|ccc|cccc} 1 & 2 & 3 & 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 8 & 1 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 & 0 & 0 & 0 \end{array}\right]\right) \tag{3.7}$$

$$\cap\,\text{null}\left(\left[\begin{array}{cc|ccc|ccc|cccc} 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 & 0 \end{array}\right]\right) \tag{3.8}$$

$$\cap\,\text{null}\left(\left[\begin{array}{cc|ccc|ccc|cccc} 0 & 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 5 & 6 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 7 & 0 & 0 & 8 & 1 & 9 \end{array}\right]\right). \tag{3.9}$$

This can be further repeated, as the union of all monomials of degree at most D multiplied by 1, x_1 and x_2 yields all monomials of degree at most $D+1$.

The idea of this paper is to do all these operations directly on the null space, never going back to the polynomials or their coefficients. For instance, given a basis for the null space of a matrix, it can be extended by adding zeros and an identity matrix to form a basis for the null space of the same matrix with additional zero columns:

$$\text{null}_\text{B}(\mathbf{M}_2) = \text{null}_\text{B}\left(\left[\begin{array}{cc|ccc|ccc} 1 & 2 & 3 & 1 & 5 & 6 \\ 2 & 1 & 7 & 8 & 1 & 9 \end{array}\right]\right) = \left[\begin{array}{cccc} \text{-2} & 1 & \text{-4} & \text{-2} \\ \hline 0 & \text{-3} & \text{-1} & \text{-1} \\ 3 & 0 & 0 & \text{-3} \\ \hline \text{-2} & 1 & 1 & 2 \\ \text{-1} & 2 & 1 & 1 \\ 0 & \text{-1} & 0 & 1 \end{array}\right] \tag{3.10}$$

(3.11)

$$\Rightarrow \text{null}_\text{B}\left(\left[\begin{array}{cc|ccc|ccc|cccc} 1 & 2 & 3 & 1 & 5 & 6 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 8 & 1 & 9 & 0 & 0 & 0 & 0 \end{array}\right]\right) = \left[\begin{array}{cccc|cccc} \text{-2} & 1 & \text{-4} & \text{-2} & 0 & 0 & 0 & 0 \\ \hline 0 & \text{-3} & \text{-1} & \text{-1} & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & \text{-3} & 0 & 0 & 0 & 0 \\ \hline \text{-2} & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ \text{-1} & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \text{-1} & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right].$$

Any permutation of the columns of a coefficient matrix, as in (3.4)–(3.9), corresponds to a permutation of the entries of the vectors in the associated null space.

3.2. Intersection of null spaces. The intersections of null spaces encountered in (3.4)–(3.6) could be computed with classical techniques (see, e.g., [21, Sections 6.4.2 and 6.4.4]). Instead, by taking advantage of the zero columns, the number of computations can be reduced. For instance, in the example above, we need to compute the following intersection:

$$\text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 1 & 2 & 3 & 1 & 5 & 6 & 0 & 0 & 0 \\ 2 & 1 & 7 & 8 & 1 & 9 & 0 & 0 & 0 \end{array}\right]\right) \tag{3.12}$$

$$\cap\,\text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 \\ 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 \end{array}\right]\right) = \tag{3.13}$$

$$\text{col}\left(\left[\begin{array}{cccc|ccc} -2 & 1 & -4 & -2 & 0 & 0 & 0 \\ \hline 0 & -3 & -1 & -1 & 0 & 0 & 0 \\ 3 & 0 & 0 & -3 & 0 & 0 & 0 \\ \hline -2 & 1 & 1 & 2 & 0 & 0 & 0 \\ -1 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]\right) \cap \text{col}\left(\left[\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -2 & 1 & -4 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & -3 & -1 & -1 & 0 & 0 & 0 \\ 3 & 0 & 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline -2 & 1 & 1 & 2 & 0 & 0 & 0 \\ -1 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]\right). \tag{3.14}$$

We now permute the columns of the (extended) coefficient matrices in (3.12)–(3.13) to partition them into (1) a group of columns where both matrices are nonzero, (2) a group of columns that are only nonzero in the first matrix and (3) a group of columns that are only nonzero in the second matrix. Correspondingly, the bases for the null spaces are partitioned into blocks, which will be referred to as the common blocks (denoted $\mathbf{C}_1$ and $\mathbf{C}_2$) and distinct blocks (denoted $\mathbf{D}_1$ and $\mathbf{D}_2$):

$$\text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 2 & 1 & 5 & 1 & 3 & 6 & 0 & 0 & 0 \\ 1 & 8 & 1 & 2 & 7 & 9 & 0 & 0 & 0 \end{array}\right]\right) \tag{3.15}$$

$$\cap\,\text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 1 & 2 & 3 & 0 & 0 & 0 & 1 & 5 & 6 \\ 2 & 1 & 7 & 0 & 0 & 0 & 8 & 1 & 9 \end{array}\right]\right) = \tag{3.16}$$

$$\text{col}\left(\left[\begin{array}{cccc|ccc} 0 & -3 & -1 & -1 & 0 & 0 & 0 \\ -2 & 1 & 1 & 2 & 0 & 0 & 0 \\ -1 & 2 & 1 & 1 & 0 & 0 & 0 \\ \hline -2 & 1 & -4 & -2 & 0 & 0 & 0 \\ 3 & 0 & 0 & -3 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]\right) \cap \text{col}\left(\left[\begin{array}{ccc|ccc} -2 & 1 & -4 & -2 & 0 & 0 & 0 \\ 0 & -3 & -1 & -1 & 0 & 0 & 0 \\ 3 & 0 & 0 & -3 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline -2 & 1 & 1 & 2 & 0 & 0 & 0 \\ -1 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]\right) \tag{3.17}$$

$$\triangleq \text{col}\left(\begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}\right) \cap \text{col}\left(\begin{bmatrix} \mathbf{C}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{D}_2 & \mathbf{0} \end{bmatrix}\right). \tag{3.18}$$

The intersection is formed by all vectors $\mathbf{z}$ such that, for some $\mathbf{u}, \mathbf{v}, \mathbf{x}$ and $\mathbf{y}$,

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{D}_2 & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}, \tag{3.19}$$

$$\Rightarrow \begin{cases} \mathbf{z}_1 = \mathbf{C}_1\mathbf{x} = \mathbf{C}_2\mathbf{u} \iff \begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} = \mathbf{0}, \\ \mathbf{z}_2 = \mathbf{D}_1\mathbf{x} = \mathbf{v}, \\ \mathbf{z}_3 = \mathbf{y} = \mathbf{D}_2\mathbf{u}. \end{cases} \tag{3.20}$$

A basis for the intersection can thus be computed as

$$\mathbf{Z} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \text{null}_\text{B}\left(\begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix}\right) \tag{3.21}$$

$$\left(= \begin{bmatrix} \mathbf{0} & \mathbf{C}_2 \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \text{null}_\text{B}\left(\begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix}\right) \right). \tag{3.22}$$

Going back to our example, using

$$\text{null}_\text{B}\left(\begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix}\right) = \text{null}_\text{B}\left(\left[\begin{array}{cccc|cccc} 0 & -3 & -1 & -1 & 2 & -1 & 4 & 2 \\ -2 & 1 & 1 & 2 & 0 & 3 & 1 & 1 \\ -1 & 2 & 1 & 1 & -3 & 0 & 0 & 3 \end{array}\right]\right) \tag{3.23}$$

$$= \left[\begin{array}{ccccc} 0 & -1 & -1 & 5 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 3 & 0 & 3 & 0 & 0 \\ 0 & 0 & -4 & -3 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 3 & 0 \\ 0 & 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right], \tag{3.24}$$

we can write (already undoing the row permutations):

$$\text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 1 & 2 & 3 & 1 & 5 & 6 & 0 & 0 & 0 \\ 2 & 1 & 7 & 8 & 1 & 9 & 0 & 0 & 0 \end{array}\right]\right) \tag{3.25}$$

$$\cap \text{null}\left(\left[\begin{array}{ccc|ccc|ccc} 0 & 1 & 0 & 2 & 3 & 0 & 1 & 5 & 6 \\ 0 & 2 & 0 & 1 & 7 & 0 & 8 & 1 & 9 \end{array}\right]\right) \tag{3.26}$$

$$= \text{col}\left(\left[\begin{array}{cccc|cccc} -2 & 1 & -4 & -2 & 0 & 0 & 0 & 0 \\ \hline 0 & -3 & -1 & -1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & -3 & 0 & 0 & 0 & 0 \\ \hline -2 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ -1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & -2 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & -1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{array}\right] \left[\begin{array}{ccccc} 0 & -1 & -1 & 5 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 3 & 0 & 3 & 0 & 0 \\ 0 & 0 & -4 & -3 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 3 & 0 \\ 0 & 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right]\right). \tag{3.27}$$

Remark 3.1. Although mathematically intersections are symmetric and

$$\begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \operatorname{null}\big(\begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix}\big) = \begin{bmatrix} \mathbf{0} & \mathbf{C}_2 \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \operatorname{null}\big(\begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix}\big), \tag{3.28}$$

numerically these two implementations may differ. In our experiments further up, we used the rule of thumb that we work with the left respectively right version of (3.28) when $\mathbf{C}_1$ respectively $\mathbf{C}_2$ is expected to have the highest accuracy. In practice, this means that we used $\mathbf{C}_1$ or $\mathbf{C}_2$, depending on which was obtained by the fewest prior intersections.

3.3. Notation for operations on null spaces. The previous subsections introduced our method through an example. To now formally write down the algorithm, we introduce notation for the two required operations: shifting null spaces and intersecting null spaces. These should operate on null space bases $\mathbf{K}_d = \operatorname{null}_\mathrm{B}(\mathbf{M}_d)$ only, as any underlying system $\mathbf{\Sigma}_d$ or its coefficient matrix $\mathbf{M}_d = \operatorname{coef}(\mathbf{\Sigma}_d)$ will not be stored (after the initial step). To keep track of which row of a null space basis corresponds to which monomial, a basis $\mathbf{K}_d$ will be written down together with its corresponding monomials as $\{\mathbf{K}_d, \mathbb{M}_d\} = \{\operatorname{null}_\mathrm{B}(\operatorname{coef}(\mathbf{\Sigma}_d)), \operatorname{mon}(\mathbf{\Sigma}_d)\}$. Here, we define the notation $\mathbb{M} = \{\mu_1, \ldots, \mu_I\}$ for an ordered set of monomials with operations:

$$\mathbb{M}_1 \cap \mathbb{M}_2 = \{\mu : \mu \in \mathbb{M}_1 \text{ and } \mu \in \mathbb{M}_2\}, \text{ intersection (ordered as in } \mathbb{M}_1), \tag{3.29}$$

$$\mathbb{M}_1 \setminus \mathbb{M}_2 = \{\mu : \mu \in \mathbb{M}_1 \text{ and } \mu \notin \mathbb{M}_2\}, \text{ set difference (ordered as in } \mathbb{M}_1), \tag{3.30}$$

$$\mathbb{M}_1 \parallel \mathbb{M}_2, \text{ concatenation of distinct sets,} \tag{3.31}$$

$$\mu\mathbb{M} = \{\mu\mu_i : \mu_i \in \mathbb{M}\}, \text{ element-wise multiplication by a monomial } \mu. \tag{3.32}$$

3.3.1. Shifting null spaces. The first required operation is shifting a system $\mathbf{\Sigma}$ with a monomial μ, i.e., multiplying every polynomial in $\mathbf{\Sigma}$ with μ, which we denote by $\mu\mathbf{\Sigma}$. This does not change the coefficient matrix or its null space, i.e., $\operatorname{coef}_{\mu\mathbb{M}}(\mu\mathbf{\Sigma}) = \operatorname{coef}_{\mathbb{M}}(\mathbf{\Sigma})$ and $\mathbf{K} = \operatorname{null}_\mathrm{B}(\operatorname{coef}_{\mu\mathbb{M}}(\mu\mathbf{\Sigma})) = \operatorname{null}_\mathrm{B}(\operatorname{coef}_{\mathbb{M}}(\mathbf{\Sigma}))$. Instead, it changes the monomials with which the coefficients are associated, from $\mathbb{M}$ to $\mu\mathbb{M}$. In our compact notation, shifting $\{\mathbf{K}, \mathbb{M}\}$ with μ will be written as $\{\mathbf{K}, \mu\mathbb{M}\}$.

3.3.2. Intersecting null spaces. The other required operation is computing the intersection of two null spaces, $\{\mathbf{K}_1, \mathbb{M}_1\}$ and $\{\mathbf{K}_2, \mathbb{M}_2\}$. To compute their intersection, these null spaces must first be expanded so that they correspond to the coefficient matrices of polynomials expressed in a common monomial basis $\mathbb{M}$, as described in (3.10)–(3.11). Next, the bases of these null spaces need to be partitioned to determine the common ($\mathbf{C}_1$ and $\mathbf{C}_2$) and distinct ($\mathbf{D}_1$ and $\mathbf{D}_2$) blocks, as was done in (3.17)–(3.18). From these blocks, a basis for the intersection can efficiently be computed using (3.21). Alternatively, $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{D}_1$ and $\mathbf{D}_2$ can directly be obtained from $\mathbf{K}_1$ and $\mathbf{K}_2$—without needing to expand first—since their partitioning follows from the decomposition of the monomials $\mathbb{M}$ in $\mathbb{M}_1 \cap \mathbb{M}_2$, $\mathbb{M}_1 \setminus \mathbb{M}_2$ and $\mathbb{M}_2 \setminus \mathbb{M}_1$.

To select the correct corresponding rows from $\mathbf{K}_1$ and $\mathbf{K}_2$, we define the following selection matrix for given sets of monomials $\mathbb{M}_a$ and $\mathbb{M}_b$:

$$(\mathbf{S}_{\mathbb{M}_a \to \mathbb{M}_b})_{ij} = \begin{cases} 1 & \text{if } (\mathbb{M}_a)_j = (\mathbb{M}_b)_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.33}$$

The intersection can then be computed with Algorithm 3.1 and will be denoted as

$$\{\mathbf{K}, \mathbb{M}\} = \{\mathbf{K}_1, \mathbb{M}_1\} \cap \{\mathbf{K}_2, \mathbb{M}_2\}. \tag{3.34}$$

Algorithm 3.1 Intersection of Macaulay null spaces

Input: $\{\mathbf{K}_1, \mathbb{M}_1\}$ and $\{\mathbf{K}_2, \mathbb{M}_2\}$
Output: $\{\mathbf{K}, \mathbb{M}\}$
1: $\mathbf{C}_1 = \mathbf{S}_{\mathbb{M}_1 \to \mathbb{M}_1 \cap \mathbb{M}_2} \mathbf{K}_1$
2: $\mathbf{C}_2 = \mathbf{S}_{\mathbb{M}_2 \to \mathbb{M}_1 \cap \mathbb{M}_2} \mathbf{K}_2$
3: $\mathbf{D}_1 = \mathbf{S}_{\mathbb{M}_1 \to \mathbb{M}_1 \setminus \mathbb{M}_2} \mathbf{K}_1$
4: $\mathbf{D}_2 = \mathbf{S}_{\mathbb{M}_2 \to \mathbb{M}_2 \setminus \mathbb{M}_1} \mathbf{K}_2$
5: $\mathbf{K} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \text{null}_\text{B}\left(\begin{bmatrix} \mathbf{C}_1 & -\mathbf{C}_2 \end{bmatrix}\right)$
6: $\mathbb{M} = (\mathbb{M}_1 \cap \mathbb{M}_2) \parallel (\mathbb{M}_1 \setminus \mathbb{M}_2) \parallel (\mathbb{M}_2 \setminus \mathbb{M}_1)$

3.4. MINI. Having defined the required operations, we now formally present the algorithm in Algorithm 3.2.

Algorithm 3.2 MINI

Input: Set of polynomials $\mathbf{\Sigma}$ of degree D in N unknowns
Output: Macaulay null space at degree E, i.e., $\mathbf{K}_E$, in monomial basis $\mathbb{M}_E$
1: $\{\mathbf{K}_D, \mathbb{M}_D\} \leftarrow \{\text{null}_\text{B}(\text{coef}(\mathbf{\Sigma})), \text{mon}(\mathbf{\Sigma})\}$
2: **for** $d = D+1, \ldots, E$ **do**
3: $\quad \{\mathbf{K}_d^{(0)}, \mathbb{M}_d^{(0)}\} \leftarrow \{\mathbf{K}_{d-1}, \mathbb{M}_{d-1}\}$
4: $\quad$ **for** $n = 1, \ldots, N$ **do**
5: $\quad\quad \{\mathbf{K}_d^{(n)}, \mathbb{M}_d^{(n)}\} \leftarrow \{\mathbf{K}_d^{(n-1)}, \mathbb{M}_d^{(n-1)}\} \cap \{\mathbf{K}_{d-1}, x_n \mathbb{M}_{d-1}\}$ (Algorithm 3.1)
6: $\quad$ **end for**
7: $\quad \{\mathbf{K}_d, \mathbb{M}_d\} \leftarrow \{\mathbf{K}_d^{(n)}, \mathbb{M}_d^{(n)}\}$ (Macaulay null space of degree d)
8: **end for**

The first step of Algorithm 3.2 is to form a coefficient matrix of the input system $\mathbf{\Sigma}$ (in a fitting monomial basis $\mathbb{M}_D$) and to compute a basis for its null space $\mathbf{K}_D$. This is the only step in which an operation is done on the system $\mathbf{\Sigma}$ or its coefficient matrix; the remaining steps only use null spaces. The computation time for the first step of the algorithm is usually negligible.

All other lines repeatedly shift null spaces with monomials $x_1, \ldots, x_N$ (and monomial 1 in line 3) and intersect them. This relies on three ideas. (1) The null space of the Macaulay matrix can be computed as the intersection of the null space of its row blocks, because $\text{null}(\begin{bmatrix} \mathbf{A}_1^\text{T} & \mathbf{A}_2^\text{T} \end{bmatrix}^\text{T}) = \text{null}(\mathbf{A}_1) \cap \text{null}(\mathbf{A}_2)$ for any $\mathbf{A}_1$ and $\mathbf{A}_2$. (2) Shifting polynomials results in a trivial extension of the null space. (3) Each row of the degree-d Macaulay matrix $\mathbf{M}_d$ corresponds to a row of the degree-$(d-1)$ Macaulay matrix $\mathbf{M}_{d-1}$ shifted with 1, x_1, …, x_{N-1} or x_N. A null space basis of $\mathbf{M}_d$ can thus be formed by intersecting $N+1$ shifted copies of $\text{null}_\text{B}(\mathbf{M}_{d-1})$.

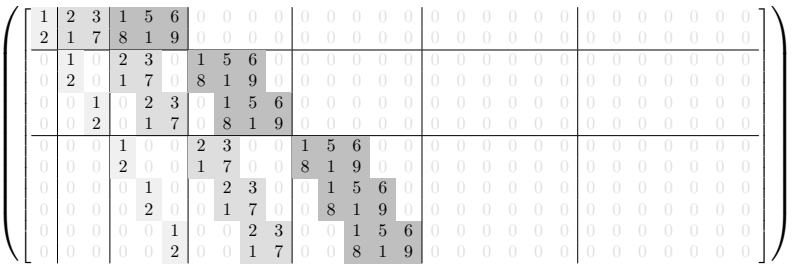
3.5. MINISkip. The MINI algorithm (described in subsection 3.4) computes the null space for all values of the degree $d = D+1, \ldots, E$. If one only requires the null space of the Macaulay matrix at a specific degree (as is the case when solving for roots following the method in section 2), one can often reduce the total number of intersections required. An open question is how to do this optimally. As a heuristic, we propose here to shift the current null space with monomials $1, x_1^k, x_2^k, \ldots$ instead of $1, x_1, x_2, \ldots$, where k is the largest possible value that ensures that the obtained

null space coincides with the null space of the Macaulay matrix.

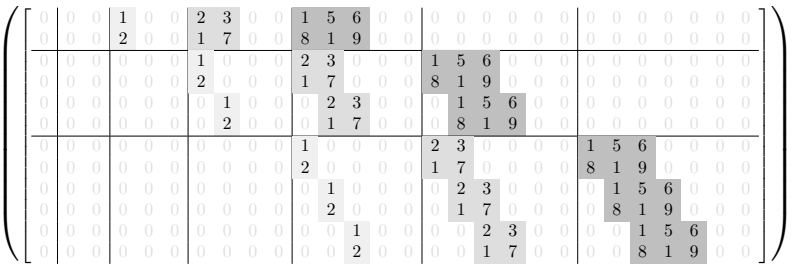
For our example (see (3.1)–(3.2) and subsection 3.1), null($\mathbf{M}_6$) can directly be formed by extending $\mathbf{M}_4$ with zero columns corresponding with shifts 1, x_1^2 and x_2^2, respectively, and intersecting the null spaces of the three resulting matrices:

(3.35) null($\mathbf{M}_6$) =

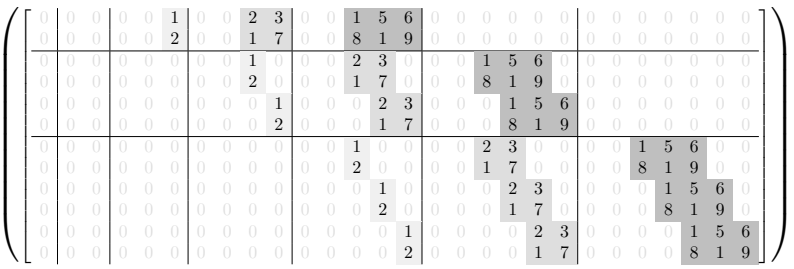
(3.36) null



(3.37) $\cap$ null



(3.38) $\cap$ null



.

As explained in subsection 3.3.1, the null space of these extended matrices can directly be formed from null($\mathbf{M}_4$). One can thus skip the computation of null($\mathbf{M}_5$) and directly compute null($\mathbf{M}_6$) from null($\mathbf{M}_4$).

To ensure that the obtained null space corresponds with the null space of $\mathbf{M}_6$, each row of $\mathbf{M}_6$ should be a row of at least one of the three matrices in (3.36)–(3.38) and no row should be present in (3.36)–(3.38) that is not present in $\mathbf{M}_6$. This condition is satisfied because $\mathbf{M}_4$ and $\mathbf{M}_6$ are the coefficient matrices of the polynomials in (3.1) multiplied with all monomials of degree at most two and four, respectively, and because all monomials of degree at most four can be constructed by multiplying a monomial of degree at most two with $1, x_1^2$ or x_2^2, as illustrated in Figure 1 (left). The same would not hold for forming null($\mathbf{M}_7$) from null($\mathbf{M}_4$) as not all monomials of degree at most five can be formed as the product of a monomial of degree at most two with 1, x_1^3 or x_2^3, as illustrated in Figure 1 (right).

This property is generalized in Theorem 3.2. Thus, the null space of the Macaulay matrix of degree $D+F$ can be obtained from the null space of the Macaulay matrix of degree $D+G$ through N intersections if $F-G \leq \left\lceil \frac{G+1}{N} \right\rceil$. (For clarity, note that Figure 1 and Theorem 3.2 concern the degree of the monomials with which the original set of polynomials $\mathbf{\Sigma}$ is multiplied (i.e., F and G) and not the degree of the

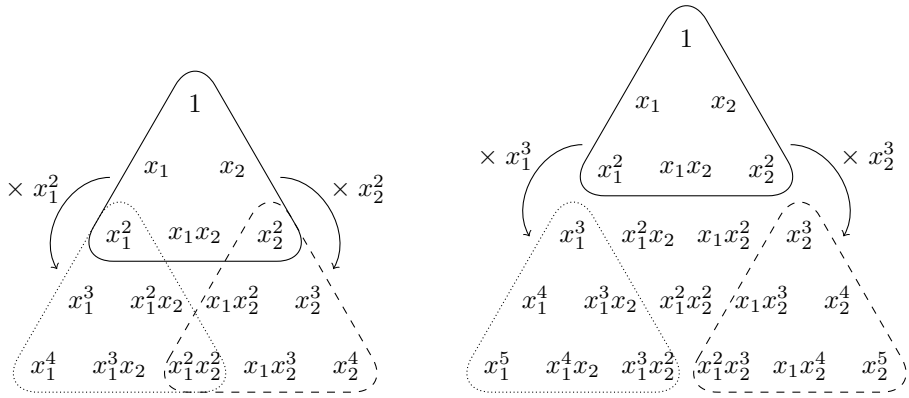


Fig. 1: All monomials of degree at most four (left figure) are the product of a monomial of degree at most two (top triangle) with $1, x_1^2$ or x_2^2. Conversely, not all monomials of degree at most five (right figure) equal the product of a monomial of degree at most two with $1, x_1^3$ or x_2^3.

polynomials that result from these products (i.e., $D+F$ and $D+G$).)

PROPOSITION 3.2. *Every monomial of degree at most F in N variables $x_1, \ldots, x_N$ can be written as the product of a monomial of degree at most G and one of the monomials 1, x_1^{F-G}, $\ldots$, x_N^{F-G} if and only if*

$$F - G \leq \left\lceil \frac{G+1}{N} \right\rceil. \tag{3.39}$$

Proof. The proof is given in Appendix A. ∎

Theorem 3.2 can now be used to accelerate Algorithm 3.2, yielding Algorithm 3.3.

Algorithm 3.3 MINISkip

Input: Set of polynomials $\mathbf{\Sigma}$ of degree D in N unknowns
Output: Macaulay null space at degree E, i.e., $\mathbf{K}_E$, in monomial basis $\mathbb{M}_E$

$\{\mathbf{K}_D, \mathbb{M}_D\} = \{\text{null}_\text{B}(\text{coef}(\mathbf{\Sigma})), \text{mon}(\mathbf{\Sigma})\}$
$d = D$
while $d \leq E$ **do**
  $\delta = \min(\lceil (d - D + 1)/N \rceil, E - d)$
  $d = d + \delta$
  $\{\mathbf{K}_d^{(0)}, \mathbb{M}_d^{(0)}\} = \{\mathbf{K}_{d-\delta}, \mathbb{M}_{d-\delta}\}$
  for $n = 1, \ldots, N$ **do**
    $\{\mathbf{K}_d^{(n)}, \mathbb{M}_d^{(n)}\} = \{\mathbf{K}_d^{(n-1)}, \mathbb{M}_d^{(n-1)}\} \cap \{\mathbf{K}_{d-\delta}, x_n^\delta \mathbb{M}_{d-\delta}\}$ (Algorithm 3.1)
  end for
  $\{\mathbf{K}_d, \mathbb{M}_d\} \leftarrow \{\mathbf{K}_d^{(n)}, \mathbb{M}_d^{(n)}\}$ (Macaulay null space of degree d)
end while

Number of intersections. As an indication of the speedup that MINISkip brings in comparison with MINI, let us compare the number of intersections performed with both methods. Each outer iteration relies on N intersections in both MINI and

MINISkip. MINI performs $\Delta = E - D$ outer iterations for a total of $N\Delta$ intersections. Let us denote the number of outer iterations in MINISkip by K. In Appendix B we show that K is approximately given by $\mathcal{O}(N \log(\Delta/N))$ (where $\Delta = E - D$), with lower and upper bound

$$\left\lceil N \log\left(\frac{\Delta}{N} + 1\right)\right\rceil \leq K \leq \lceil 1.24 N \log(\Delta + 1)\rceil. \tag{3.40}$$

The total number of intersections is thus reduced from $\mathcal{O}(N\Delta)$ to $\mathcal{O}(N^2 \log(\Delta/N))$. If $\Delta = \mathcal{O}(DN)$ (e.g., for square systems, the bound on the degree of regularity [1] implies that $\Delta \leq (N-1)(D-1)+2$), the total number of intersections is reduced from $\mathcal{O}(N^2 D)$ to $\mathcal{O}(N^2 \log D)$.

Remark 3.3 (Parallel implementation). Both Algorithms 3.2 and 3.3 were written in a sequential order, but the intersection of the $N+1$ null spaces (which is performed in the inner loop) is associative and can be performed in any order. By organizing the computations in a tree-like structure where pairs of subspaces are intersected at each level, the total number of sequential intersection steps is reduced to $\lceil \log_2(N+1) \rceil$.

3.6. Accuracy. As with many numerical algebraic methods, Macaulay null-space approaches for root finding are, in general, not backward stable [29, 23]. Nevertheless, these methods remain widely used in practice and are often competitive for overdetermined in numerical settings. In this subsection, we focus only on the accuracy of the first step of the Macaulay null-space methods: computing the null space of the Macaulay matrix. This step can be carried out in a stable manner, for example via an SVD. Our goal is to examine how the iterative construction in MINI influences the accuracy of this step.

For each intersection, a basis for the null space of $\begin{bmatrix}\mathbf{C}_1 & -\mathbf{C}_2\end{bmatrix}$ needs to be determined. This is computed using the SVD, where the right singular vectors corresponding to singular values which are equal to zero, form a basis for the null space. In practice, singular values below a threshold τ are treated as zero. Suppose the M singular values are ordered from large to small as $\sigma_1 \geq \cdots \geq \sigma_R \geq \tau > \sigma_{R+1} \geq \cdots \geq \sigma_M$. The accuracy of a computed subspace basis is characterized by the subspace angle between the true subspace and the subspace spanned by the estimated basis. Roughly speaking, for a perturbation of norm $\mathcal{O}(\epsilon_{\text{mach}})$, the error in the subspace angle is bounded by $\epsilon_{\text{mach}} / (\sigma_R - \sigma_{R+1})$ [21, Section 8.6.2].

As with PNLA [3], accuracy can degrade during intermediate null space computations in MINI. This happens when the null space is not well defined, specifically when $\sigma_R - \sigma_{R+1}$ is small. This problem is intrinsic to these methods as intermediate Macaulay matrices, whose null spaces are computed in MINI or PNLA, may have a small but nonzero σ_R. The problem is more pronounced in MINI than in PNLA, as MINI relies on more intermediate null spaces. Moreover, these additional intermediate null spaces are of "partial" Macaulay matrices—constructed using only a subset of the monomials of a given degree—which experimentally appear more likely to have a small σ_R.

To address this issue, the dimension of the null space can be intentionally overestimated, as long as the incorrectly included vectors are filtered away in subsequent intersections. If k additional singular vectors are included, the error on the estimated subspace is bounded by $\epsilon_{\text{mach}} / (\sigma_{R-k} - \sigma_{R-k+1})$ instead of $\epsilon_{\text{mach}} / (\sigma_R - \sigma_{R+1})$. As these are absolute differences, the difference of the larger singular values is usually larger, leading to a smaller subspace error. The more relevant subspace error is actually between the extended estimated subspace (of dimension $M - R + k$) and the true

subspace (of dimension $M - R$), which we suspect would instead result in a subspace error bounded by $\epsilon_{\text{mach}} / (\sigma_{R-k} - \sigma_{R+1})$. In that case, an overestimated dimension is always beneficial, provided the spurious extra vectors are removed later on. Experimentally (see section 4) we noticed that consecutive intersections removed the additional vectors as long as the threshold τ was not too large.

3.6.1. Refinement. Any error accumulated in the null space can easily be removed by an iterative solver. For instance, in the experiment of subsection 4.2, conjugate gradients on the normal equations [21, Section 11.3.9] is used after MINI to return the accuracy to machine precision. One advantage of MINI is that any intermediate null space is the null space of a known matrix, namely a (partial) Macaulay matrix. An iterative method could thus be added after any intersection, if needed. Whether this is needed could be decided based on the singular values.

3.7. Computational complexity. This subsection studies the computational complexity of MINI (Algorithm 3.2). For ease of derivation, we analyze the complexity in Big-O notation, omitting constant factors. As section 4 will show, the constant factors are small enough that the improvements of MINI are meaningful—not just in theory, but also in practice. MINI relies on computing intersections (Algorithm 3.1), whose complexity is first analyzed in subsection 3.7.1. Then, the overall complexity of MINI is given in subsection 3.7.2. When MINI is specifically used for root-solving, it is run until the degree of regularity E^*. For that case, we further work out the computational complexity, specifically for square systems (subsection 3.7.3) and systems overdetermined by one equation (subsection 3.7.4). Finally, subsection 3.7.6 summarizes the obtained complexities.

3.7.1. Null space intersections. We analyze the computational complexity of using Algorithm 3.1 to intersect two null spaces with bases $\mathbf{K}_1 \in \mathbb{C}^{(I_c+I_1)\times R_1}$ and $\mathbf{K}_2 \in \mathbb{C}^{(I_c+I_2)\times R_2}$ with I_c common monomials, which yields a null space basis $\mathbf{K} \in \mathbb{C}^{(I_c+I_1+I_2)\times R_r}$. Therefore, the encountered matrices are: $\mathbf{C}_1 \in \mathbb{C}^{I_c \times R_1}$, $\mathbf{D}_1 \in \mathbb{C}^{I_1 \times R_1}$, $\mathbf{C}_2 \in \mathbb{C}^{I_c \times R_2}$, $\mathbf{D}_2 \in \mathbb{C}^{I_2 \times R_2}$ and $\text{null}_\text{B}\big(\begin{bmatrix}\mathbf{C}_1 & -\mathbf{C}_2\end{bmatrix}\big) = \begin{bmatrix}\mathbf{X}^\text{T} & \mathbf{U}^\text{T}\end{bmatrix}^\text{T} \in \mathbb{C}^{(R_1+R_2)\times R_r}$. The products with zeros can be omitted, so we only compute $\text{null}_\text{B}\big(\begin{bmatrix}\mathbf{C}_1 & -\mathbf{C}_2\end{bmatrix}\big)$, $\mathbf{C}_1\mathbf{X}$, $\mathbf{D}_1\mathbf{X}$ and $\mathbf{D}_2\mathbf{U}$, which take[4] $\mathcal{O}\big(I_c(R_1+R_2)^2\big)$, $\mathcal{O}(I_c R_1 R_r)$, $\mathcal{O}(I_1 R_1 R_r)$ and $\mathcal{O}(I_2 R_2 R_r)$ flops, respectively. Using $R_r \leq R_1 + R_2$ brings the total to

$$\mathcal{O}\big(I_c(R_1+R_2)^2 + I_1 R_1 R_r + I_2 R_2 R_r\big) \leq \mathcal{O}\big((I_1+I_2+I_c)\max(R_1,R_2)^2\big). \tag{3.41}$$

For systems in many variables, R_1 and R_2 will be significantly smaller than I_1, I_2 and I_c. Therefore, we did not find any alternative intersection method that matched the accuracy and simplicity of our approach, while being equally or more computationally efficient. For instance, as the identity matrices in (3.18) are of size $I_2 \times I_2$ and $I_1 \times I_1$, any method that explicitly forms and uses them would likely require more than $\mathcal{O}(I_1^3 + I_2^3)$ operations, leading to a much higher computational complexity.

3.7.2. Overall complexity. We analyze the complexity of computing the null space of the Macaulay matrix of degree E of a system of S polynomials of degree D in N variables with Algorithm 3.2. To this end, we first define:

$C_d \triangleq$ the number of monomials in $\mathbb{M}_d$,

$R_{\max} \triangleq \max\limits_{d,n}\Big(\dim\Big(\text{col}\Big(\mathbf{K}_d^{(n)}\Big)\Big)\Big)$, the largest encountered null space dimension.

[4]Computing the null space of an $m \times n$ matrix via its singular value decomposition (SVD) is assumed to require $\mathcal{O}(mn^2)$ operations [21, Fig. 8.6.1].

Thus the size of a null space basis $\mathbf{K}_d^{(n)}$ is bounded by $C_d \times R_{\max}$.

Algorithm 3.2 (MINI) starts by computing the null space of the coefficient matrix $\mathbf{C}$, requiring $\mathcal{O}(SC_D^2)$ operations. All other steps are null space intersections. Each outer iteration relies on N intersections, each with a cost bounded by $\mathcal{O}(C_d R_{\max}^2)$ (see subsection 3.7.1). Thus, the total cost of Algorithm 3.2 is at most

$$\mathcal{O}\left(SC_D^2 + \sum_{d=D+1}^{E} NC_d R_{\max}^2\right). \tag{3.42}$$

To simplify this, we will first use a few properties, including Theorem 3.4.

PROPOSITION 3.4 (Asymptotic approximation of the binomial coefficient). *For large enough n and $k > 1$, using $e_{[x]} \triangleq \left(1 + \frac{1}{x}\right)^x$, the following holds*

$$\binom{kn}{n} = \mathcal{O}\left(\frac{k^n e_{[k-1]}^n}{\sqrt{n}}\right). \tag{3.43}$$

Proof. The proof is given in Appendix C.1 ∎

Remark 3.5. Note that for $0 < x < \infty$, $e_{[x]}$ increases monotonically and is bounded by $\lim_{x\to 0} e_{[x]} = 1 < e_{[x]} < e = \lim_{x\to\infty} e_{[x]}$. For illustration, we give some numerical values: $e_{[1/2]} = \sqrt{3} \approx 1.73, e_{[1]} = 2, e_{[2]} = 2.25, e_{[3]} \approx 2.37, e_{[5]} \approx 2.49$ and $e_{[10]} \approx 2.59$.

Using Theorem 3.4, the Hockey-stick identity and the fact that $C_d \leq \binom{N+d}{N}$, where $\binom{N+d}{N}$ equals the number of monomials of degree d in N variables, on (3.42) we obtain—as detailed in Appendix C.2—the upper bound on the complexity

$$\mathcal{O}\left(SC_D^2 + \sqrt{N} R_{\max}^2 \left(1 + \frac{E}{N+1}\right)^{N+1} e_{\left[\frac{E}{N+1}\right]}^{N+1}\right). \tag{3.44}$$

3.7.3. Square systems. We now make three assumptions:

1. The null space should be computed of the Macaulay matrix at the degree of regularity E^*, as required for root-solving (see section 2). Therefore, we can use the bound on the degree of regularity $E^* \leq S(D-1)+1$ [1].
2. The first part SC_D^2 is negligible, which in practice seems to hold unless both N and D are small.
3. The dimension of the null space $R_{d,n}$ does not grow much larger than the number of roots, equal to at most D^N (Bézout's theorem [13, Section 3.3]). More precisely, we assume $R_{\max} = \mathcal{O}(D^N)$. While a loose theoretical bound $R_{\max} \leq ND^N$ could be established—since at most N null spaces of dimension D^N (based on the Hilbert series) are intersected—our experiments (see Appendix C.3) suggest that $R_{\max}$ grows as $R_{\max} = \mathcal{O}(D^N)$.

Using these assumptions on (3.44), we obtain—as detailed in Appendix C.4—the approximate upper bound

$$\mathcal{O}\left(\sqrt{N} D^{3N+1} e_{[D-1]}^{N+1}\right). \tag{3.45}$$

We can compare this to the cost obtained when directly computing the null space via an SVD. The Macaulay matrix at degree E^* is of size $SC_{E^*-D} \times C_{E^*}$, meaning

its null space is computed in $\mathcal{O}(SC_{E^*-D}C_{E^*}^2)$ flops. A similar derivation—as detailed in Appendix C.5—gives the upper bound

$$\mathcal{O}\left(\frac{1}{\sqrt{N}}D^{3N}e_{[D-1]}^{3N}\right). \tag{3.46}$$

This MINI gives an exponential speedup of roughly $\mathcal{O}(e_{[D-1]}^{2N})$.

3.7.4. Overdetermined systems. A similar derivation to the one in subsection 3.7.3 could be made for overdetermined systems. The issue is that the obtained upper bound would not be sharp, because the bound on the degree of regularity, i.e., $E^* \leq S(D-1)+1$ [1], does not seem sharp for generic systems of $S > N+1$ equations. For generic systems of $S = N+1$ equations, the bound still appears to be reasonably tight and we thus focus on this case.

For $S = N+1$, the upper bound on the computational complexity via the SVD, obtained by following a similar reasoning to subsection 3.7.3, is:

$$\mathcal{O}\left(\frac{1}{\sqrt{N}}D^{3N+3}e_{[D-1]}^{3N+3}\right). \tag{3.47}$$

For MINI though, when $S = N+1$, the last iterations seem to involve null spaces of low dimension. Even though $C_{d,n}$ is the largest in these last iterations, these iterations are typically negligible in computation time. Experimentally, we notice that the iterations involving the null spaces of the largest dimension are typically the iterations where $d \approx E^*/2$. As a very rough guess, we approximate the cost by setting $E = E^*/2$, assuming all following iterations have a negligible cost. Following a similar reasoning to subsection 3.7.3 yields a rough approximation of the computational complexity of MINI when $S = N+1$:

$$\mathcal{O}\left(\sqrt{N}\frac{D^{3N+1}}{2^{N+1}}e_{[\frac{D-1}{2}]}^{N+1}\right). \tag{3.48}$$

3.7.5. Refinement. Refinement via conjugate gradients requires it_{CG} matrix-vector products per null space vector. Since the Macaulay matrix is stored sparsely, each product costs $\mathcal{O}(SC_{E-D}C_D)$ flops, giving a total cost of $\mathcal{O}(RSC_{E-D}C_D it_{\mathrm{CG}})$ flops. For square systems at the degree of regularity, using similar approximation as in subsection 3.7.3, this simplifies to $\mathcal{O}\big(\sqrt{N}D^{3N}e_{[D-1]}^{N}it_{\mathrm{CG}}/D!\big)$, which is similar in order to MINI but with a smaller constant factor.

3.7.6. Summarizing the complexity. In this section we have derived the complexity of computing the Macaulay null space with MINI or via the SVD. A summary for $S = N$ and $S = N+1$ is given in Table 1.

MINI's main advantage appears when the number of variables N grows: the complexity reduction compared to SVD is exponential in N, which explains the large practical speedups in section 4. For fixed N and increasing degree D, both methods scale similarly, as roughly $O(D^{3N})$. The reason MINI is faster for large N is that all operations are performed in the null space, whose dimension ($R_E \leq D^N$) is much smaller than the row space dimension ($C_E - R_E \approx D^N e_{[D-1]}^N$). When N is small, these dimensions are comparable and MINI can be slower in practice.

Although MINISkip (Algorithm 3.3) was not discussed, its complexity is bounded by that of MINI (assuming $R_{\max}$ is equal for both). Because iterations are skipped, the cost should be even lower.

Table 1: Comparison of approximate asymptotic complexity for large N. Only the exponential part (without additive constants) is shown here, see subsections 3.7.3 and 3.7.4. The proposed algorithm strongly reduces the computation time, especially for systems of low degree. For quadratic square systems ($S = N$ and $D = 2$), this improvement is equivalent to an improvement from a cubic to a square complexity ($(4^N)^3$ to $(4^N)^2$). When one additional equation is given ($S = N + 1$), the reduction is even larger.

	$D = 2$	$D = 3$	$D = 4$	large D
SVD when $S = N$	2^{6N}	$3^{3N}(2.25)^{3N}$	$4^{3N}(2.37)^{3N}$	$D^{3N}e^{3N}$
MINI when $S = N$	2^{4N}	$3^{3N}(2.25)^{N}$	$4^{3N}(2.37)^{N}$	$D^{3N}e^{N}$
SVD when $S = N + 1$	2^{6N}	$3^{3N}(2.25)^{3N}$	$4^{3N}(2.37)^{3N}$	$D^{3N}e^{3N}$
MINI when $S = N + 1$	$2^{2.79N}$	3^{3N}	$4^{3N}(1.08)^{N}$	$D^{3N}\left(\frac{e}{2}\right)^{N}$

4. Experiments. This section aims to show experimentally that the proposed methods are significantly faster than state-of-the-art methods for computing the Macaulay null space, while maintaining high accuracy. Four methods are compared: MINI (Algorithm 3.2), MINISkip (Algorithm 3.3), PNLA [3] (specifically the SVD-based version) and SVD (directly obtaining the null space through the SVD on the full Macaulay matrix).

First, subsection 4.1 compares the runtime of the different methods. It validates the theoretical complexity from subsections 3.7.3 and 3.7.4 and shows the significant reduction in computation time obtained with MINI. Subsection 4.2 then compares the accuracy, showing that it is close to that of SVD or PNLA and that it can be improved through iterative refinement. For further insight, some additional experiments are included. Subsection 4.3 studies the influence of the tolerance τ. Although most choices lead to an accurate solution, a larger τ generally further reduces the error.

Experiment setup. All experiments use dense systems consisting of S polynomials in N variables, each of degree D. For square systems ($S = N$), the coefficients are randomly drawn from a standard normal distribution. To generate overdetermined systems ($S > N$) with a certain number of roots L, the coefficient matrices are generated as $\mathbf{C} = (\text{null}_\text{B}(\mathbf{V}_D^\text{T})\mathbf{W})^\text{T}$ instead, where $\mathbf{V}_D$ is a matrix whose L columns are multivariate Vandermonde vectors of degree D in the L roots $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}$, and where $\mathbf{W} \in \mathbb{R}^{(C_D - L)\times S}$ and $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}$ are randomly drawn from a standard normal distribution. Unless stated otherwise, the null space is computed of the Macaulay matrix at the degree of regularity E^*. For all methods, the construction of the required Macaulay matrix (or the necessary parts of the Macaulay matrix for PNLA) is not included in the timing experiments, as the exact implementation can have a great impact on the timings. This choice advantages PNLA and SVD, which rely on the formation of large matrices, compared to MINI and MINISkip. The error on the estimated null space is computed by first orthogonalizing its basis, then evaluating the following:

$$\frac{\|\mathbf{M}_{E^*}\mathbf{K}_{E^*}\|_F}{\|\mathbf{M}_{E^*}\|_F}, \tag{4.1}$$

where $\mathbf{M}_{E^*}$ is the Macaulay matrix and $\mathbf{K}_{E^*}$ is an (orthonormal) basis for the estimated null space.

Roots are recovered from the Macaulay null space via a tensor decomposition (following the method described in subsection 2.2, which corresponds to [41, Algorithm 1]). Since the decomposition method can greatly affect accuracy, we report results for both GEVD (see [2]) and GESD (see [18, 5]). The root error is defined as

$$\left\| \mathbf{X} - \hat{\mathbf{X}}\mathbf{\Pi} \right\|_F, \tag{4.2}$$

where $\mathbf{X}$ and $\hat{\mathbf{X}}$ contain the reference and estimated roots, respectively, and $\mathbf{\Pi}$ aligns them by permutation. Each root is represented in projective coordinates by appending a homogenization variable (a row of ones) and normalizing to unit Euclidean norm. For square systems, reference roots are obtained by refining the estimated roots obtained via SVD and GEVD using nonlinear least squares. For overdetermined systems, the roots are known by construction.

All experiments were performed in MATLAB 2023b on a laptop with an AMD Ryzen 7 PRO 5850U CPU (8 cores, 16 threads) and 32 GB RAM. The code of the proposed algorithms and the code to reproduce these experiments can be found at https://www.tensorlabplus.com/papers/widdershoven2025mini.html.

4.1. Computation time. We first benchmark square systems, as these are commonly reported in the literature and provide a clear baseline, even though faster root-solving methods such as homotopy continuation methods exist for this case (see Appendix D for an experimental comparison). We then study overdetermined systems, analyzing how runtime depends on the number of variables, equations and roots. The overdetermined case is interesting because Macaulay null-space methods benefit from their numerical linear-algebra foundation, allowing redundancy to stabilize the computation and helping to mitigate noise in a least-squares sense, making them an effective option for such systems. The results show that the proposed methods scale better than the state of the art for computing the Macaulay null space, particularly with the number of variables.

4.1.1. Square systems. To compare the runtime of the different Macaulay null-space methods on square systems ($S = N$), we run experiments for different degrees D with an increasing number of variables N. To ensure accurate time measurements, each experiment is repeated until it runs for at least three seconds in total. Figure 2 presents the results, validating the asymptotic complexities derived in subsection 3.7.3 and determining that MINI and MINISkip outperform PNLA and SVD for $N \gtrapprox 5$. MINISkip offers moderate improvement over MINI, except for $D = 2$, where skipping is not possible.

For $N = 8$ and $D = 2$, the null space of dimension 256 of a Macaulay matrix of size 24024×12870 is computed in 3.5 s with MINI, instead of 600 s with SVD or 110 s with PNLA. For $N = 11$ and $D = 2$, a null space of dimension 2048 of the Macaulay matrix of size 1847560×705432 is computed in under 3 hours (10^4 seconds) using MINI—a problem size infeasible for current Macaulay null-space methods. Simply storing the null space basis (705432×2048) requires 11 GB of memory.

4.1.2. Overdetermined systems. The first set of experiments considers quadratic ($D = 2$) systems with a single root ($L = 1$), overdetermined by one, two or N equations, with results shown in Figure 3. Additional experiments examine the effect of the number of equations S and the number of roots L, with results shown in Figure 4.

The results reveal several trends. MINI (and MINISkip) become faster as the number of equations increases, because the dimensions of all null spaces throughout

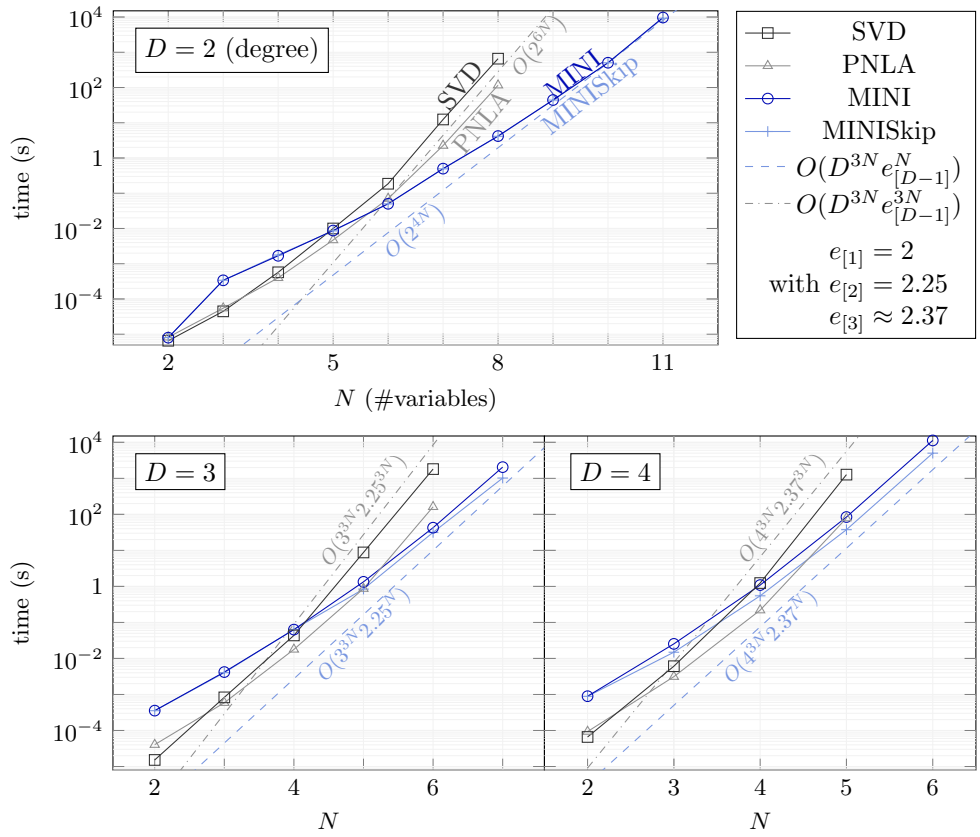


Fig. 2: MINI and MINISkip scale better in the number of variables N. This is true regardless of the degree D. MINISkip offers a moderate further improvement over MINI, except for $D = 2$, where none of the iterations can be skipped and MINISkip and MINI are thus identical. This experiment also validates the rough asymptotic complexities from Table 1, shown in dashed lines.

the algorithms become smaller. In contrast, the runtime of SVD and PNLA depends primarily on the degree of regularity, producing step-like patterns in Figures 3 and 4. Adding equations that do not reduce the degree of regularity E^* slightly increases the runtime of SVD and PNLA because the Macaulay matrix becomes taller. When E^* is decreased, the computation time of SVD and PNLA is significantly reduced. MINI is not affected much by the degree of regularity and consequently overestimating the degree of regularity has little to no effect on the computation time. This is also visible when varying the number of roots (see Figure 4), which can significantly change the runtime of SVD and PNLA as it effects the degree of regularity, but has little impact on MINI.

For $S = N + 1$ (Figure 3), MINI (and MINISkip) performs significantly better than SVD and PNLA. Compared to $S = N$ (Figure 2), this can be explained by MINI accelerating with an additional equation, while SVD and PNLA are heavily slowed down by the higher degree of regularity. The theoretical complexities for $S = N + 1$ derived in subsection 3.7.4 are shown in dotted lines in Figure 3. These align with

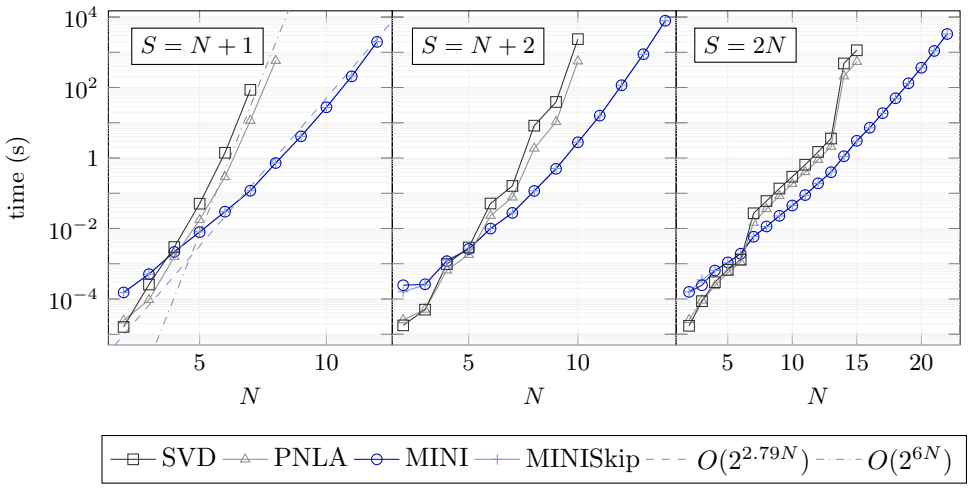


Fig. 3: Timings for computing the one-dimensional null space ($L = 1$) at the degree of regularity for quadratic ($D = 2$) systems overdetermined by one, two and N equations. The dotted lines give the approximate complexities, summarized in Table 1. Our proposed method clearly outperforms the state-of-the-art methods.

observed trends, although experimental costs are slightly higher due to the increasing number of monomials in later iterations.

4.2. Accuracy comparison. This section evaluates the accuracy of the proposed methods in comparison with SVD and PNLA. We first consider noiseless randomly generated square systems, as used in Figure 2, which provides a baseline where all methods perform similarly. To examine a more difficult problem, we then modify the system by shifting one variable, which moves all roots away from the origin and introduces large errors in any Macaulay null-space method. To argue that the conclusions from these first experiments also hold for overdetermined systems, we repeat the first experiment with double the equations. Finally, we test the same experiment with noise, since robustness to perturbations is an important property of numerical Macaulay-based approaches.

4.2.1. Noiseless systems. To evaluate the accuracy, we conduct a Monte Carlo simulation with 100 random systems of $S = 5$ polynomials of degree $D = 3$ in $N = 5$ variables[5]. Figure 5 presents the error distribution for the Macaulay null space at the degree of regularity plus one[6] ($E^* + 1 = 11$), for different methods.

In general, all methods achieve accuracy close to machine precision. Compared to SVD or PNLA, MINI and MINISkip estimate the null space slightly less accurately, due to some error accumulations in their additional intermediate steps. However, any computed null space basis can be efficiently refined using CG at modest extra cost: MINI takes about 1 s, and typically fewer than 10 CG iterations are needed, with

[5] $D = 3$ was chosen such that MINI and MINISkip differ. $S = N = 5$ was then selected to allow 100 Monte Carlo simulations to run within a reasonable amount of time.

[6] The roots could be obtained from the Macaulay null space at the degree of regularity, but this requires the decomposition of a tensor for which GEVD and GESD cannot be used [37], making it much more difficult to solve.

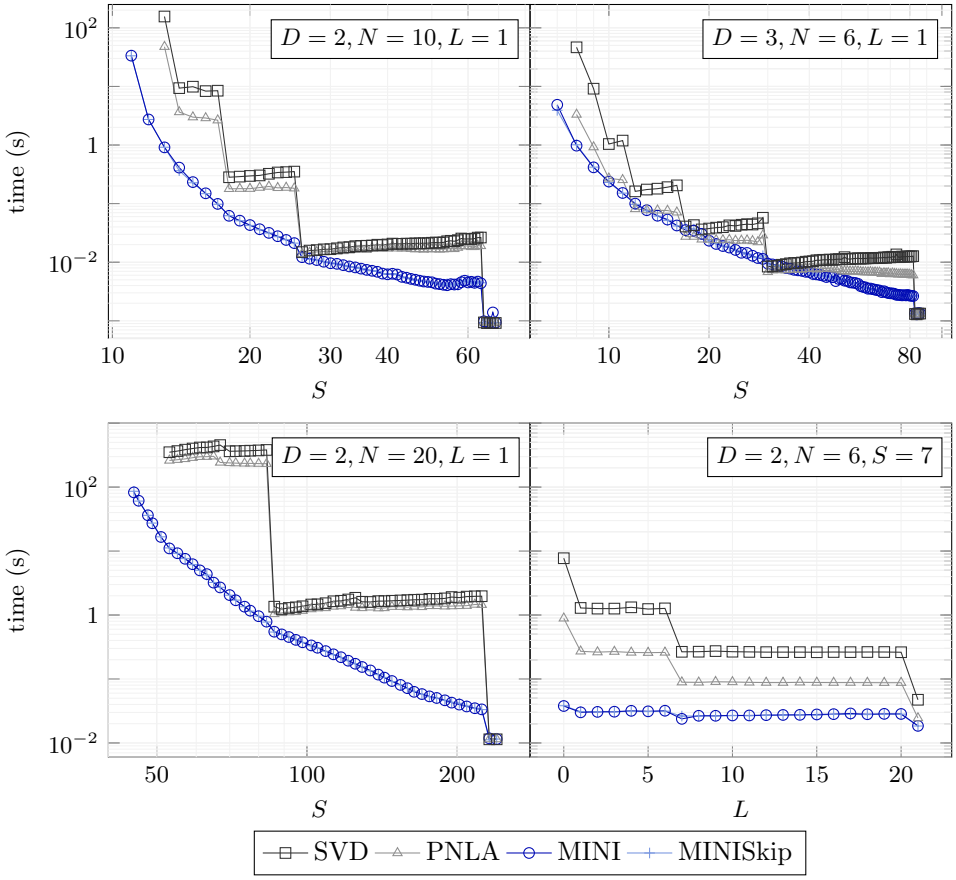


Fig. 4: MINI and MINISkip become faster with additional equations because the null space dimension decreases, while PNLA and SVD depend mainly on the degree of regularity (visible in step-like patterns). For all four methods, the number of roots has little effect unless it alters the degree of regularity.

each iteration adding about 0.04 s, for a total of about 0.4 s. Moreover, the accuracy loss on the Macaulay null space does not seem to translate in an accuracy loss on the obtained roots. Given this, the significantly faster computation times of MINI and MINISkip make them an effective choice for solving systems with many variables.

4.2.2. Shifted systems. To tackle a more challenging problem for Macaulay-based methods, we repeat the previous experiment, but shift the first variable by five units, replacing each polynomial $p(\mathbf{x})$ by $q(\mathbf{x})$ defined as

$$q(x_1, x_2, x_3, x_4, x_5) = p(x_1 + 5, x_2, x_3, x_4, x_5).$$

The results are shown in Figure 6. This seemingly minor change in one coordinate leads to substantial accuracy loss for all Macaulay null-space methods. Interestingly, in this setting, MINI and MINISkip recover the roots with significantly higher accuracy. The shift alters the Macaulay null space, for which the Vandermonde vectors in the roots form a basis. This basis is now ill-conditioned as the first coordinate of the

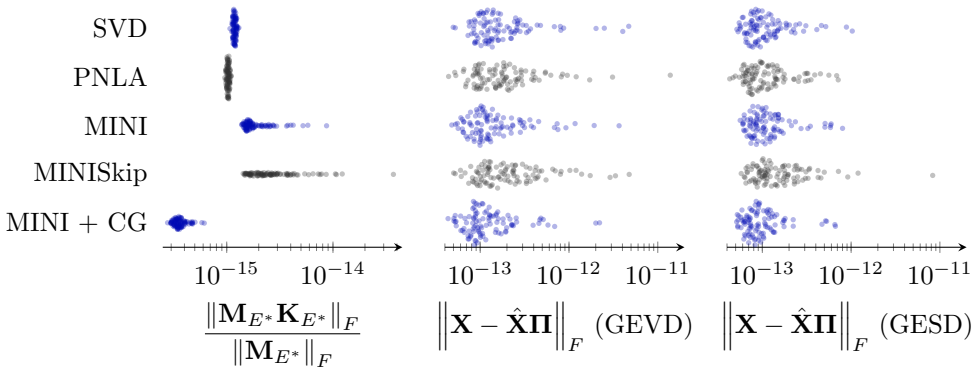


Fig. 5: All methods achieve errors near machine precision. MINI and MINISkip show slightly higher error on the null space than SVD and PNLA, but when computing roots, all methods perform similarly. MINI and MINISkip remain attractive for large systems because they maintain low error while significantly reducing computation time (see Figure 2). The null space error can be further mitigated with conjugate gradients (CG). Both tensor decomposition approaches (GEVD and GESD) yield similar root error. Each dot represents one of 100 runs per method, vertically offset to show distribution.

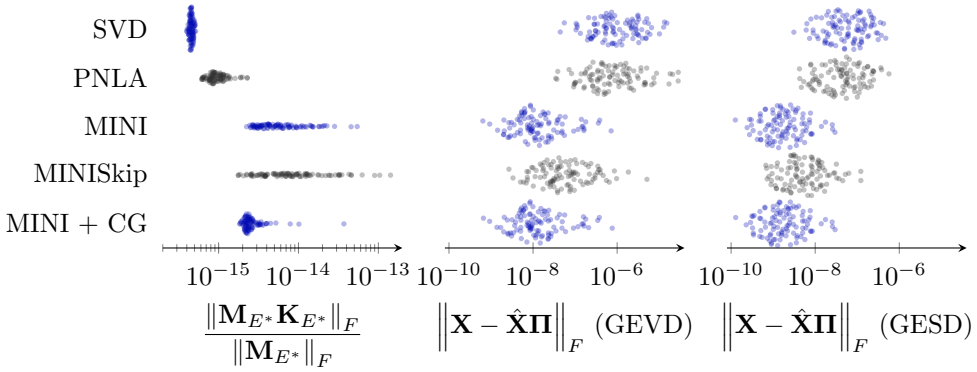


Fig. 6: For this more challenging problem, MINI and MINISkip lose more accuracy on the null space compared to SVD and PNLA, but remain substantially more accurate when estimating the roots. This advantage is observed for both tensor decomposition methods (GEVD and GESD), with GESD performing noticeably better than GEVD in this experiment. The CG refinement was capped at 100 iterations and in most cases it did not fully converge within this limit. This refinement had little impact on the root error. Each dot represents one of 100 runs per method, vertically offset to illustrate the distribution of errors.

roots, which is on average around 5, is raised to powers up to 11. When the null space is computed in a single step, the least-squares solution obtained via SVD accurately estimates large values, while tolerating larger relative errors on small values. By contrast, MINI and MINISkip iteratively shift and intersect, which better preserves the

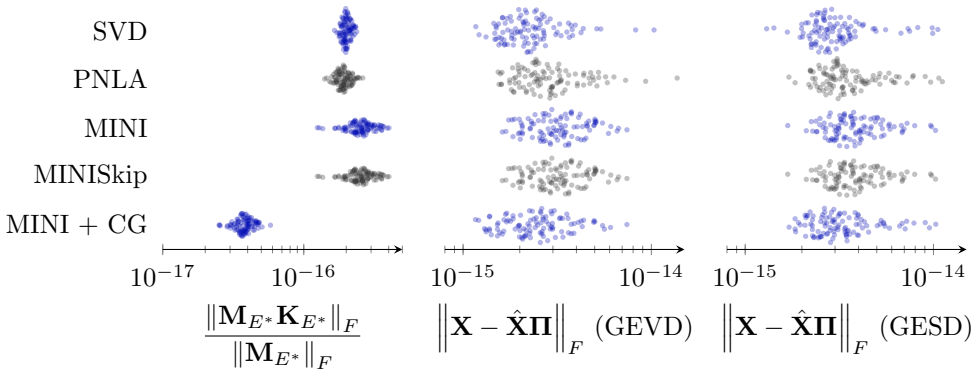


Fig. 7: MINI remains effective for overdetermined systems. In these sets of experiments, all methods perform almost the same, giving accuracies of around machine precision. Each dot represents one of 100 runs per method and is vertically shifted to illustrate the distribution of the error.

relative scaling between rows of the null space basis and improves the estimation of rows with small magnitude. This can then lead to a more accurate Vandermonde structure and, consequently, better root recovery.

4.2.3. Overdetermined systems. In this section, we want to test if the accuracy remains good for overdetermined systems. Therefore, we perform the same experiment as in subsection 4.2.1, but with double the equations ($S = 10$) and generated as specified in the experiment setup to have $L = 5$ roots. Figure 7 shows the results, confirming that the conclusions also hold for overdetermined systems. To obtain these results, the tolerance τ was set to 4×10^{-2}. A smaller value resulted in a slight increase of all errors for MINI and MINISkip.

4.2.4. Noisy systems. Even when the noise level is significantly higher than machine precision, MINI can compute the null space of the Macaulay matrix to a reasonable accuracy. To test this, the same experiment as in subsection 4.2.3 is repeated, but with noise added to the coefficients such that the mean relative error on the coefficients is 10^{-4}. Figure 8 shows the results, confirming that, even with noise, MINI is effective.

4.3. Influence of the numerical tolerance on the null space accuracy. This subsection includes two additional experiments to better understand how the null space error evolves and how the tolerance τ affects it.

First, we repeat the experiment from subsection 4.2.1 using different tolerances τ (fixed at 10^{-3} in subsection 4.2). Figure 9 shows the results, supporting the conclusion from subsection 3.6: a larger tolerance τ generally leads to a slightly lower error, provided τ is not excessively high, which could otherwise lead to an overestimated (final) null space dimension. It should be noted that all tolerances still lead to a small error and that this choice is thus not critical.

The second experiment is designed to better understand the effect of small singular values in intermediate steps and the role of the tolerance τ in these cases. We use systems of the form

$$\{c_{s,0} + c_{s,1}x_1^2 + c_{s,2}x_2^2 + c_{s,3}x_3^2 + (c_{s,3} + \alpha c_{s,4})x_4^2 = 0\}_{s=1}^4, \tag{4.3}$$

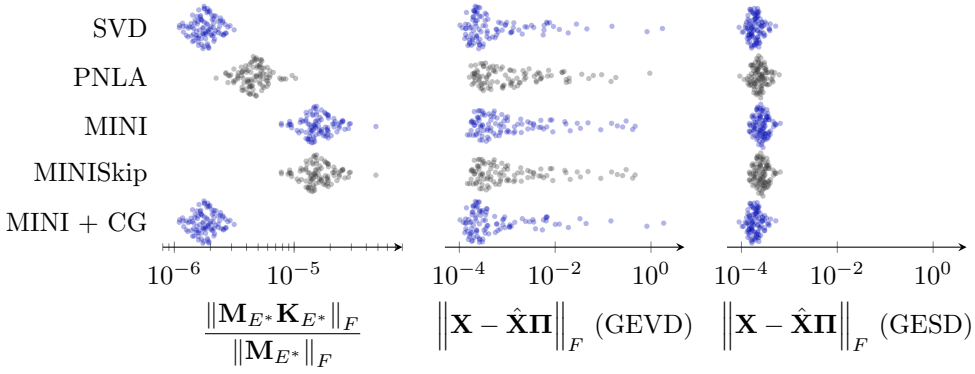


Fig. 8: MINI remains effective in noisy settings. In this experiment, the null space error obtained with MINI is on average one order of magnitude larger than with SVD, while the root error is about the same. This loss of accuracy on the null space can be mitigated by applying conjugate gradient refinement to the null space. For root recovery, using a more robust tensor decomposition method such as GESD instead of GEVD [18] further improves accuracy significantly. Each dot represents one of 100 runs per method and is vertically shifted to illustrate the distribution of the error.

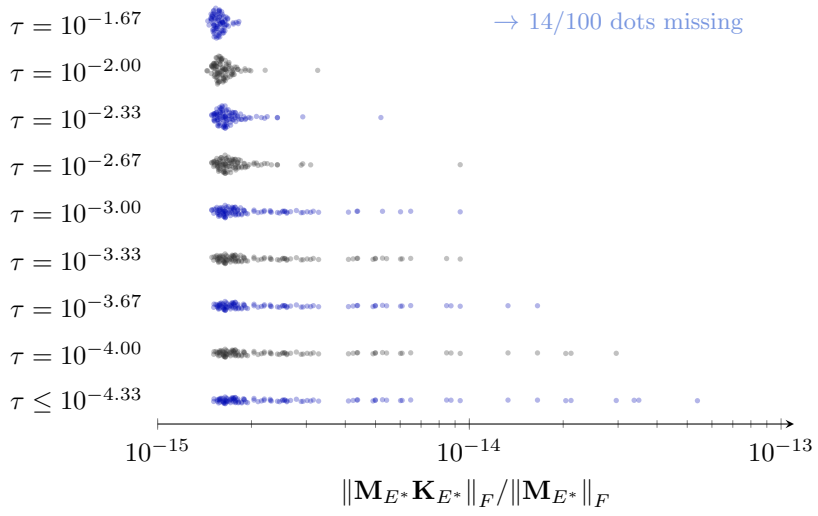


Fig. 9: A larger value of the tolerance τ yields a higher accuracy and reduces the number of outliers, aligning with the analysis in subsection 3.6. However, excessively large τ values lead to an overestimation of the null space dimension, which was the case in 14/100 of the runs with $\tau = 10^{-1.67}$. The smallest singular value significantly larger than machine precision encountered was $\sigma = 10^{-4.18}$, meaning MINI behaved identically for all $\tau < 10^{-4.18}$. Each dot represents one of 100 runs per tolerance τ and is vertically shifted to illustrate the distribution of the error.

for varying α and where $c_{s,n}$ are randomly drawn from a normal distribution for all s and n. For any α, the Macaulay matrix of degree $E^* = 4$ has a 16-dimensional null space. MINI computes this through $N(E^* - D) = 8$ intersections (i.e., the inner loop in Algorithm 3.2 is run eight times), with each intersection computing the null space of a (partial) Macaulay matrix. The issue is that these (partial) Macaulay matrices can have small but nonzero singular values, which results in an unclear null space dimension. In this case, the null spaces obtained after the fifth and sixth intersection turn out to have dimension 19 and 18 when $\alpha = 1$, and dimension 20 and 20 when $\alpha = 0$, respectively. When α is in between 0 and 1, the smallest nonzero singular value in those intersections seems to be approximately of magnitude α. Figure 10 shows for which weights α and which tolerances τ on the singular values, MINI computes the null space of the correct dimension.

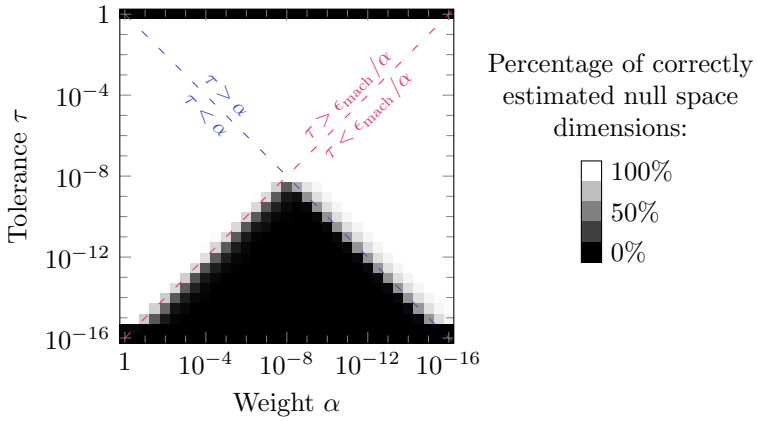


Fig. 10: Proportion of correctly estimated null space dimensions out of 100 randomly generated systems of the form (4.3) with varying weights α and varying tolerance τ. A run is considered successful if the resulting null space is of the correct dimension ($L = 16$), in which case the error (see (4.1)) was always below 5e-16. In this experiment, for tolerances between $1 > \tau > \sqrt{\epsilon_{\text{mach}}}$, MINI always computed a null space of the correct dimension. For further analysis, see subsection 4.3.

By construction, in the fifth and sixth intersection, singular values of magnitude α will be present. The tolerance τ determines whether their corresponding singular vectors are included in the null space ($\tau > \alpha$) or not ($\tau < \alpha$).

In exact arithmetic, if $\alpha > 0$, these singular values are nonzero, meaning their corresponding singular vectors are not part of the null space ($\tau < \alpha$). However, if these singular vectors are included in the fifth and sixth iteration ($\tau > \alpha$), they will be removed in the seventh intersection, as is the case when $\alpha = 0$. Thus, in both cases, the dimension of the null space at the degree of regularity will be correct.

In finite precision though, when excluding a singular vector corresponding to a singular value of magnitude α (i.e., $\tau < \alpha$), the estimated null space will only be accurate up to $\epsilon_{\text{mach}}/\alpha$, see subsection 3.6. This can be a problem in the next intersection, as all singular values that should be zero (or ϵ_{mach} in practice) are now of magnitude $\epsilon_{\text{mach}}/\alpha$. If $\tau < \epsilon_{\text{mach}}/\alpha$, the null space dimension will be underestimated, which can not be corrected in further intersections. Figure 10 indeed shows that if $\tau < \alpha$ and $\tau < \epsilon_{\text{mach}}/\alpha$ the dimension is incorrectly estimated.

By choosing $\tau > \sqrt{\epsilon_{\text{mach}}}$, one can guarantee that $\tau > \alpha$ or $\tau > \epsilon_{\text{mach}}/\alpha$ will hold. In this experiment, $\tau > \sqrt{\epsilon_{\text{mach}}}$ thus ensures that the null space dimension at the degree of regularity is correctly estimated, but this conclusion seems to hold more generally. This fits well with the conclusion of the experiment in Figure 9, which suggested that the tolerance should be chosen quite large to improve the accuracy.

5. Conclusion and future work. We presented MINI, a new approach for computing the null space of the Macaulay matrix. MINI computes $\text{null}(\mathbf{M}_d)$ from $\text{null}(\mathbf{M}_{d-1})$ for $d = D+1, \ldots, E$ through efficient null space intersections, without ever forming the Macaulay matrices $\{\mathbf{M}_d\}_{d=D+1}^{E}$. Since the null space dimension grows exponentially slower than the row space dimension in the Macaulay construction, MINI achieves an exponential reduction in computation time compared to the state-of-the-art alternatives.

This efficiency is reflected in the derived complexity bounds. For instance, for quadratic square systems in N variables, the complexity is reduced from $\mathcal{O}(2^{6N})$ to $\mathcal{O}(2^{4N})$. Experiments confirm that this also translates into substantial runtime reductions in practice, including for overdetermined systems. For example, for systems of $S = 30$ equations in $N = 15$ variables, SVD requires 1100 s, PNLA 540 s, while MINI completes in 3 s. In our experiments, the method tends to outperform existing Macaulay null-space approaches from about $N \gtrapprox 5$. While the null space accuracy is slightly lower, root accuracy is largely unaffected (or even improved) and can be refined at modest cost. We also showed that in some cases intersections can be skipped, leading to an even faster algorithm.

Future work could be geared towards an even better strategy to reduce the number and the size of required intersections. One option is to remove the requirement that the final null space is that of a Macaulay matrix of full degree, which was always imposed here. Although MINI relies on monomial shifts, an adaptation for Chebyshev Macaulay matrices is also worth exploring for their favorable numerical properties. Refinement of obtained null spaces could be studied in more detail to potentially accelerate its computation and convergence.

Acknowledgments. The authors want to thank their colleagues from ESAT for the many interesting discussions on the topic. They especially thank Nico Vervliet for the many fruitful suggestions and observations. The authors also acknowledge the use of generative AI tools for language refinement in this work. Finally, they recognize that the basics of this method were discovered in parallel by Lukas Vanpoucke, Sibren Lagauw and Bart De Moor [38], although their results have not been published.

Appendix A. Maximum degree skipping.

Proof of Theorem 3.2. We will prove both directions separately.

(⇒) Necessity: Assume, for contradiction, that

$$F - G > \left\lceil \frac{G+1}{N} \right\rceil, \tag{A.1}$$

$$\Rightarrow \quad F - G \geq \left\lceil \frac{G+1}{N} \right\rceil + 1, \tag{A.2}$$

$$\Rightarrow \quad N(F-G-1) \geq N\left\lceil \frac{G+1}{N} \right\rceil \geq G+1. \tag{A.3}$$

This inequality ensures that we can distribute $G+1$ units of degree across N variables such that each variable receives at most $F-G-1$ degrees.

Consider the monomial:

$$\mu = x_1^{F-G-1} x_2^{F-G-1} \cdots x_Q^{F-G-1} x_{Q+1}^{R}, \tag{A.4}$$

where $Q = \left\lfloor \frac{G+1}{F-G-1} \right\rfloor$ and $R = (G+1) \mod (F-G-1)$. This construction ensures that $\deg(\mu) = G+1$. By the previous inequality, such a distribution is possible because $G+1 \leq N(F-G-1)$.

Now, observe that no variable has a power larger than or equal to $F-G$. Thus, factoring out a monomial of the form x_i^{F-G} from μ would never result in a monomial. Thus, μ can not be factored as the product of a monomial of degree at most G and any monomial of the form x_i^{F-G} or 1. This contradicts the assumption, establishing the necessity of (3.39).

(⇐) Sufficiency: Assume now that

$$F - G \leq \left\lceil \frac{G+1}{N} \right\rceil. \tag{A.5}$$

Consider any monomial μ of degree H with $H \leq F$. We need to show that μ can be written as the product of a monomial of degree at most G and one of the monomials $1, x_1^{F-G}, \ldots, x_N^{F-G}$.

If $H \leq G$, then μ itself is of degree at most G, and we can simply factor 1.

If $H \geq G+1$, as all exponents are integers, at least one of the N exponents in μ must be at least:

$$\left\lceil \frac{H}{N} \right\rceil \geq \left\lceil \frac{G+1}{N} \right\rceil \geq F - G. \tag{A.6}$$

Let x_n be a variable whose exponent in μ is at least $F-G$. Factoring x_n^{F-G} from μ leaves a monomial of degree:

$$H - (F-G) \leq F - (F-G) = G. \tag{A.7}$$

Thus, every monomial of degree at most F can be factored into a product of a monomial of degree at most G and one of the monomials $1, x_1^{F-G}, \ldots, x_N^{F-G}$. ∎

Appendix B. Number of intersections when skipping degrees. In Algorithm 3.3, the recurrence relation

$$d_{k+1} = d_k + \left\lceil \frac{d_k - D + 1}{N} \right\rceil \tag{B.1}$$

is used, starting from $d_0 = D$. The question is how many steps K are needed such that d_K becomes larger than or equal to E.

Let us do a change of variables $x_k = d_k - D$ such that the problem becomes

$$x_{k+1} = x_k + \left\lceil \frac{x_k + 1}{N} \right\rceil \tag{B.2}$$

from $x_0 = 0$ to $x_K = \Delta = E - D$. Define α_k such that $\lceil (x_k + 1)/N \rceil = (x_k + \alpha_k)/N$. Thus, α_k satisfies $1 \leq \alpha_k \leq N$, and the recurrence relation is rewritten to

$$x_{k+1} = \frac{N+1}{N} x_k + \frac{\alpha_k}{N}. \tag{B.3}$$

For a lower bound on the number of iterations, we assume all α_k are as large as possible, namely equal to N. For an upper bound we do the opposite, setting them to 1 instead. This gives, using either $\alpha = N$ or $\alpha = 1$:

$$x_{k+1} = \frac{N+1}{N} x_k + \frac{\alpha}{N} \tag{B.4}$$

$$\iff x_K = \left(\frac{N+1}{N}\right)^K \left(x_0 + \sum_{k=1}^{K} \frac{\alpha}{N} \left(\frac{N}{N+1}\right)^k \right), \tag{B.5}$$

$$= \left(\frac{N+1}{N}\right)^K \left(x_0 + \frac{\alpha}{N} \sum_{k=1}^{K} \left(\frac{N}{N+1}\right)^k \right), \tag{B.6}$$

$$= \left(\frac{N+1}{N}\right)^K \left(x_0 + \alpha \left(1 - \left(\frac{N}{N+1}\right)^K \right) \right), \tag{B.7}$$

$$= (x_0 + \alpha) \left(\frac{N+1}{N}\right)^K - \alpha, \tag{B.8}$$

$$\iff K = \log_{\frac{N+1}{N}} \frac{x_K + \alpha}{x_0 + \alpha}, \tag{B.9}$$

$$= \beta(N) N \log \frac{x_K + \alpha}{x_0 + \alpha} \quad \text{with} \quad \beta(N) = \frac{1}{N \log \frac{N+1}{N}}. \tag{B.10}$$

Note that $1.24 > \beta(2) \geq \beta(N) > \lim_{N\to\infty} \beta(N) = 1$ for all N satisfying $2 \leq N < \infty$. Now using $x_0 = 0$, $x_K = \Delta$ and either $\alpha = N$ or $\alpha = 1$, yields the lower and upper bounds $N \log(\Delta/N + 1) \leq K \leq 1.24 N \log(\Delta + 1)$.

For the average case, we assume that α_k is uniformly distributed with mean $(N+1)/2$. The reasoning is that $x_k + 1 \mod N$ for large x_k behaves quasi-randomly. The expected value of K can then be determined following a similar reasoning as in (B.4)–(B.10). This yields $K = \mathcal{O}(N \log(2\Delta/(N+1) + 1)) \approx \mathcal{O}(N \log(\Delta/N))$.

Appendix C. Derivation of the asymptotic complexity.

C.1. Asymptotic approximation of the binomial coefficient.

Proof of Theorem 3.4.

$$\binom{kn}{n} = \frac{(kn)!}{n!(kn-n)!} \tag{C.1}$$

$$\Downarrow \text{Stirling's approximation, valid for large } n \tag{C.2}$$

$$\sim \sqrt{\frac{kn}{2\pi n(kn-n)}}\frac{(kn)^{kn}}{n^n(kn-n)^{kn-n}} \tag{C.3}$$

$$= \sqrt{\frac{k}{2\pi n(k-1)}}\left(\frac{kn}{n}\right)^n\left(\frac{kn}{kn-n}\right)^{kn-n} \tag{C.4}$$

$$= \sqrt{\frac{k}{2\pi n(k-1)}}k^n\left(\left(1+\frac{1}{k-1}\right)^{k-1}\right)^n \tag{C.5}$$

$$= \frac{1}{\sqrt{2\pi n}}\sqrt{\frac{k}{k-1}}k^n e^n_{[k-1]} \tag{C.6}$$

$$= \mathcal{O}\left(\frac{k^n e^n_{[k-1]}}{\sqrt{n}}\right). \tag{C.7}$$ ∎

C.2. Simplification of the overall complexity. Starting from (3.42), (3.44) is derived as follows:

$$\mathcal{O}\left(SC_D^2 + NR_{\max}^2\sum_{d=D+1}^{E} C_d\right), \tag{C.8}$$

$$\Downarrow\ C_d \le \binom{N+d}{N}, \tag{C.9}$$

$$\le \mathcal{O}\left(SC_D^2 + NR_{\max}^2\sum_{d=D+1}^{E}\binom{N+d}{N}\right), \tag{C.10}$$

$$\Downarrow \text{ Hockey-stick identity: } \sum_{i=0}^{s}\binom{r+i}{r} = \binom{r+s+1}{r+1}, \tag{C.11}$$

$$< \mathcal{O}\left(SC_D^2 + NR_{\max}^2\binom{N+E+1}{N+1}\right), \tag{C.12}$$

$$\Downarrow \text{ Theorem 3.4,} \tag{C.13}$$

$$= \mathcal{O}\left(SC_D^2 + \sqrt{N}R_{\max}^2\left(1+\frac{E}{N+1}\right)^{N+1} e^{N+1}_{[\frac{E}{N+1}]}\right). \tag{C.14}$$

C.3. Maximum null space dimension throughout MINI. In experiments, we observed that $R_{d,n}$ occasionally exceeds D^N during inner iterations but not by much, see Table 2.

Table 2: Experimental maximum null space dimension $R_{\max}$ throughout MINI for random systems of degree D in N variables, suggesting that $R_{\max} = \mathcal{O}(D^N)$ holds.

D	3	5	10	2	2	2	3	4
N	2	2	2	3	5	10	3	4
$R_{\max}$	10	26	101	10	36	1088	29	261
D^N	9	25	100	8	32	1024	27	256
$R_{\max}/D^N$	1.11	1.04	1.01	1.25	1.13	1.06	1.07	1.02

C.4. Simplification of the complexity for square systems (MINI). We start from (3.44), to compute (3.45), the upper bound for the complexity of MINI.

$$\mathcal{O}\left(SC_D^2 + \sqrt{N}R_{\max}^2\left(1+\frac{E}{N+1}\right)^{N+1} e_{\left[\frac{E}{N+1}\right]}^{N+1}\right), \tag{C.15}$$

$$\Downarrow\ E^* \leq S(D-1)+1 \text{ (assumption 1)} \tag{C.16}$$

$$\leq \mathcal{O}\left(SC_D^2 + \sqrt{N}R_{\max}^2\left(1+\frac{N(D-1)+1}{N+1}\right)^{N+1} e_{\left[\frac{N(D-1)+1}{N+1}\right]}^{N+1}\right), \tag{C.17}$$

$$= \mathcal{O}\left(SC_D^2 + \sqrt{N}R_{\max}^2\left(D+\frac{2-D}{N+1}\right)^{N+1} e_{\left[D-1-\frac{2-D}{N+1}\right]}^{N+1}\right), \tag{C.18}$$

$$\Downarrow\ D \geq 2 \Rightarrow \frac{2-D}{N+1} \leq 0 \tag{C.19}$$

$$\leq \mathcal{O}\Big(SC_D^2 + \sqrt{N}R_{\max}^2 D^{N+1} e_{[D-1]}^{N+1}\Big), \tag{C.20}$$

$$\Downarrow\ SC_D^2 \text{ is negligible (assumption 2)} \tag{C.21}$$

$$\approx \mathcal{O}\Big(\sqrt{N}R_{\max}^2 D^{N+1} e_{[D-1]}^{N+1}\Big), \tag{C.22}$$

$$\Downarrow\ R_{\max} = \mathcal{O}\big(D^N\big) \text{ (assumption 3)} \tag{C.23}$$

$$= \mathcal{O}\Big(\sqrt{N}D^{3N+1} e_{[D-1]}^{N+1}\Big). \tag{C.24}$$

C.5. Simplification of the complexity for square systems (SVD). A similar derivation to Appendix C.4 for computing the null space via SVD yields (3.46):

$$\mathcal{O}(SC_{E^*-D}C_{E^*}^2) \tag{C.25}$$

$$\Downarrow\ E^* \leq S(D-1)+1,\ C_d \leq \binom{N+d}{N} \text{ and Theorem 3.4,} \tag{C.26}$$

$$\leq \mathcal{O}\left(S\frac{\left(D+\frac{1-D}{N}\right)^N e_{\left[D-1+\frac{1-D}{N}\right]}^N}{\sqrt{N}}\left(\frac{\left(D+\frac{1}{N}\right)^N e_{\left[D-1+\frac{1}{N}\right]}^N}{\sqrt{N}}\right)^2\right) \tag{C.27}$$

$$\Downarrow \begin{cases} D \geq 2 \Rightarrow \frac{1-D}{N+1} < 0, \\ \left(D+\frac{1}{N}\right)^N = D^N\left(1+\frac{1}{DN}\right)^N = D^N \sqrt[D]{e_{[DN]}} < eD^N, \\ e_{[D-1+\frac{1}{N}]}^N = \left(1+\frac{1}{D-1+\frac{1}{N}}\right)^{N(D-1+\frac{1}{N})} < \left(1+\frac{1}{D-1}\right)^{N(D-1+\frac{1}{N})} \\ = \left(1+\frac{1}{D-1}\right)^{(D-1)N}\left(1+\frac{1}{D-1}\right) = e_{[D-1]}^N\left(1+\frac{1}{D-1}\right) \leq 2e_{[D-1]}^N, \end{cases} \tag{C.28}$$

$$\leq \mathcal{O}\left(\frac{1}{\sqrt{N}}D^{3N}e_{[D-1]}^{3N}\right). \tag{C.29}$$

Appendix D. Comparison with other polynomial-solving methods. This appendix compares MINI against methods from two fundamentally different paradigms: Gröbner-based solvers, represented by `msolve` [6], and homotopy continuation, represented by `HomotopyContinuation.jl` [8]. To compare these methods, several choices and assumptions have to be made, which are detailed in Appendix D.1. Afterwards, three experiments from section 4 are revisited: computation time for square systems (subsection 4.1.1), computation time for overdetermined systems with $S = 2N$ (subsection 4.1.2), and accuracy under noise (subsection 4.2.1). Accuracy on noiseless systems is not assessed, as both `msolve` and `HomotopyContinuation.jl` are by design meant to be exact in that setting, the former through symbolic computation, the latter through iterative refinement. As the methods are implemented in different programming languages (MATLAB, Julia, Python, C), comparisons should be interpreted with care, especially for small problem sizes, where overhead may dominate the computation time.

D.1. Setup of `msolve` and `HomotopyContinuation.jl`. Neither method was originally intended to solve noisy overdetermined polynomial equations, so several adaptations were required to include them in the comparisons of section 4.

Coefficient representation. Since `msolve` only accepts integer or rational coefficients, floating-point inputs must first be converted. The rounding procedure used in this section is explained in subsection D.1.1.

Exact roots for overdetermined systems. `msolve` finds only roots that satisfy the system exactly. For square systems this is not an issue, as a square system generically has exact roots. For overdetermined systems, however, this requires coefficients to be represented exactly, since any finite-precision perturbation generically destroys the common roots. In Appendix D.3, exact integer systems are generated for `msolve`, as `msolve` could not solve the initially generated floating-point systems.

Squaring up overdetermined systems. `HomotopyContinuation.jl` handles overdetermined systems by premultiplying the coefficient matrix with a random $(N \times S)$ matrix, solving the resulting square system, and retaining only roots that satisfy the original equations. The same squaring-up strategy could be applied to `msolve` to bypass the requirement for exact common roots, but `msolve` is significantly slower than `HomotopyContinuation.jl` on squared-up systems, so this was not included.

Timing quirk. To eliminate compilation overhead, one warm-up run was performed for each problem size before recording the timings. Even so, for overdetermined systems, the first timed run of `HomotopyContinuation.jl` on a given system (after warm-up with a system of the same size) remained slower than subsequent runs on the same system. Timings are therefore reported for both a first-run (after warmup) and a second-run: the former is practically relevant when using `HomotopyContinuation.jl`, while the latter serves as a lower bound on what a dedicated reimplementation might achieve.

D.1.1. Integer rounding for `msolve`. Unlike the methods compared in section 4, `msolve` represents coefficients as integers or rationals rather than floating-point numbers. To enable a comparison, each floating-point coefficient vector $\mathbf{c}$ is approximated by a vector of B-bit signed integers via

$$\mathbf{c}_{B\,\text{bits}} = \text{round}\left((2^{B-1} - 1) \frac{\mathbf{c}}{\max(\text{abs}(\mathbf{c}))} \right), \tag{D.1}$$

which scales the vector so that the largest magnitude equals $2^{B-1}-1$ and then rounds.

Figure 11 shows the effect of this rounding in the experiment of subsection 4.2.1: four and eight bits yield low accuracy and frequently the wrong number of solutions, while sixteen bits or more give quite accurate results for the tested systems, be it with a loss of numerically accurate digits.

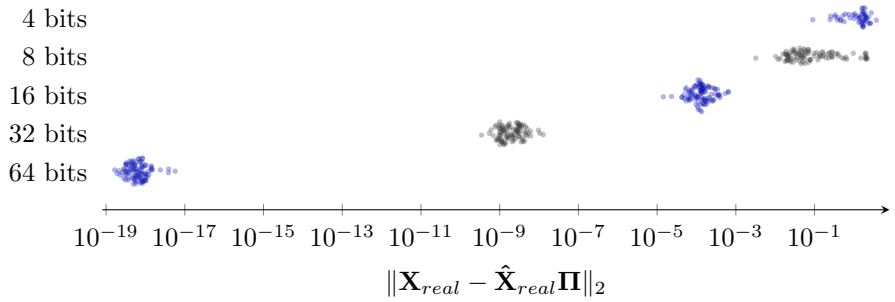


Fig. 11: Accuracy of the real-valued[7] roots returned by `msolve` after rounding floating-point coefficients to B-bit signed integers via (D.1). Each dot represents the error of one random degree-3 system ($N = 5$), vertically offset to show distribution. The number of real roots was incorrect in 83 out of 100 cases for 4-bit rounding and 30 out of 100 for 8-bit rounding. If `msolve` returned fewer real-valued roots than expected, only those were used; if it returned more, the closest ones were selected.

D.2. Computation time for square systems. We revisit the degree-two timing experiment from subsection 4.1.1, adding `msolve` and `HomotopyContinuation.jl` to the comparison. Figure 12 presents the timing results for the full comparison.

D.3. Computation time for overdetermined systems. We revisit the experiment with $S = 2N$ from subsection 4.1.2, adding `HomotopyContinuation.jl` and `msolve`. Figure 13 presents the results.

D.4. Accuracy under noise. The squaring-up step applied in Appendix D.3 by `HomotopyContinuation.jl` discards the redundancy of overdetermined systems, which is precisely what Macaulay-based methods exploit to mitigate noise in a least-squares fashion. To assess the effect of this difference, we repeat the noisy overdetermined accuracy experiment from subsection 4.2.3 ($D = 3$, $S = 10$, $N = 5$, $L = 5$), adding `HomotopyContinuation.jl`[8]. Results are shown in Figure 14.

D.5. Conclusions. The experiments confirm the picture sketched in subsection 1.1: for square systems, `HomotopyContinuation.jl` is faster, with a lower observed complexity. For overdetermined systems (with $S = 2N$), MINI outperforms `HomotopyContinuation.jl` in speed by exploiting the additional equations directly, rather than squaring up. In noisy settings, this least-squares treatment of redundant equations also yields more accurate roots than the squaring-up approach. `msolve` requires exact integer coefficients, which is inconvenient for the floating-point overdetermined setting.

[7] Note that by default `msolve` only returns real-valued roots; hence, only those were used in the comparison.

[8] `HomotopyContinuation.jl`'s built-in root certification rejected all roots of the tested noisy systems, so certification was done manually.

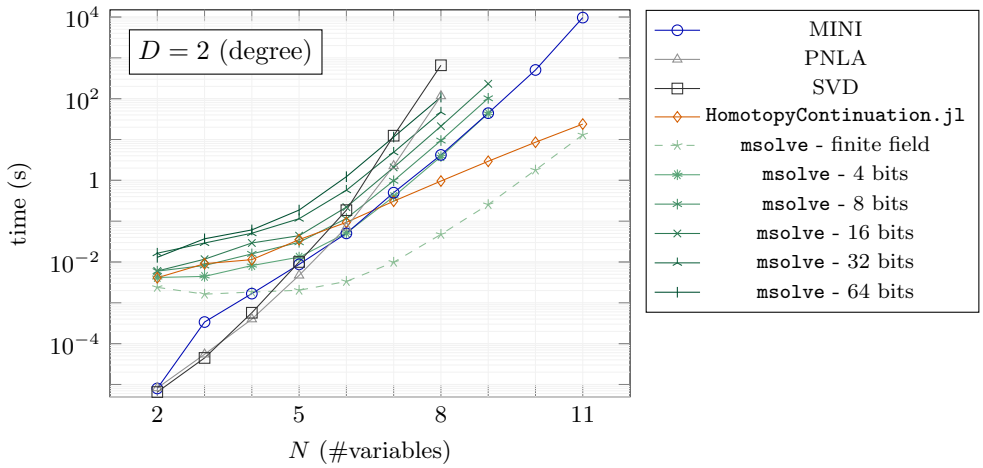


Fig. 12: `HomotopyContinuation.jl` is asymptotically faster than MINI for quadratic square systems. MINI has a similar complexity to 4-bit `msolve`, although 4 bits is not sufficient to find the roots accurately (as suggested by the results in Figure 11). Doubling the number of bits for `msolve` roughly doubles the required time. For reference, working over a finite field (specifically, $\mathrm{GF}(2^{30}+3)$) was added, but this solves an entirely different problem.

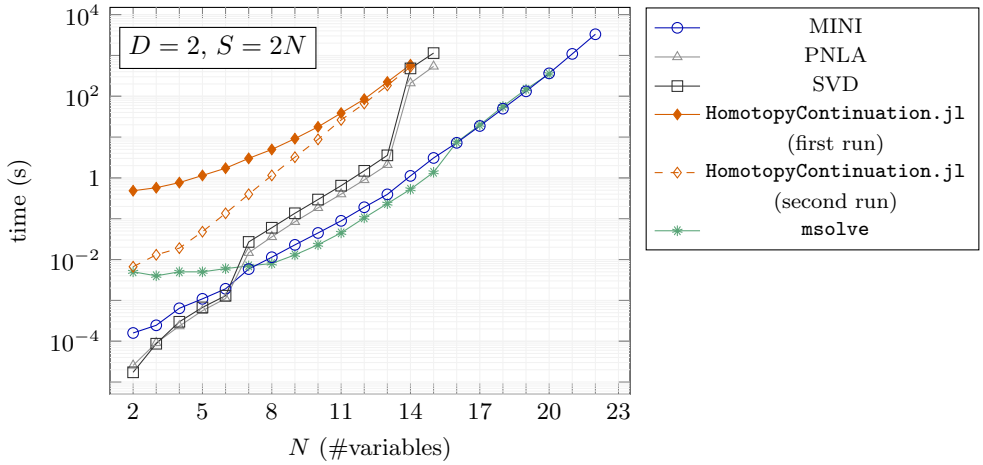


Fig. 13: MINI is faster than `HomotopyContinuation.jl` and handles overdetermined systems with approximate roots, unlike `msolve`. `HomotopyContinuation.jl` is approximately three orders of magnitude slower, as it does not exploit the overdeterminedness of the system and instead solves a squared-up system. `msolve` requires exact roots (see Appendix D.1), which overdetermined floating-point systems generically do not have. It was therefore run on separate integer systems and included for reference only. Note that these integer systems can be solved by any method, so their timings form a meaningful comparison in their own right.

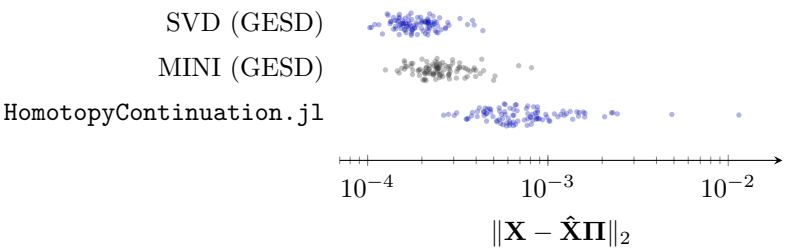


Fig. 14: Squaring up an overdetermined noisy system (as performed by `HomotopyContinuation.jl`) discards redundant equations and can significantly degrade root accuracy. Macaulay-based methods instead incorporate all equations in a least-squares fashion, leading to more accurate estimates.

REFERENCES

[1] M. Bardet, J.-C. Faugère, and B. Salvy, *Asymptotic behaviour of the index of regularity of semi-regular quadratic polynomial systems*, in MEGA 2005 - 8th International Symposium on Effective Methods in Algebraic Geometry, Porto Conte, Alghero, Sardinia, Italy, May 2005, pp. 1–17.

[2] K. Batselier, *A Numerical Linear Algebra Framework for Solving Problems with Multivariate Polynomials*, PhD thesis, KU Leuven, Leuven, Belgium, 2013.

[3] K. Batselier, P. Dreesen, and B. De Moor, *A fast recursive orthogonalization scheme for the Macaulay matrix*, J. Comput. Appl. Math., 267 (2014), pp. 20–32.

[4] M. R. Bender and S. Telen, *Toric eigenvalue methods for solving sparse polynomial systems*, Math. Comput., 91 (2022), pp. 2397–2429.

[5] M. R. Bender and S. Telen, *Yet another eigenvalue algorithm for solving polynomial systems*, 2022. arXiv:2105.08472.

[6] J. Berthomieu, C. Eder, and M. Safey El Din, *msolve: A library for solving polynomial systems*, 46th International Symposium on Symbolic and Algebraic Computation, Saint Petersburg, Russia, July 2021, ACM, pp. 51–58.

[7] É. Bézout, *Théorie générale des équations algébriques*, Imprimeries Ph.-D. Pierres (Paris), 1779.

[8] P. Breiding and S. Timme, *HomotopyContinuation.jl: A package for homotopy continuation in Julia*, in International Congress on Mathematical Software, Springer, 2018, pp. 458–465.

[9] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, PhD thesis, University of Innsbruck, Innsbruck, Austria, 1965.

[10] M. Burr, F. Sottile, and E. Walker, *Numerical homotopies from Khovanskii bases*, Mathematics of computation, 92 (2023), pp. 2333–2353.

[11] A. Cayley, *On the theory of elimination*, Cambridge Dublin Math. J. 3, (1848), pp. 116–120.

[12] T. Chen and T.-Y. Li, *Homotopy continuation method for solving systems of nonlinear and polynomial equations*, Commun. Inf. Syst., 15 (2015), pp. 119–307.

[13] D. A. Cox, J. B. Little, and D. O'Shea, *Using Algebraic Geometry*, vol. 185 of Graduate Texts in Mathematics, Springer, New York, NY, USA, 1998.

[14] D. A. Cox, J. B. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, New York, NY, USA, 2013.

[15] A. L. Dixon, *The eliminant of three quantics in two independent variables*, Proc. London Math. Soc, (1909), pp. 49–69.

[16] T. Duff, N. Hein, and F. Sottile, *Certification for polynomial systems via square subsystems*, J. Symb. Comput., 109 (2022), pp. 367–385.

[17] I. Z. Emiris, *Sparse elimination and applications in kinematics*, PhD thesis, University of California, Berkeley, Berkeley, CA, 1994.

[18] E. Evert, M. Vandecappelle, and L. De Lathauwer, *A recursive eigenspace computation for the canonical polyadic decomposition*, SIAM J. Matrix Anal. Appl., 43 (2022), pp. 274–

300.
[19] J.-C. Faugère, *A new efficient algorithm for computing Gröbner bases (F4)*, Journal of pure and applied algebra, 139 (1999), pp. 61–88.
[20] J.-C. Faugère, *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)*, in Proc. of the 2002 international symposium on Symbolic and algebraic computation, 2002, pp. 75–83.
[21] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, 4 ed., 2012.
[22] N. Govindarajan, R. Widdershoven, S. Chandrasekaran, and L. De Lathauwer, *A fast algorithm for computing macaulay null spaces of bivariate polynomial systems*, SIAM J. Matrix Anal. Appl., 45 (2024), pp. 368–396.
[23] E. Graf and A. Townsend, *Numerical instability of algebraic rootfinding methods*, (2024). arXiv:2408.02805.
[24] R. Hartshorne, *Algebraic Geometry*, vol. 52, Springer, New York, NY, USA, 2013.
[25] A. Kipnis and A. Shamir, *Cryptanalysis of the HFE public key cryptosystem by relinearization*, in Advances in Cryptology — CRYPTO' 99, Santa Barbara, California, USA, 1999, Springer, pp. 19–30.
[26] D. Lazard, *Résolution des systèmes d'équations algébriques*, Theoret. Comput. Sci., 15 (1981), pp. 77–110.
[27] T. Li, *Numerical solution of polynomial systems by homotopy continuation methods*, Handbook of Numerical Analysis, 11 (2003), pp. 209–304.
[28] F. S. Macaulay, *Some formulae in elimination*, Proc. London Math. Soc, 1 (1902), pp. 3–27.
[29] V. Noferini and A. Townsend, *Numerical instability of resultant methods for multidimensional rootfinding*, SIAM Journal on Numerical Analysis, 54 (2016), pp. 719–743.
[30] L. Pachter and B. Sturmfels, *Algebraic statistics for computational biology*, vol. 13, Cambridge university press, 2005.
[31] J. M. Selig, *Geometric fundamentals of robotics*, in Monographs in Computer Science, 2004.
[32] A. Sommese and C. Wampler, *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science*, Information and Interdisciplinary Subjects Series, World Scientific, 2005.
[33] L. Sorber, M. Van Barel, and L. De Lathauwer, *Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization*, SIAM J. Optim., 23 (2013), pp. 695–720.
[34] H. J. Stetter, *Numerical polynomial algebra*, SIAM, Philadelphia, PA, USA, 2004.
[35] J. J. Sylvester, *On a theory of the syzygetic relations of two rational integral functions, comprising an application to the theory of Sturm's functions, and that of the greatest algebraical common measure*, Phil. Trans. R. Soc., 143 (1853), pp. 407–548.
[36] J. Vanderstukken and L. De Lathauwer, *Systems of polynomial equations, higher-order tensor decompositions and multidimensional harmonic retrieval: A unifying framework. Part I: The canonical polyadic decomposition*, SIAM J. Matrix Anal. Appl., 42 (2021), pp. 883–912.
[37] J. Vanderstukken, P. Kürschner, I. Domanov, and L. De Lathauwer, *Systems of polynomial equations, higher-order tensor decompositions, and multidimensional harmonic retrieval: A unifying framework. Part II: The block term decomposition*, SIAM J. Matrix Anal. Appl., 42 (2021), pp. 913–953.
[38] L. Vanpoucke, S. Lagauw, and B. De Moor, *Exploiting the Macaulay matrix structure for the globally optimal identification of autonomous systems*, in European Research Network System Identification 2024 (ERNSI 2024), Venice, Italy, Sept. 2024, https://ftp.esat.kuleuven.be/pub/stadius//lvanpouc/ERNSI2024.pdf. Poster.
[39] C. Vermeersch and B. De Moor, *Recursive algorithms to update a numerical basis matrix of the null space of the block row, (banded) block toeplitz, and block macaulay matrix*, SIAM J. Sci. Comput., 45 (2023), pp. A596–A620.
[40] C. Vermeersch and B. L. R. de Moor, *A column space based approach to solve systems of multivariate polynomial equations*, 54 (2021), pp. 137–144.
[41] R. Widdershoven, N. Govindarajan, and L. De Lathauwer, *Overdetermined systems of polynomial equations: Tensor-based solution and application*, in Proc. of the 31st European Signal Processing Conference (EUSIPCO), Helsinki, Finland, September 2023, pp. 650–654.