Towards Efficient Algorithms for Approximately Solving (Overdetermined) Systems of Polynomial Equations

Nithin Govindarajan* Raphaël Widdershoven* Lieven De Lathauwer*

* KU Leuven

General outline

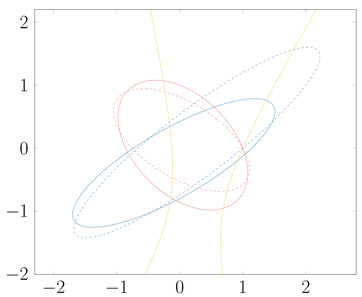
The problem of solving multivariate polynomial equations is a generalization of linear systems and univariate root finding. While classical work mostly focused on the (noiseless) square case, our goal is to solve noisy overdetermined systems that may only admit approximate roots. We introduce a method that reduces the root retrieval to a tensor decomposition problem. Central to our method is the construction of the null space of the Macaulay matrix. However, a key challenge is the rapid increase in the size of this matrix. To address this, our work explores the matrix's underlying structure to significantly speed up the computation of its null space.

I. The problem

Noisy (overdetermined) polynomial systems

Solve a system of S multivariate polynomial equations in $N \leqslant S$ variables:

$$\begin{aligned} -3-\ x-2y+4x^2+6xy+7y^2 &= 0 \\ -2-\ x+\ y+3x^2-7xy+5y^2 &= 0 \\ 1+7x+\ y-8x^2+3xy+\ y^2 &= 0 \end{aligned}$$



Adding noise to the equations *destroys* the single exact root at $(1, 0)$

Practical example: Multi-source localization using Friis equations, i.e., given *noisy* measured quantities P_s^r, P_1^t, P_2^t for $s = 1, \ldots, S$, determine receiver locations (x_1^t, y_1^t) and (x_2^t, y_2^t) from

$$P_s^r = \frac{A_s^r A_1^t}{\lambda^2} \frac{P_1^t}{(x_s^r - x_1^t)^2 + (y_s^r - y_1^t)^2} + \frac{A_s^r A_2^t}{\lambda^2} \frac{P_2^t}{(x_s^r - x_2^t)^2 + (y_s^r - y_2^t)^2}$$

If $S \geqslant 5$, positions can be *retrieved* up to permutation ambiguity!

The Macaulay matrix & its null space

Given a polynomial system $\Sigma : \{f_s\}_{s=1}^S$, the Macaulay matrix M is defined as the *transpose* of the linear map describing the polynomial multiplication

$$\mathscr{M} : \quad (h_1, \ldots, h_S) \mapsto \sum_{s=1}^{S} h_s f_s.$$

For our example, the Macaulay matrix at degree $d = 3$ equals

$$\begin{array}{c} p_1(x,y) \\ p_2(x,y) \\ p_3(x,y) \\ xp_1(x,y) \\ xp_2(x,y) \\ xp_3(x,y) \\ yp_1(x,y) \\ yp_2(x,y) \\ yp_3(x,y) \end{array} \left[\begin{array}{cc|cc|cc|ccc} -3 & -1 & -2 & 4 & 6 & 7 & 0 & 0 & 0 \\ -2 & -1 & 1 & 3 & -7 & 5 & 0 & 0 & 0 \\ 1 & 7 & 1 & -8 & 3 & 1 & 0 & 0 & 0 \\ \hline 0 & -3 & 0 & -1 & -2 & 0 & 4 & 6 & 7 \\ 0 & -2 & 0 & -1 & 1 & 0 & 3 & -7 & 5 \\ 0 & 1 & 0 & 7 & 1 & 0 & -8 & 3 & 1 \\ \hline 0 & 0 & -3 & 0 & -1 & -2 & 0 & 4 & 6 \\ 0 & 0 & -2 & 0 & -1 & 1 & 0 & 3 & -7 \\ 0 & 0 & 1 & 0 & 7 & 1 & 0 & -8 & 3 \end{array}\right] \left[\begin{array}{c} 1 \\ \hline x \\ y \\ \hline x^2 \\ xy \\ y^2 \\ \hline x^3 \\ x^2y \\ xy^2 \\ y^3 \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 0 \end{array}\right]$$

If d is large enough: $\dim \operatorname{null} \mathrm{M}(d) =$ no. of projective roots.

Roots are recovered from null space by performing an *unmixing* operation: involves computing a *canonical polyadic decomposition* of a tensor [2]

II. Fast null space algorithms

Main computational Problem: Macaulay matrix grows *quickly* in size!

A low displacement rank (LDR) approach

The dimensions of the bivariate Macaulay matrix $\mathrm{M}(d)$ grow *quadratically* w.r.t. d, but its rank under the displacement operator

$$\mathscr{D} : \quad \mathrm{X} \mapsto \operatorname{diag}\left\{\mathrm{Z}_{i,1} \otimes \mathrm{I}_S\right\}_{i=\Delta d+1}^{1} \mathrm{X} - \mathrm{X} \operatorname{diag}\left\{\mathrm{Z}_{j,\varphi_j}\right\}_{j=d+1}^{1}$$

under grows only *linearly* with d.

Use the GKO algorithm [1] to find the null space:

1. Apply unitary transformations Φ and Ψ such that $\Phi \mathrm{M}(d) \Psi =: \hat{\mathrm{M}}(d)$ is Cauchy-like, i.e., its entries are of the form

$$\left[\hat{\mathrm{M}}(d)\right]_{ij} := [\Phi \mathrm{M}(d) \Psi]_{ij} = \frac{\boldsymbol{u}_i^* \boldsymbol{v}_j}{\mu_i - \nu_j}, \qquad \boldsymbol{u}_i, \boldsymbol{v}_j \in \mathbb{C}^{S(\Delta d+1)}.$$

2. Let $r(d) := \operatorname{rank} \mathrm{M}(d)$. Compute a rank-revealing LU (RRLU) factorization

$$\begin{aligned} \Pi_1 \widehat{\mathrm{M}}(d) \Pi_2 &= \begin{bmatrix} \mathrm{I}_{r(d)} \\ \widehat{\mathrm{M}}_{21} \widehat{\mathrm{M}}_{11}^{-1} & \mathrm{I}_{d_\Sigma^2} \end{bmatrix} \begin{bmatrix} \widehat{\mathrm{M}}_{11} & \\ & \widehat{\mathrm{M}}_{22} - \widehat{\mathrm{M}}_{21} \widehat{\mathrm{M}}_{11}^{-1} \widehat{\mathrm{M}}_{12} \end{bmatrix} \begin{bmatrix} \mathrm{I}_{r(d)} & \widehat{\mathrm{M}}_{11}^{-1} \widehat{\mathrm{M}}_{12} \\ & \mathrm{I}_{d_\Sigma^2} \end{bmatrix} \\ &\approx \begin{bmatrix} \widehat{\mathrm{M}}_{11} \\ \widehat{\mathrm{M}}_{21} \end{bmatrix} \begin{bmatrix} \mathrm{I}_{r(d)} & \widehat{\mathrm{M}}_{11}^{-1} \widehat{\mathrm{M}}_{12} \end{bmatrix} \end{aligned}$$

A method relying on intersection of shifted null spaces (MINI algorithm)

Take advantage of the nested recursive structure:

$$\operatorname{null} \left[\begin{array}{c|c|c|c} 1 & 2\ 3 & 1\ 5\ 6 & 0\ 0\ 0\ 0 \\ 2 & 1\ 7 & 8\ 1\ 9 & 0\ 0\ 0\ 0 \\ \hline 0 & 1\ 0 & 2\ 3\ 0 & 1\ 5\ 6\ 0 \\ 0 & 2\ 0 & 1\ 7\ 0 & 8\ 1\ 9\ 0 \\ 0 & 0\ 1 & 0\ 2\ 3 & 0\ 1\ 5\ 6 \\ 0 & 0\ 2 & 0\ 1\ 7 & 0\ 8\ 1\ 9 \end{array}\right] = \operatorname{null} \left[\begin{array}{c|c|c|c} 1 & 2\ 3 & 1\ 5\ 6 & 0\ 0\ 0\ 0 \\ 2 & 1\ 7 & 8\ 1\ 9 & 0\ 0\ 0\ 0 \end{array}\right] \cap \operatorname{null} \left[\begin{array}{c|c|c|c} 0 & 1\ 0 & 2\ 3\ 0 & 1\ 5\ 6\ 0 \\ 0 & 2\ 0 & 1\ 7\ 0 & 8\ 1\ 9\ 0 \end{array}\right] \cap \operatorname{null} \left[\begin{array}{c|c|c|c} 0 & 0\ 1 & 0\ 2\ 3 & 0\ 1\ 5\ 6 \\ 0 & 0\ 2 & 0\ 1\ 7 & 0\ 8\ 1\ 9 \end{array}\right]$$

$$\Downarrow$$

$$\operatorname{null} \left[\begin{array}{c|c|c|c} 1 & 2\ 3 & 1\ 5\ 6 & 0\ 0\ 0\ 0 \\ 2 & 1\ 7 & 8\ 1\ 9 & 0\ 0\ 0\ 0 \\ \hline 0 & 1\ 0 & 2\ 3\ 0 & 1\ 5\ 6\ 0 \\ 0 & 2\ 0 & 1\ 7\ 0 & 8\ 1\ 9\ 0 \\ 0 & 0\ 1 & 0\ 2\ 3 & 0\ 1\ 5\ 6 \\ 0 & 0\ 2 & 0\ 1\ 7 & 0\ 8\ 1\ 9 \\ \hline 0 & 0\ 0 & 1\ 0\ 0 & 2\ 3\ 0 \\ 0 & 0\ 0 & 2\ 0\ 0 & 1\ 7\ 0 \\ 0 & 0\ 0 & 0\ 1\ 0 & 0\ 2\ 3 \\ 0 & 0\ 0 & 0\ 2\ 0 & 0\ 1\ 7 \end{array}\right] = \operatorname{null} \left[\begin{array}{c|c|c|c} 1 & 2\ 3 & 1\ 5\ 6 & 0\ 0\ 0\ 0 \\ 2 & 1\ 7 & 8\ 1\ 9 & 0\ 0\ 0\ 0 \\ \hline 0 & 1\ 0 & 2\ 3\ 0 & 1\ 5\ 6\ 0 \\ 0 & 2\ 0 & 1\ 7\ 0 & 8\ 1\ 9\ 0 \\ 0 & 0\ 1 & 0\ 2\ 3 & 0\ 1\ 5\ 6 \\ 0 & 0\ 2 & 0\ 1\ 7 & 0\ 8\ 1\ 9 \end{array}\right] \cap \operatorname{null} \left[\begin{array}{c|c|c|c} 0 & 1\ 0 & 2\ 3\ 0 & 1\ 5\ 6\ 0 \\ 0 & 2\ 0 & 1\ 7\ 0 & 8\ 1\ 9\ 0 \\ \hline 0 & 0\ 0 & 1\ 0\ 0 & 2\ 3\ 0 \\ 0 & 0\ 0 & 2\ 0\ 0 & 1\ 7\ 0 \\ 0 & 0\ 0 & 0\ 1\ 0 & 0\ 2\ 3 \\ 0 & 0\ 0 & 0\ 2\ 0 & 0\ 1\ 7 \end{array}\right] \cap \operatorname{null} \left[\begin{array}{c|c|c|c} 0 & 0\ 1 & 0\ 2\ 3 & 0\ 1\ 5\ 6 \\ 0 & 0\ 2 & 0\ 1\ 7 & 0\ 8\ 1\ 9 \\ \hline 0 & 0\ 0 & 0\ 1\ 0 & 0\ 2\ 3 \\ 0 & 0\ 0 & 0\ 2\ 0 & 0\ 1\ 7 \end{array}\right]$$

Shifted null spaces and subspace intersections can be computed *fast* from previous steps [3].

Krylov subspace methods

Main idea:
Coupled with fast mat-vecs, use a Krylov method like CG or *implicitly restarted Lanczos bidiagonalization* to find the smallest right singular vectors.

For example, the Macaulay matrix at degree 7 for the Chebyshev polynomial

$$p(x) = \frac{c_0}{2} + \sum_{k=1}^{4} c_k T_k(x) \tag{1}$$

is the first 4 rows of the matrix

$$\mathrm{W}(\boldsymbol{a}) = \frac{1}{2} \underbrace{\begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & 0 & 0 & 0 \\ c_1 & c_0 & c_1 & c_2 & c_3 & c_4 & 0 & 0 \\ c_2 & c_1 & c_0 & c_1 & c_2 & c_3 & c_4 & 0 \\ c_3 & c_2 & c_1 & c_0 & c_1 & c_2 & c_3 & c_4 \\ c_4 & c_3 & c_2 & c_1 & c_0 & c_1 & c_2 & c_3 \\ 0 & c_4 & c_3 & c_2 & c_1 & c_0 & c_1 & c_2 \\ 0 & 0 & c_4 & c_3 & c_2 & c_1 & c_0 & c_1 \\ 0 & 0 & 0 & c_4 & c_3 & c_2 & c_1 & c_0 \end{bmatrix}}_{\mathrm{SymToep}(\boldsymbol{c})} + \frac{1}{2} \underbrace{\begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & 0 & 0 & 0 \\ c_1 & c_2 & c_3 & c_4 & 0 & 0 & 0 & 0 \\ c_2 & c_3 & c_4 & 0 & 0 & 0 & 0 & 0 \\ c_3 & c_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -c_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & -c_4 & -c_3 \\ 0 & 0 & 0 & 0 & 0 & -c_4 & -c_3 & -c_2 \end{bmatrix}}_{\mathrm{Hank}(\boldsymbol{c})}$$

With $\boldsymbol{a} = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & 0 & 0 & 0 \end{bmatrix}$, the mat-vecs can be accelerated with *fast cosine transforms* due to the decomposition

$$\mathrm{W}(\boldsymbol{a}) = \mathrm{D}_n \mathrm{C}_n^\top \operatorname{diag}\left(\mathrm{C}_n^\top \mathrm{S}_n \boldsymbol{a}\right) \mathrm{C}_n^\top \mathrm{S}_n^{-1} \tag{2}$$

Comparison

Method	Benefit	Drawback
LDR	Memory efficient; reduces complexity from $O(d^6)$ to $O(d^5)$ for bivariate systems	Does not generalize nicely to many variable systems: diminishing returns!
MINI	Exponential improvement for many variables: from $O(2^{6N})$ to $O(2^{4N})$ for square degree two systems	Bad with handling noise; intermediate null spaces can get too large for overdetermined systems with few roots
Krylov	Beneficial if number of roots R is small: complexity goes $O(d^{3N})$ to $O(RNd^{2N} \log d)$.	Possible slow convergence due to ill-conditioning of the matrix.

Open problem: preconditioners for Krylov method

Find a $\mathrm{P} \approx (\mathrm{M}^\top \mathrm{M})^\dagger$, with M being a multi-level block-toeplitz matrix.

References

[1] 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.

[2] 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 Journal On Matrix Analysis And Applications, 42 (2021), pp. 883–912.

[3] R. Widdershoven, N. Govindarajan, and L. De Lathauwer, *Fast macaulay null space through the intersection of shifted null spaces*, Technical Report 25-56, ESAT-STADIUS, KU Leuven (In preparation), (2025).

nithin.govindarajan@kuleuven.be