Solving Multivariate Polynomial Systems and Rectangular Multiparameter Eigenvalue Problems with MacaulayLab

CHRISTOF VERMEERSCH and BART DE MOOR, Center for Dynamical Systems, Signal Processing, and Data Analytics (STADIUS), Department of Electrical Engineering (ESAT), KU Leuven, Belgium

We present the MATLAB toolbox MacaulayLab, which provides numerical linear algebra algorithms for solving multivariate polynomial systems and rectangular multiparameter eigenvalue problems. In this paper, we introduce the capabilities and structure of the toolbox. We demonstrate the software on some illustrative examples and compare its performance with other software packages, such as PHCpack and MultiParEig. Some core features of MacaulayLab are the fact that it (i) solves two key problems via a common approach, (ii) works independently of the chosen polynomial basis and monomial ordering, and (iii) is capable of dealing with positive-dimensional solution sets at infinity. The toolbox and a large collection of test problems are freely available at www.macaulaylab.net.

CCS Concepts: • **Mathematics of Computing → Solvers**; **Computations on Polynomials**; **Computations on Matrices**.

Additional Key Words and Phrases: Multivariate Polynomial Systems, Multiparameter Eigenvalue Problems, (Block) Macaulay Matrices.

ACM Reference Format:
Christof Vermeersch and Bart De Moor. 2025. Solving Multivariate Polynomial Systems and Rectangular Multiparameter Eigenvalue Problems with MacaulayLab. *ACM Trans. Math. Softw.* 00, 0, Article 000 (2025), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

MacaulayLab is a MATLAB toolbox that uses numerical linear algebra techniques to solve two challenging, but important, problems—it tackles systems of multivariate polynomial equations and rectangular multiparameter eigenvalue problems. On the one hand, finding the common roots of multivariate polynomial systems is a well-known problem in algebraic geometry and computational nonlinear algebra. Because of the omnipresence of multivariate polynomials in scientific models and engineering applications, there have been a lot of effort in developing solvers for multivariate polynomial systems. While iterative root-finding methods, such as the efficient homotopy continuation methods, track solution paths, algebraic root-finding methods transform the system into (multiple) univariate root-finding or eigenvalue problems. The algorithms implemented in MacaulayLab take the latter approach to compute the common roots and produce one or multiple generalized eigenvalue problems. Rectangular multiparameter eigenvalue problems, on the other hand, are less known for the general audience, but also appear in various modeling efforts. MacaulayLab is one of the first toolboxes focussing on rectangular multiparameter eigenvalue problems. Similar

Authors' Contact Information: Christof Vermeersch, christof.vermeersch@esat.kuleuven.be; Bart De Moor, bart.demoor@esat.kuleuven.be, Center for Dynamical Systems, Signal Processing, and Data Analytics (STADIUS), Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, 3000 Leuven, Belgium.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-7295/2025/0-ART000
https://doi.org/XXXXXXX.XXXXXXX

as for the polynomials, the algorithms in MacaulayLab rephrase the rectangular multiparameter eigenvalue problems as one or multiple generalized eigenvalue problems. Very recently, the authors of [20] have shown that it is also possible to translate (linear) rectangular problems into a more established square formulation, enabling a new set of algorithms to be used.

While these two key problems may seem unrelated at first glance, multivariate polynomial systems and multiparameter eigenvalue problems are both special cases of the same, broader problem and are connected through the (block) Macaulay matrix. Section 2 gives a rigorous definition of both key problems and reveals their relation. In that sense, MacaulayLab is quite unique; the combined nature of the two key problems allows for one tool(box) to tackle them both.

MacaulayLab uses the (block) Macaulay matrix to solve multivariate polynomial systems and rectangular multiparameter eigenvalue problems via a similar numerical linear algebra methodology. As explained in Section 3, it transforms both key problems into a multidimensional realization problem by using the column space or (right) null space of the (block) Macaulay matrix constructed from the coefficients of the polynomials or the coefficient matrices of the rectangular multiparameter eigenvalue problem. The multidimensional realization problem corresponds to one or multiple generalized eigenvalue problems that yield the solutions of the original problem. We explain in Section 4 how this transformation is obtained by only using numerical linear algebra tools, such as singular value, QR, or eigenvalue decompositions. The toolbox relies on the decades of advancements in numerical linear algebra, resulting in computationally efficient and numerically robust algorithms. An important feature of MacaulayLab is that all the algorithms are implemented without depending on a particular polynomial basis or monomial ordering, allowing the user to choose what suits the application best (see Section 5). It is known that in some situations an orthogonal polynomial basis, like the Chebyshev polynomials, has superior numerical properties [31, 32], while results in algebraic geometry often depend on the chosen monomial ordering [15]. Offering the user the choice to select a particular polynomial basis and monomial ordering can thus be very useful in applications.

MacaulayLab is written in MATLAB and freely available at www.macaulaylab.net. The toolbox has been developed to be user-friendly and easy to use, whether the user simply wants to solve a problem or wants to learn more about its properties. The website contains, next to the toolbox and its documentation, many examples and highlights of research that uses the toolbox. Furthermore, the authors have gathered an extensive collection of test problems (see Section 6.1) and made it available together with the toolbox. These problems can be used to experiment with MacaulayLab, but also to test other software packages, as illustrated in the comparison of Section 7.

Summary of the paper's outline and objectives. We continue this paper by rigorously defining the key problems that MacaulayLab tackles in Section 2. We explain why we discuss both multivariate polynomial systems and rectangular multiparameter eigenvalue problems and give a summary of the existing solution software. The common (block) Macaulay matrix approach is discussed in Section 3. *The paper's first objective is to discuss the implementation of the toolbox and present its most important functionalities.*[1], which is done in Section 4. Next, we highlight one of the most important features in Section 5: the independence of the polynomial basis and monomial ordering. Section 6 compares some of the solution strategies in MacaulayLab, which is explained by means of the accompanying collection of test problems. *A second objective is to evaluate the software's performance with respect to other state-of-the-art solvers*, which is the topic of Section 7. With that in mind, we discuss the current limitations and sketch the path of future developments in Section 8 and conclude the paper in Section 9.

[1]Notice that the paper is not a guide on how to use MacaulayLab. For more information about using the toolbox, we refer the interested reader to the user manual and introductory MATLAB script (also available online at www.macaulaylab.net).

Hardware specifications and code blocks. All computations in this paper are performed on a MacBook Pro that has an M1 CPU (2020) working at 3.2 GHz (8 cores) and 16 GB RAM. This choice has been made to evaluate the performance and capabilities of the toolbox under typical working conditions, rather than on a server with extensive memory resources. Throughout the text, we have added some code blocks that demonstrate the toolbox by using `MacaulayLab` commands.

Code 1. Let us start by solving a trivariate polynomial system from the test collection.

```
>> system = readproblem("noon3");
>> solutions = macaulaylab(system,verbose = true);
```

The output of `MacaulayLab` is given in Fig. 1. When using the verbose option of `MacaulayLab`, the output shows in real-time the progress of the solver.

2 Two key problems and the available software to solve them

Let $\mathcal{P}_j(\boldsymbol{x})$ be a matrix polynomial in m variables, $\boldsymbol{x} = (x_1, \ldots, x_m)$, with $k \times l$ coefficient matrices $\boldsymbol{A}_j^{(\boldsymbol{i})} \in \mathbb{C}^{k\times l}$ and degree d_j. For example, the degree three matrix polynomial

$$\mathcal{P}_1(\boldsymbol{x}) = \boldsymbol{A}_1^{(13)} x_1 x_2^2 + \boldsymbol{A}_1^{(00)} = \begin{bmatrix} x_1x_2^3 + 1 & 2x_1x_2^3 + 2 \\ 3x_1x_2^3 + 3 & 4x_1x_2^3 + 4 \\ 5x_1x_2^3 + 5 & 6x_1x_2^3 + 6 \end{bmatrix}, \quad \text{forwhich } \boldsymbol{A}_1^{(13)} = \boldsymbol{A}_1^{(00)} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

The subscript indicates to which matrix polynomial the coefficient matrix belongs, while the multi-index $\boldsymbol{i} = (i_1, \ldots, i_n) \in \mathbb{N}^m$ indexes the coefficient matrices and corresponds to the power of the associated monomial $\boldsymbol{x}^{\boldsymbol{i}}$. Consider now the system of s matrix equations in m-variables,

$$\mathcal{P}_1(\boldsymbol{x})\,\boldsymbol{y} = \cdots = \mathcal{P}_s(\boldsymbol{x})\,\boldsymbol{y} = \boldsymbol{0}, \quad \|\boldsymbol{y}\| = 1, \tag{1}$$

in which each matrix polynomial is multiplied by the same nonzero vector $\boldsymbol{y} \in \mathbb{C}_0^{l\times 1}$. `MacaulayLab` can be used to find the solutions $(\boldsymbol{x}, \boldsymbol{y})$ that satisfy these matrix equations. Two well-known key problems can be distilled from (1): the multivariate polynomial root-finding problem (Section 2.1) and the rectangular multiparameter eigenvalue-finding problem (Section 2.2).

2.1 First key problem: multivariate polynomial root-finding

Suppose that the coefficient matrices are scalars, i.e., $k = l = 1$ such that $\boldsymbol{A}_j^{(\boldsymbol{i})} = a_j^{(\boldsymbol{i})} \in \mathbb{C}$. Consequently, $\boldsymbol{y} \in \mathbb{C}^{1\times 1}$ is a scalar with $\|\boldsymbol{y}\| = 1$; it is always equal to one and can be omitted. The s matrix polynomials are multivariate polynomials.

2.1.1 Multivariate polynomial root-finding problem. Multivariate polynomials are powerful tools to model problems from various origins. Finding the common roots of a multivariate polynomial system is can be defined as follows.

Definition 1. Given s polynomials $p_j(\boldsymbol{x})$ with coefficients $a_j^{(\boldsymbol{i})} \in \mathbb{C}$, the *multivariate polynomial root-finding problem* consists in finding all m-tuples $\boldsymbol{x}^* \in \mathbb{C}^m$, so that each $p_j(\boldsymbol{x}^*) = 0$, with $j = 1, \ldots, j$.

Example 1. A system of two bivariate polynomials of degree two looks as following:

$$\begin{cases} p_1(x_1, x_2) = a_1^{(20)} x_1^2 + a_1^{(11)} x_1x_2 + a_1^{(02)} x_2^2 + a_1^{(10)} x_1 + a_1^{(01)} x_2 + a_1^{(00)}, \\ p_2(x_1, x_2) = a_2^{(20)} x_1^2 + a_2^{(11)} x_1x_2 + a_2^{(02)} x_2^2 + a_2^{(10)} x_1 + a_2^{(01)} x_2 + a_2^{(00)}. \end{cases} \tag{2}$$

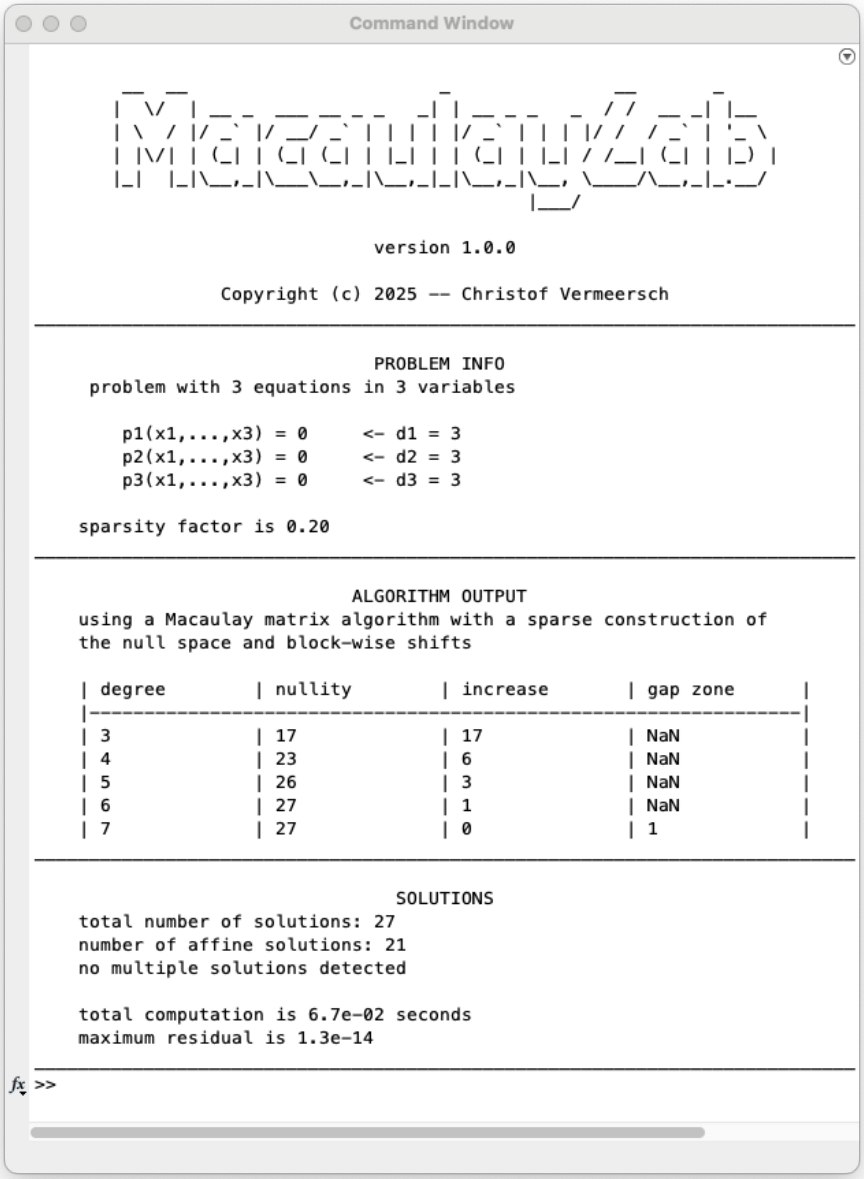


Fig. 1. Output of MacaulayLab when solving the NOON3 system from the collection via the default Macaulay matrix approach. When the verbose option is enabled, the toolbox displays a lot of information about the problem, the solution procedure, and the obtained (affine) solutions.

Because of their modeling capabilities, multivariate polynomial systems have applications in diverse areas of science and engineering, for example, in robotics, game theory, computational chemistry and biology, computer vision , system identification, and model order reduction. Extensive lists of possible applications can be found in reference books like [10, 24, 29].

2.1.2 Existing root-finding software. Currently, the most efficient way of obtaining the common roots of a system of multivariate polynomials with available software is via homotopy continuation algorithms. These algorithms employ a mixture of techniques from algebraic geometry and nonlinear optimization to continuously deform a starting system with known solutions into the original system with unknown solutions, while tracking the paths of the solutions (see, for example, [23, 28, 35]). Although issues with ill-conditioning still exist, homotopy continuation methods are inherently parallel, i.e., each isolated solution can be computed independently, and are currently among the most competitive algorithms to solve systems of multivariate polynomial equations. Their main disadvantage is that they only work for square (i.e., the number of equations is equal to the number of unknown variables) systems of multivariate polynomial equations. Because of their efficiency and applicability, software for homotopy continuation comes in many flavours. Some toolboxes that use homotopy continuation to tackle polynomial systems are `PHCpack` [36], `HOMPACK` [37], `PHoM` [17, 18], `HOM4PS` [8], `NAG4M2` [22] (part of `Macaulay2`), `Bertini` [2], and `HomotopyContinuation.jl` [7].

Another approach to deal with systems of multivariate polynomial equations is to reduce the problem into subsequent univariate root-finding problems or eigenvalue problems. On the one hand, symbolic software packages, like `Maple`, `Singular`, `Magma`, and `msolve`, use a symbolic approach to solve polynomial systems, by creating a Gröbner basis to create a triangular system that can be solved via back-substitution or to construct the multiplication matrices (the eigenvalues of which are related to the solutions of the polynomial system). It is, on the other hand, also possible to rely solely on numerical linear algebra techniques to rephrase polynomial systems as eigenvalue problems. `MacaulayLab` falls into this category of solvers. [3, 14] have also approached the problem via the Macaulay matrix and have implemented their algorithms in the `MATLAB` packages `PNLA` and `RootFinding`[2], respectively. Similar to `MacaulayLab`, they both use the Macaulay matrix to set up one or multiple multidimensional realization problems. While `RootFinding` is a quite naive implementation of the Macaulay matrix approach as described in [14], `PNLA` offers functions that take advantage of the structure and sparsity of the Macaulay matrix. When the system consists of two bivariate polynomials, the `MATLAB` package `BiRoots` by [26] can be used, which transforms the problem into a square multiparameter eigenvalue problem and solves this problem via `MultiParEig` (see Section 2.2). For users of the Julia language, we want to highlight the packages `AlgebraicSolvers.jl` by [30] and `JuliaEigenvalueSolver.jl` by [6], which tackle the polynomial systems by using the Macaulay (or a related) matrix to set up the multiplication matrices.

2.2 Second key problem: rectangular multiparameter eigenvalue-finding

Another problem that can be deduced from (1) is the rectangular multiparameter eigenvalue problem. By setting $s = 1$, there is only one matrix equation and $k \geq l + n - 1$ in order to have a well-determined problem.

2.2.1 Rectangular multiparameter eigenvalue-finding problem. The rectangular multiparameter eigenvalue problem is a generalization of the one-parameter polynomial eigenvalue problem with multiple spectral parameters. It is given by a multivariate polynomial matrix. The solutions of the

[2]The `MATLAB` package `RootFinding` is available upon request with its developer, Philippe Dreesen.

rectangular multiparameter eigenvalue problem, or the eigenvalues of the polynomial matrix, are the points $\boldsymbol{x}^* \in \mathbb{C}^m$ for which the rank of the polynomial matrix drops below the normal rank,

$$\operatorname{rank}\big(\boldsymbol{\mathcal{P}}\,(\boldsymbol{x}^*)\big) < \operatorname{nrank}\big(\boldsymbol{\mathcal{P}}\,(\boldsymbol{x})\big). \tag{3}$$

Every eigenvalue $\boldsymbol{x}^*$ has at least one eigenvector $\boldsymbol{y}^* \in \mathbb{C}_0^{l\times 1}$ associated to it. The rectangular multiparameter eigenvalue problem is given in the following definition.

Definition 2. Given coefficient matrices $\boldsymbol{A}^{(i)} \in \mathbb{C}^{k\times l}$ (with $k \geq l$) that lead to a full normal rank matrix polynomial $\boldsymbol{\mathcal{P}}\,(\boldsymbol{x})$, the *rectangular multiparameter eigenvalue-finding problem* consists in finding all m-tuples $\boldsymbol{x}^* \in \mathbb{C}^m$ and corresponding vectors $\boldsymbol{y}^* \in \mathbb{C}_0^{l\times 1}$, so that $\boldsymbol{\mathcal{P}}\,(\boldsymbol{x}^*)\boldsymbol{y}^* = \boldsymbol{0}$, with $\|\boldsymbol{y}^*\| = 1$.

Example 2. Let the coefficient matrices $\boldsymbol{A}^{(i)}$ be $k\times l$ rectangular matrices. A rectangular, quadratic two-parameter eigenvalue problem is then given by

$$\boldsymbol{\mathcal{P}}(x_1, x_2)\,\boldsymbol{y} = \left(\boldsymbol{A}^{(20)}x_1^2 + \boldsymbol{A}^{(11)}x_1x_2 + \boldsymbol{A}^{(02)}x_2^2 + \boldsymbol{A}^{(10)}x_1 + \boldsymbol{A}^{(01)}x_2 + \boldsymbol{A}^{(00)}\right)\boldsymbol{y} = \boldsymbol{0}, \tag{4}$$

where the solutions (x_1^*, x_2^*) are the eigenvalues and $\boldsymbol{y}^* \neq 0$ with $\|\boldsymbol{y}\| = 1$ are the associated eigenvectors.

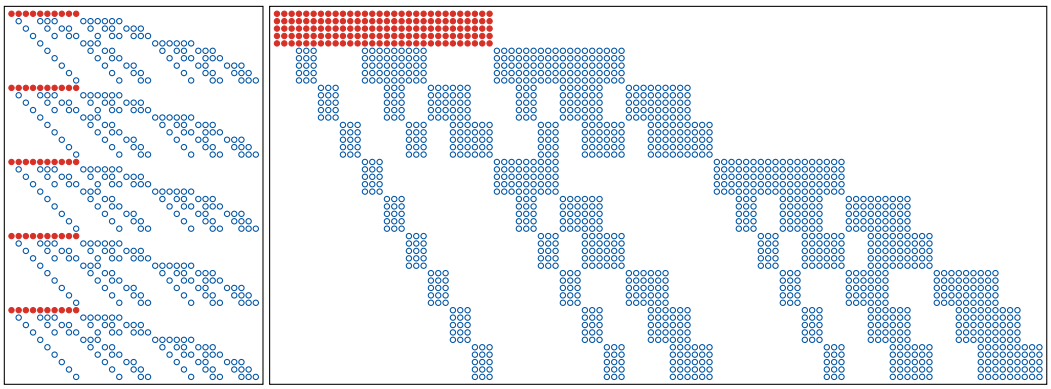
Rectangular multiparameter eigenvalue problems appear when solving certain differential equations, such as the higher-order Heine–Stieltjes problem [27] or when tackling the minimum rank problem in cryptoanalysis [16]. Recently, these problems also emerged as a globally optimal approach to identify the parameters of misfit-latency models [11, 12, 34] and to reduce the order of single-input, single-output models given by their transfer function [1, 21].

2.2.2 Existing eigenvalue-finding software. `MacaulayLab` is one of the first toolboxes focussing on rectangular multiparameter eigenvalue problems. Very recently, [20] have shown that it is possible to translate (linear) rectangular problems into a square formulation. This transformation, together with the available methods for solving square multiparameter eigenvalue problems, is added to `MultiParEig` in one of its last updates[3], adding a second toolbox to list of software packages that can solve rectangular multiparameter eigenvalue problems. This toolbox by Plestenjak [25] features a wide array of algorithms (both direct and iterative) and can deal with singular and non-singular problems. When the problem is not linear, however, an additional linearization step is required (before or after the transformation), while `MacaulayLab` deals with polynomial problems directly.

3 Common (block) Macaulay matrix approach

Independently of the fact whether the matrix equations $\boldsymbol{\mathcal{P}}_j(\boldsymbol{x})\,\boldsymbol{y} = \boldsymbol{0}$ are multivariate polynomials or rectangular matrix polynomials, the (block) Macaulay matrix approach starts with multiplying the matrix equations by different monomials $\boldsymbol{x}^{\boldsymbol{i}}$ up to a certain degree $d - d_j$, resulting in a set of matrix equations $\boldsymbol{x}^{\boldsymbol{i}}\boldsymbol{\mathcal{P}}_j(\boldsymbol{x})\,\boldsymbol{y} = \boldsymbol{0}$ of total degree d or smaller. The coefficients or coefficient matrices

[3]Methods to solve linear rectangular multiparameter eigenvalue problems were added to `MultiParEig` with the release of version 2.7.0.0 (December 6, 2022).



(a) (b)

Fig. 2. Visualization of the (block) Macaulay matrix for a multivariate polynomial system (Fig. 2a) and a rectangular multiparameter eigenvalue problem (Fig. 2b). The (block) Macaulay matrix is constructed from the coefficients of the polynomials or the coefficient matrices of the rectangular multiparameter eigenvalue problem. The original coefficients are indicated by solid red dots (•), while the open blue dots (◦) are the result of the multiplication of the problem with the different monomials.

of these matrix equations are arranged in a structured matrix. For example,

$$
\begin{array}{r}
\\
\mathcal{P}_1\,\boldsymbol{y}\rightarrow \\
x_1\mathcal{P}_1\,\boldsymbol{y}\rightarrow \\
x_2\mathcal{P}_1\,\boldsymbol{y}\rightarrow \\
\mathcal{P}_2\,\boldsymbol{y}\rightarrow \\
x_1\mathcal{P}_2\,\boldsymbol{y}\rightarrow \\
x_2\mathcal{P}_2\,\boldsymbol{y}\rightarrow
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc}
\boldsymbol{y} & x_1\boldsymbol{y} & x_2\boldsymbol{y} & x_1^2\boldsymbol{y} & x_1x_2\boldsymbol{y} & x_2^2\boldsymbol{y} & x_1^3\boldsymbol{y} & x_1^2x_2\boldsymbol{y} & x_1x_2^2\boldsymbol{y} & x_2^3\boldsymbol{y} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow
\end{array} \\
\begin{bmatrix}
\boldsymbol{A}_1^{(00)} & \boldsymbol{A}_1^{(10)} & \boldsymbol{A}_1^{(01)} & \boldsymbol{A}_1^{(20)} & \boldsymbol{A}_1^{(11)} & \boldsymbol{A}_1^{(02)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{A}_1^{(00)} & \boldsymbol{0} & \boldsymbol{A}_1^{(10)} & \boldsymbol{A}_1^{(01)} & \boldsymbol{0} & \boldsymbol{A}_1^{(20)} & \boldsymbol{A}_1^{(11)} & \boldsymbol{A}_1^{(02)} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_1^{(00)} & \boldsymbol{0} & \boldsymbol{A}_1^{(10)} & \boldsymbol{A}_1^{(01)} & \boldsymbol{0} & \boldsymbol{A}_1^{(20)} & \boldsymbol{A}_1^{(11)} & \boldsymbol{A}_1^{(02)} \\
\boldsymbol{A}_2^{(00)} & \boldsymbol{A}_2^{(10)} & \boldsymbol{A}_2^{(01)} & \boldsymbol{A}_2^{(20)} & \boldsymbol{A}_2^{(11)} & \boldsymbol{A}_2^{(02)} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{A}_2^{(00)} & \boldsymbol{0} & \boldsymbol{A}_2^{(10)} & \boldsymbol{A}_2^{(01)} & \boldsymbol{0} & \boldsymbol{A}_2^{(20)} & \boldsymbol{A}_2^{(11)} & \boldsymbol{A}_2^{(02)} & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_2^{(00)} & \boldsymbol{0} & \boldsymbol{A}_2^{(10)} & \boldsymbol{A}_2^{(01)} & \boldsymbol{0} & \boldsymbol{A}_2^{(20)} & \boldsymbol{A}_2^{(11)} & \boldsymbol{A}_2^{(02)}
\end{bmatrix}
\end{array}
\tag{5}
$$

is a (block) Macaulay matrix constructed from two matrix equations, $\boldsymbol{\mathcal{P}}_1(x_1, x_2)$ and $\boldsymbol{\mathcal{P}}_2(x_1, x_2)$, and multiplied by monomials $x_1^{i_1}x_2^{i_2}$, where $d = 2$. The block rows correspond to the different matrix equations, while the block columns are labeled by the different monomials up to degree d. The specific structure of the (block) Macaulay matrix depends on the polynomial basis and monomial ordering that is used. Fig. 2 shows a (block) Macaulay matrix for a multivariate polynomial system and a rectangular multiparameter eigenvalue problem.

The (block) Macaulay matrices form a family of structured matrices parametrized by its degree d. If the degree d of the (block) Macaulay matrix is large enough, then its nullity corresponds to the number of solutions of (1). In that case, the null space has a shift-invariant structure that can be exploited to retrieve these solutions. One possible basis for this null space consists of one (block) multivariate Vandermonde vector for every solution, i.e.,

$$
\boldsymbol{v} = \begin{bmatrix} \boldsymbol{y}^{\mathrm{T}} & x_1\boldsymbol{y}^{\mathrm{T}} & x_2\boldsymbol{y}^{\mathrm{T}} & x_1^2\boldsymbol{y}^{\mathrm{T}} & x_1x_2\boldsymbol{y}^{\mathrm{T}} & x_2^2\boldsymbol{y}^{\mathrm{T}} & x_1^3\boldsymbol{y}^{\mathrm{T}} & x_1^2x_2\boldsymbol{y}^{\mathrm{T}} & x_1x_2^2\boldsymbol{y}^{\mathrm{T}} & x_2^3\boldsymbol{y}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}, \tag{6}
$$

because of the arrangement of the coefficients/coefficient matrices. The shift-invariant structure means that we can select rows of the basis matrix of the null space, multiply them by a polynomial in $\boldsymbol{x}$, and find the result again as a combination of rows in that same basis matrix. This shift operation

Input: coefficients or coefficient matrices of the problem (+ options for the algorithm)
1: **while** the solution subspace is not yet converged **do**
2: Enlarge the solution subspace **Step ①**
3: Check the rank structure of the solution subspace **Step ②**
4: **end while**
5: **if** the solution subspace is the null space **then** (*null space based approach*)
6: Perform the column compression **Step ③**
7: Construct multiplication maps from the compressed null space
8: **else** (*column space based approach*)
9: Construct multiplication maps from the column space **Step ④**
10: **end if**
11: Compute the eigenvalues of the multiplication maps
12: Cluster the affine solutions (optional) **Step ⑤**
13: Evaluate the residual errors **Step ⑥**
Output: affine solutions and residual errors (+ obtained information on the problem)

Fig. 3. Pseudocode for the (block) Macaulay matrix approach. Six coherent steps can be deduced from the pseudocode, which are explained and linked to the implementation of `MacaulayLab` in Section 4.

can be written mathematically as

$$\boldsymbol{S}_1\boldsymbol{V}\boldsymbol{D} = \boldsymbol{S}_2\boldsymbol{V}, \tag{7}$$

where the matrices $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ select rows, before and after multiplication with the polynomial in $\boldsymbol{D}$, from the basis matrix $\boldsymbol{V}$. The basis matrix $\boldsymbol{V}$ consists of (block) multivariate Vandermonde vectors constructed from the solutions of the problem and the linearly independent rows that we select are related to the solutions of the problem. In practice, we do not know the solutions and obtain a basis matrix $\boldsymbol{Z}$ via numerical linear algebra. Both basis matrices are related by a nonsingular transformation matrix $\boldsymbol{T}$, resulting in

$$(\boldsymbol{S}_1\boldsymbol{Z})\,\boldsymbol{T}\boldsymbol{D} = (\boldsymbol{S}_2\boldsymbol{Z})\,\boldsymbol{T}. \tag{8}$$

These matrix pairs, $(\boldsymbol{S}_2\boldsymbol{Z}, \boldsymbol{S}_1\boldsymbol{Z})$, that we obtain from shift operations with different monomials are the generalized multiplication matrices, the eigenvalues of which constitute the solutions of the problem in (1).

The above-described procedure is called the null spaced based approach. A similar procedure exists that uses the column space of the (block) Macaulay matrix to construct the multiplication maps. It starts from a linear algebra observation that every linearly independent row in the null space corresponds with a linearly dependent column in the column space. Instead of constructing the generalized eigenvalue problems from the linearly independent row of the null space, one can use the linearly dependent columns of the (block) Macaulay matrix, avoiding the construction of a basis of the null space. The pseudocode of both approaches is summarized in Fig. 3.

Constructing the generalized eigenvalue problems as explained above is only valid for problems with affine and simple solutions. Solutions at infinity need to be removed from the solution space before setting up the shift problems, which can be done via a column compression in the null space that deflates these solutions at infinity or happens implicitly when setting up de multiplication maps from the column space. This requires rank checks to determine the rank structure (i.e., checking how many affine solutions and solutions at infinity the solution space contains). Furthermore, the obtained affine solutions, being it from the null space or column space, can have multiplicities. The multiplicities cause the accuracy of the algorithm to drop. The (block) Macaulay matrix algorithm

in MacaulayLab uses a clustering step to identify and refine these multiple solutions, improving their accuracy.

4 Structure of the software package

After defining the problem in the correct format (Step ⓪), MacaulayLab solves the multivariate polynomial system or multiparameter eigenvalue problem in six steps:

- Step ①: enlarge the solution subspace,
- Step ②: check whether the solution subspace can accommodate the shift,
- Step ③: remove the solutions at infinity,
- Step ④: exploit the shift-invariance of the solution subspace,
- Step ⑤: cluster the affine solutions to obtain a better accuracy (optional), and
- Step ⑥: compute the residual errors of the obtained affine solutions.

Fig. 4 visualizes the main functions in MacaulayLab that enable these different steps and shows how they interact with each other. (Notice that `enlarge` and `check` are two local functions inside the solver `macaulaylab`.) While most of the time the same functions can deal with both problem types, this requires a very careful implementation that takes into account the subtle differences between them. The toolbox is built in a modular fashion, which allows easy expansions and improvements in future updates. Note that many of the functions depend on the chosen polynomial basis (implemented in `<basis>`) and monomial order (implemented in `<position>`), more information about this feature is provided in Section 5.

In the remainder of this section, we dive deeper into the six different steps required to solve both key problems via the (block) Macaulay matrix approach. We deal with one step at a time and highlight the different implementation decisions. It is important to note that implementation decisions always represent making a substantiated choice in the trade-off between computational efficiency and numerical robustness.

4.0 Define the problem in the correct format

In order to keep the toolbox user-friendly, describing a problem is kept very simple. Both problem types are internally represented by the same class `problemstruct`: all necessary information is stored in the cell arrays `coef` and `supp`[4]. Although it is also possible to submit the problem directly in its internal representation, the sub-classes `systemstruct` and `mepstruct` provide constructors to set-up the specific problems more easily (Fig. 5). The uniform internal data structure allows the functions of MacaulayLab to manipulate both types of problems similarly.

Code 2. We can use MacaulayLab to define the multivariate polynomial system given by $p_1(\boldsymbol{x}) = x_1^2 + x_1x_2 - 2 = 0$ and $p_2(\boldsymbol{x}) = x_2^2 + x_1x_2 - 2 = 0$.

```
>> syms x1 x2
>> p1 = x1^2 + x1*x2 - 2; p2 = x2^2 + x1*x2 - 2;
>> problem = problemstruct({p1,p2});
```

It is possible to define the problem directly in its internal representation via `problemstruct`.

[4]A cell array is a data type in MATLAB with indexed data containers, called cells, where each cell can contain any type of data. For MacaulayLab, each cell of `coef` and `supp` correspond to one polynomial (matrix) equation. A multidimensional array in the corresponding `coef` cell contains the coefficients/coefficient matrices, while a two-dimensional array in the `supp` cell stores the support of these coefficients/coefficient matrices.

Fig. 4. Diagram of the main functions (rectangles) implemented in MacaulayLab and their dependencies (arrows). The gray rectangles indicate local functions from `macaulaylab` and the dashed arrows denote dependencies that are only used in the null space based approach. Since the solution approach is similar for both problem types, the same six steps are used and most functions are capable of dealing directly with both problem types (scalars coefficients are replaced by coefficient matrices, and vice versa). Note that most functions depend on the chosen polynomial basis (implemented in `<basis>`) and monomial ordering (implemented in `<order>`).

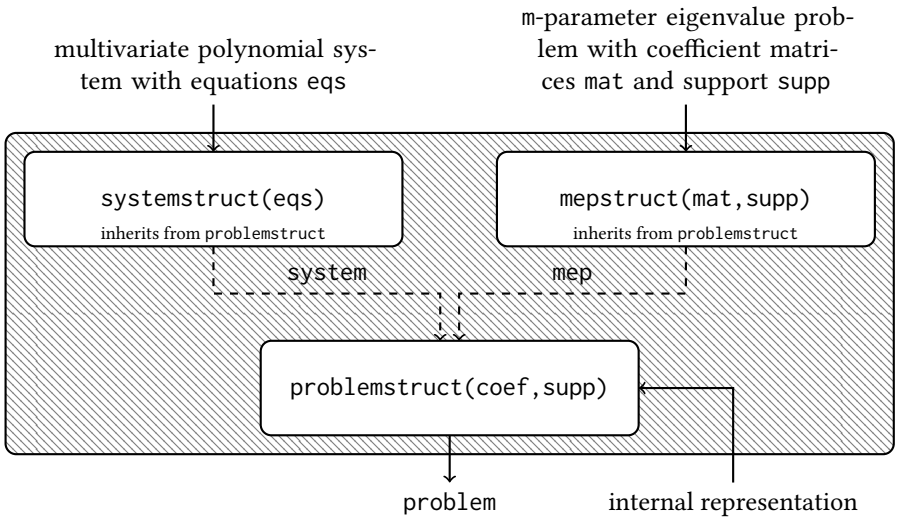


Fig. 5. Construction of the data structure to represent a multivariate polynomial system or a rectangular multiparameter eigenvalue problem. Both problem types are internally represented by the same `problemstruct`: all necessary information is stored in the cells `coef` and `supp`. The sub-classes `systemstruct` and `mepstruct` provide constructors to set-up the problems more easily, but it is also possible to submit the problem directly in its internal representation. It is possible to specify the basis in which the problem is constructed.

```
>> coef = {[1; 1; -2], [1; 1; -2]};
>> supp = {[2 0; 1 1; 0 0], [0 2; 1 1; 0 0]};
>> problem = problemstruct(coef,supp);
```

Similarly, we can define rectangular multiparameter eigenvalue problems via `mepstruct`. Quickly defining problems with random coefficients or coefficient matrices is also possible via `randsystem` and `randmep`.

4.1 Enlarge the solution subspace

The first step is to build and enlarge the solution subspace until it is large enough, which is checked for every degree in Step ② (see how this is implemented in Section 4.2). `MacaulayLab` supports two solution subspaces: the column space and right null space of the (block) Macaulay matrix. The column space can be enlarged iteratively via `macaulay` or recursively via `macaulayupdate`, in which the previous column space is used to build the new one. Enlarging the right null space can be done iteratively (via `macaulay` and `null`), recursively (via `macaulayupdate` and `nullrecrmacaulay`), or sparsely (via `nullsparsemacaulay`), in which case the right null space is determined without constructing the corresponding (block) Macaulay matrix. It is important to note that, depending on the chosen polynomial basis and monomial ordering, the resulting matrices can have a very different structure. However, everything is implemented independent of the polynomial basis and monomial ordering, as we explain in Section 5.

Code 3. The (block) Macaulay matrix of a problem gives us the column space. If the solution subspace is not large enough, it can be extended.

```
>> M = macaulay(problem,10);
>> M = macaulayupdate(M,11);
```

The other solution space is the null space, which can be constructed via a standard approach or via one of the two more efficient approaches.

```
>> Z = null(M);
>> Z = nullrecrmacaulay(Z,M);
>> Z = nullsparsemacaulay(Z,problem);
```

4.2 Check the rank structure

Of course, we need to know whether the solution subspace contains a gap zone that can accommodate the shift used in Step ④, i.e., whether the degree of the (block) Macaulay subspace is large enough. We do this via rank checks with an absolute or relative tolerance, which can be adapted by the user. There are two approaches implemented to do this: by looking at the increase of the rank for subsequent rows/columns or entire subsequent degree blocks. By default, we work block-wise per degree block, since this is numerically more robust and computationally more efficient than row-wise rank checks.

Code 4. For example, we can check whether the constructed null space has a gap zone by checking the rank block-wise per degree block.

```
>> [dgap, ma] = gap(Z, 2)

dgap =
    2

ma =
    2
```

The output of `gap` shows that there is a gap zone at degree two and that there are two affine solutions in the solution space. Notice that `MacaulayLab` also has a function to determine the linearly independent rows in a row-wise fashion (`stdmonomials`) and an adaptation of `gap` to check the rank structure based on these row indices (`gapstdmonomials`).

Both the basis matrix of the right null space and the flipped transpose of the (block) Macaulay matrix can be interpreted as block row matrices, when we consider the rows/columns degree block-wise. To improve computational efficiency, we can use recursive rank checks implemented in `recrrank`. This gives, together with the recursive construciton of the solution space in Step ①, the double recursive nature to the solution algorithm.

Note that we typically do not immediately check the rank structure for all iterations of the enlargement. For zero-dimensional solution sets, we wait until the nullity is stabilized, and we only start checking the rank structure of the solution space after stabilization. This way, we avoid many superfluous rank checks and create a more efficient solution algorithm. When the problem has a positive-dimensional solution space, it is useless to wait for stabilization, since this will never happen. By setting an optional flag (i.e., `posdim = true`), the solution algorithm does not wait for

the nullity to stabilize, but checks the rank structure of the solution subspace for every iteration, a necessity for these problems with a positive-dimensional solution space.

4.3 Perform the column compression

For the null space based approach, we need to deflate the solutions at infinity from the right null space via a column compression of its basis matrix, which is implemented via a single singular value decomposition. In the column space based approach, however, this is not necessary, since the backward QR decompositions in Step ④ to set-up the shift problems remove those infinite solutions implicitly.

Code 5. Where nullity with all solutions was four, the column compression removes the two solutions at infinity.

```
>> W11 = columncompr(Z,dgap,ma); rank(W11)

ans =
    2
```

4.4 Find eigenvalues of multiplication maps

Given a large-enough solution subspace, the shift problems are set-up as described in Section 3. We consider $n + 1$ shift problems: we shift with a random linear shift polynomial and with the n different solutions components (i.e., the variable coordinates x_i, for $i = 1, \ldots, n$). Other shift polynomials are possible, but using a random shift polynomial avoids false multiplicities [9]. We use a Schur decomposition to solve the first shift problem and re-use the obtained unitary matrix to obtain the upper triangular matrices of the other n shift problems by pre-multiplication and post-multiplication. This makes sure that the different components of the same solution are at the same position on the diagonal of the upper triangular matrices, which avoids the need for matching the different solution components. Note that the $n + 1$ default shift problems only consist of linear shifts. This has a clear advantage: We do only need a gap zone of one degree block. A shift polynomial with a large degree requires a large gap zone, hence more iterations are needed to construct a solution subspace that can accommodate this shift.

Code 6. We can set up the multiplication maps from the compressed null space. The rows with indices in `idx` are shifted by the polynomial x_1 in `shift` (`monomials` defines the monomials in the null space). The eigenvalues of the multiplication maps give the evaluation of the shift polynomial in the affine solutions.

```
>> [A, B] = multmapnull(W11, monomials, shift, idx);
>> eig(A,B)

ans =
    1.0000
   -1.0000
```

When choosing the column space based approach, every shift problem requires one backward QR decomposition to construct the necessary matrices. The backward QR decompositions can be implemented easily (when not considering the structure and sparsity of the (block) Macaulay

matrix) as `R = qr(fliplr(N)); R = fliplr(R);`. This backward QR decomposition implicitly removes the solutions at infinity.

To avoid the sparse multiplication of the row-selection and row-combination matrices with the subspace matrix when constructing the multiplication maps, we perform these multiplications indirectly and do not build the row-selection/combination matrices explicitly. It is again important to take into account the correct polynomial basis and monomial ordering when shifting, otherwise the results are meaningless.

4.5 Cluster the multiple solutions

An additional, but optional, step in the solution approach is to cluster the obtained affine solutions. We cluster the different solutions based on the evaluations of the random shift polynomial, which are obtained as the eigenvalues of the first shift problem. Clustering means that similar evaluations of the shift polynomial are considered to be equal solutions of the problems; hence, we construct clusters of multiple solutions. When the shift polynomial is chosen to be random (for other polynomials, we do not perform this clustering step by default), similar evaluations correspond to the same solutions with a probability equal to one. Every solution for which the evaluation of the random shift polynomial is grouped in the same cluster, is considered to be the same solution. Therefore, we take the geometric mean of the cluster for every component of the solution (i.e., the cluster center). The idea for this additional clustering step comes from [9] in the context of polynomial system solving, but is also very useful when solving rectangular multiparameter eigenvalue problems.

Code 7. We use a specific bivariate polynomial systems to demonstrate the clustering step.

```
>> syms x1 x2
>> p1 = x2^2 - 4*x2 + 4; p2 = x1^2 - 2x1*x2 + 2*x1 + x2^2 - 2*x2 + 1;
>> problem = systemstruct({p1, p2});
```

This problem has a quadruple solution in $(1, 2)$. The clustering step improves the accuracy visibly.

```
>> macaulaylab(problem, clustering = false)

ans =
    1.0000 + 0.0000i     2.0000 + 0.0000i
    1.0000 - 0.0000i     2.0000 - 0.0000i
    0.8425 + 0.0000i     2.2067 + 0.0000i
    1.1575 + 0.0000i     1.7933 + 0.0000i

>> macaulaylab(problem, clustering = true)

ans =
    1.0000 + 0.0000i     2.0000 + 0.0000i
    1.0000 + 0.0000i     2.0000 + 0.0000i
    1.0000 + 0.0000i     2.0000 + 0.0000i
    1.0000 + 0.0000i     2.0000 + 0.0000i
```

4.6 Compute the residual errors

Finally, we compute the (absolute and relative) residual error for every obtained affine solution:

- For a multivariate polynomial system, we evaluate all the polynomials in the obtained affine solution and sum the absolute values of (residual) values. The relative residual error is obtained by dividing the absolute residual by the sum of the absolute coefficients.
- For a rectangular multiparameter eigenvalue problem, we evaluate the matrix polynomial in the obtained affine solutions and compute the associated eigenvector as the right-most right singular vector of the matrix. The 2-norm of the matrix-vector product is the absolute residual, which can again be divided by the coefficients of the problem to obtain the relative residual error.

In `MacaulayLab`, we use a clever way to compute both absolute residuals. Regardless of the problem type, we compute the smalles singular value of the evaluated problem. This values corresponds to the magnitude of the vector of evaluated polynomials when the problem is multivariate polynomial system, while it corresponds to smallest possible residual that we can obtain in the matrix-vector product between the matrix polynomial and its eigenvector. Depending on the chosen polynomial basis, a particular evaluation function `value` is used to achieve this goal.

Code 8. The (absolute and relative) residual can be computed for every point $\boldsymbol{x} \in \mathbb{C}^m$.

```
>> residuals([0.9999 1.9999], problem)

ans =
    5.6566e-04
```

Note that this information is also stored in the output structure of `MacaulayLab`.

5 Polynomial basis and monomial ordering

`MacaulayLab` is implemented independently from the specific polynomial basis and monomial ordering. This means that the user can choose the polynomial basis and monomial ordering that suits the application best, while all the functions keep working out-of-the-box. How is this implemented in the toolbox?

- Every time it is necessary to shift (i.e., multiply) two monomials in the algorithm, the functions call the chosen function `<basis>`, which implements the basic shift operation in a particular polynomial basis. By using the correct shift function, the other functions do not need to care about the specific polynomial basis. For example, in the standard monomial basis, when shifting $\phi_{11}(x) = x_1 x_2$ with $\phi_{10}(x) = x_1$, `monomial` results in $\phi_{21}(x) = x_1^2 x_2$, while `chebyshev`, which implements the Chebyshev basis, yields $\frac{1}{2}\left(\phi'_{21}(x) + \phi'_{01}(x)\right)$, where the basis polynomials now correspond to $\phi'_{21}(x) = \left(2x_1^2 - 1\right) x_2$ and $\phi'_{01}(x) = x_2$. The `<basis>` also contains the necessary information to evaluate the matrix polynomial(s) correctly.
- Similarly, the monomial ordering is important when we need to know the position of a monomial in the polynomial basis (for example, to build the (block) Macaulay matrix or set-up the shift problems). By leaving this computation to a dedicated function that implements the correct monomial ordering `<order>`, the correct position in the chosen monomial ordering is always obtained. For example, using `grinvlex` places the bivariate monomial $x_1^2 x_2$ at position 8, while `grevlex` yields 9.

By default, `MacaulayLab` uses the standard monomial basis and graded reverse lexicographic ordering. The toolbox also contains the pre-implemented Chebyshev basis, graded lexicographic ordering, graded inverse lexicographic ordering, and graded negative lexicographic ordering. Users can easily implement and use other polynomial bases or monomial orderings.

Code 9. It is easy to solve a problem given in the Chebyshev polynomial basis, while using, for example, the graded lexicographic ordering.

```
>> solutions = macaulaylab(problem,basis = @chebyshev,order = @grlex);
```

6 Functionalities of the toolbox

The previous sections already showed that MacaulayLab has many options that can be chosen from, for example, whether to use the right null space or column space to set up the multiplication maps. To learn more about the different options, we once more refer the interested reader to the manual of MacaulayLab (www.macaulaylab.net). In this section, we discuss the most important options and compare the different solution approaches (Section 6.2). However, we first present the database of test problems that is developed alongside the toolbox (Section 6.1), since we will use its test problems to demonstrate MacaulayLab.

6.1 Database with examples

Parallel with MacaulayLab, we also gathered different multivariate polynomial systems and rectangular multiparameter eigenvalue problems in a database, which can be used to test the toolbox's features and act as benchmarks for other software. The database is available on the website of MacaulayLab (www.macaulaylab.net) and can be downloaded separately. It includes a wide variety of problems, which were gathered from our own research, existing repositories (such as `Test Database of Polynomial Systems` and `Posso Test Suite`), and papers that we encountered (a reference to the original source is included with every test problem).

At the time of writing this paper, the database contains 290 multivariate polynomial systems and 30 rectangular multiparameter eigenvalue problem. In Tables 1 and 2, we highlight some systems and multiparameter eigenvalue problems from that database. Additional information about the problems is also stored and available in the database, accessible by calling the overloaded function `disp` (Fig. 6). For the problems that we solved via MacaulayLab, we also provide the numerical solutions in a separate file. Furthermore, the database includes scripts that illustrate the applications behind the problems and functions to interact with the database.

Because MacaulayLab is polynomial basis independent (see Section 5), the database also contains problems that are given in another polynomial basis. When using such a problem, the MacaulayLab solver recognizes the correct polynomial basis automatically and applies the correct shift and evaluation operations (if the corresponding functions are implemented). These problems can be recognized easily in the database, because an identifier for the polynomial basis is appended to the name. For example, `problem_cheb.m` for the Chebyshev polynomial basis.

6.2 Different solution approaches in MacaulayLab

To demonstrate the different solution options available in MacaulayLab, we apply the solver to two problems from the database using various option configurations. Fig. 7 compares the relative computation times for the NOON4 multivariate polynomial system and the CUBE rectangular multiparameter eigenvalue problem. All computation times are expressed relative to the slowest configuration in each comparison. The first configuration employs the column space approach, while the remaining configurations explore different null space based approaches.

The first two configurations utilize column-wise and row-wise rank checks combined with column/row-wise shift operations. These configurations are notably slower due to the large number of rank checks required. Moreover, rank checks on the columns of the (block) Macaulay matrix are

Table 1. Small selection of multivariate polynomial systems in the database, with some of their key properties: a system has s equations in m variables with maximum total degree equal to d, resulting in a total of m_b solutions, of which m_a are affine solutions. The database contains a wide variety of problems. An underscore is used to indicate whether a problem is given in another polynomial basis than the standard monomial basis.

name	s	d	m	m_b	m_a
NOON3	3	3	3	27	21
BATSELIER5	3	12	3	1728	1728
CONFORM	3	4	3	64	16
DREESEN10	4	4	4	∞	2
CYCLIC5	5	7	5	120	70
WALSH	6	7	6	∞	7
KATSURA7	8	2	8	128	128
OVERDET1	8	4	4	10	10
WALSH_CHEB	6	3	6	∞	7

Table 2. Small selection of rectangular multiparameter eigenvalue problems in the database, with some of their key properties: an m-parameter eigenvalue problem has maximum total degree d and $k \times l$ coefficient matrices, resulting in a total of m_b solutions, of which m_a are affine solutions. The database contains a wide variety of problems. An underscore is used to indicate whether a problem is given in another polynomial basis than the standard monomial basis.

name	d	m	k	l	m_b	m_a
VOLKMER	1	2	12	6	6	6
MUHIC4	1	2	18	9	∞	4
ALSUBAIE3	1	3	5	3	∞	4
H2FOM2R3	2	3	10	8	∞	209
HKP2	2	2	3	2	12	12
H2F5	2	2	6	5	∞	17
H2FOURDISK	2	4	10	7	∞	129
CUBE	3	2	21	20	1890	1890
WING_CHEB	2	1	3	3	6	6

more time-consuming than those on the rows of the null space, as the column vectors are typically longer.

The recursive and sparse enlargement of the solution space significantly reduce the computation time. Although the performance gains are modest for the selected examples, these strategies become much more effective for larger problems. The sparse method offers the additional advantage of eliminating the need to store the (block) Macaulay matrix in memory.

Enabling the `posdim = true` flag roughly doubles the computation time for both problems. This option introduces numerous unnecessary rank checks (for problems with zero-dimensional solution sets), which results in substantial computational overhead.

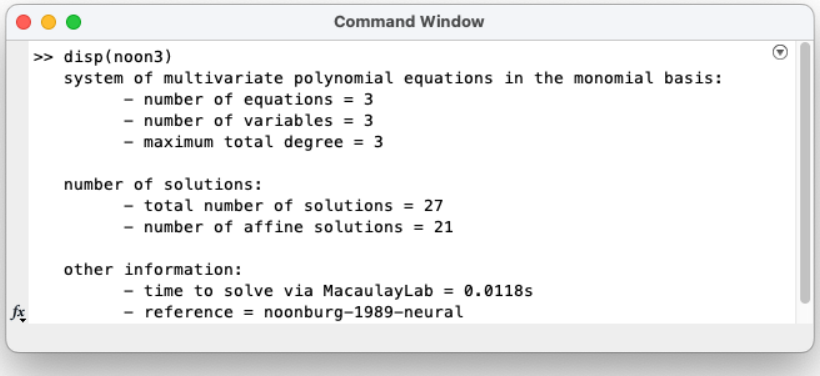


Fig. 6. Information displayed for the NOON3 system in the database.

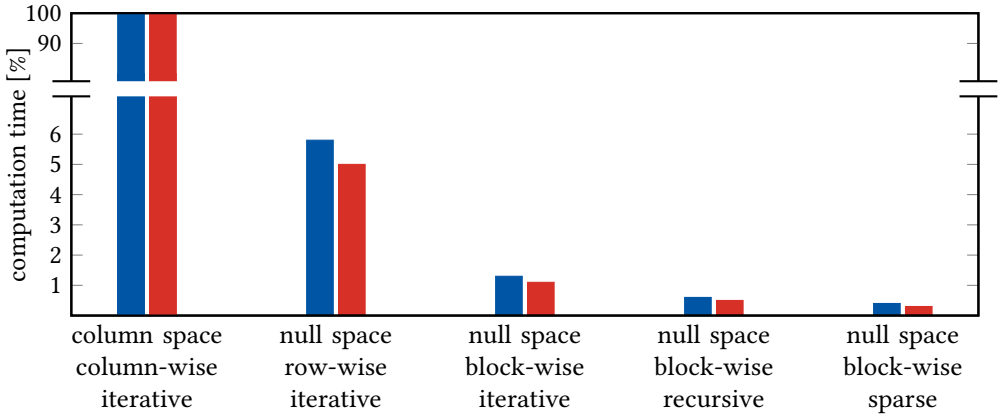


Fig. 7. Different configuration for MacaulayLab are used to solve the NOON4 multivariate polynomial system ( ) and CUBE rectangular multiparameter eigenvalue problem ( ). The options are mentioned below the computation times: the first option is the solution space, the second option is the decision of how the rank checks and shifts in the multiplication maps are performed, and the third option is the chosen enlargement strategy.

7 Comparison with other solvers

We compare MacaulayLab with other solvers for multivariate polynomial systems and rectangular multiparameter eigenvalue problems. The test problems that we consider is a subset of our database in Section 6.1. We use performance profiles to visualize the comparison. For a thorough explanation on performance profiles, we refer the interested reader to [13]. However, a minimal introduction is given below.

For each problem p in the test set $\mathcal{P}$ and solver s in the comparison set $\mathcal{S}$, we use $t_{p,s}$ to denote the computation time required to solve problem p by solver s. The performance of a solver s on a

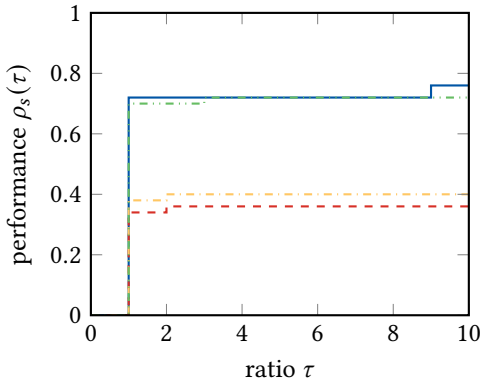


Fig. 8. Performance profiles for a subset of 50 multivariate polynomial systems. We run MacaulayLab (———), PNLA, using the singular value (- - -) or QR decomposition (-·-·-), and PHClab (-··-··) on this subset.

problem p is given by the performance ratio,

$$r_{p,s} = \frac{t_{p,s}}{\min_{\hat{s}}\{t_{p,\hat{s}} : \hat{s} \in \mathcal{S}\}}, \tag{9}$$

which compares the solver s with the best obtained result. The performance profile $\rho_s(\tau)$ of a solver,

$$\rho_s(\tau) = \frac{1}{n_p}\#\{p \in \mathcal{P} : r_{p,s} \leq \tau\} \tag{10}$$

reveals the probability for the solver s to be within a factor τ of the best possible performance ratio. The performance profile $\rho_s(\tau) : \mathbb{R} \to [0, 1]$ is the (cumulative) distribution function for the performance ratio $r_{p,s}$ and is a nondecreasing, piecewise constant function. Notice that the value of $\rho_s(1)$ is the probability that the solver will win over the rest of the solvers in $\mathcal{S}$.

7.1 Other multivariate polynomial system solvers

We compare MacaulayLab with the PNLA toolbox, which is using the singular value or QR decomposition, PHClab toolbox (a MATLAB interface to PHCpack), and Maple. The test problems is a random subset of 50 multivariate polynomial systems of the database. The performance profile is given in Fig. 8.

7.2 Other multiparameter eigenvalue solvers

For rectangular multiparameter eigenvalue problems, the 30 problems in the databased are solved by MacaulayLab and MultiParEig. Since MultiParEig only works with linear problems, all polynomial rectangular multiparameter eigenvalue problems are linearized prior to solving them with the toolbox. While the most efficient null space based approach is used for MacaulayLab, two approaches of MultiParEig are used in the comparison: a compressed Kronecker matrix approach that creates the multiplication maps directly and a randomized sketching approach that transforms the problem into a system of square multiparameter eigenvalue problems. The performance profiles of the three solvers are given in Fig. 9. Because the problem set contains a lot of polynomial problems, MacaulayLab has an overall better performance than the other two approaches that require a linearized version of the problems.

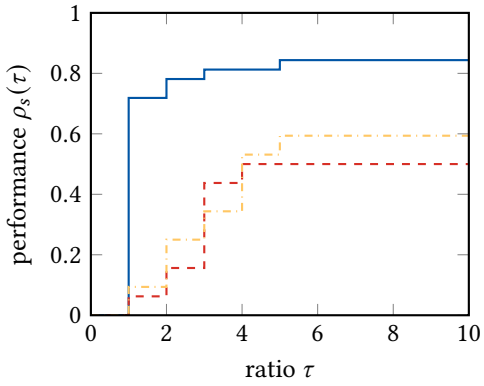


Fig. 9. Performance profiles for 30 rectangular multiparameter eigenvalue problems. We compare `MacaulayLab` (——) with the Kronecker matrix approach (- - -) and randomized sketching approach (-·-·-) of `MultiParEig`. The problem set contains a lot of polynomial problems, which explains the overall better preformance of `MacaulayLab`.

8 Current limitations and future developments

Taking inspiration from other toolboxes and recent advances in the literature, it is evident that future releases of `MacaulayLab` will contain additional features:

- Since the sparse, structured (block) Macaulay matrix constitutes the core of the toolbox, considering the arrangement of the elements in the matrices is an important avenue to further improve the computational capabilities of the code. The recursive and sparse construction of a basis matrix of the null space already do this to a large extent. We will consider the sparsity in the involved matrices and investigate the advantages and disadvantages of working with the QR decomposition instead of the singular value decomposition [3].
- Taking into account the support of the (matrix) polynomials could help to reduce the size of the involved matrices a priori and sounds very promising [5, 6].
- Performing rank checks is an important step in the (block) Macaulay matrix approach. A lot of care is necessary with these rank decisions. In one of the next releases, more support to take the correct rank decisions (i.e., different types of rank decisions, visual aids, etc.) will be added to the toolbox. Related to these rank checks are also the introduction of a block column approach, which would reduce the number of rank checks and improve the robustness of the column space based approach, and the consideration of using projective shifts [4], which could eliminate the need for rank checks in specific situations (like zero-dimensional solution sets). The advantages and disadvantages of these adaptations are prone to additional research.
- Instead of computing the eigenvalues of multiple generalized eigenvalue problems independently and using clustering step to combine and refine the solutions, we could use simultaneous triangularization to compute the solutions of the different shift problems faster and more accurate [19]. Combining the different generalized eigenvalue problems in a tensor structure could be another interesting approach to explore [33].
- Finally, we also want to give the user the possibility to run the solver in an interactive mode, where the user can choose *at run time* which options and approach to use. Such an *interactive solver* would be very useful for users who are not familiar with the toolbox or want to learn more about the problem's properties.

9 Conclusion

Throughout this paper, we emphasized the versatility and potential of `MacaulayLab` as a toolbox for solving multivariate polynomial systems and rectangular multiparameter eigenvalue problems. `MacaulayLab` stands out as a general-purpose multivariate polynomial system solver, capable of dealing with systems in different polynomial bases, overdetermined systems, and systems with positive-dimensional solution sets at infinity. By introducing the block Macaulay matrix as natural extension of the (scalar) Macaulay matrix, `MacaulayLab` becomes the first dedicated toolbox to tackle rectangular multiparameter eigenvalue problems, while maintaining the same flexibility as when solving systems of multivariate polynomial equations.

While homotopy continuation may offer faster solution algorithms, these methods may suffer from ill-conditioning and (only) deal with square systems, leaving a niche that can be filled by the Macaulay matrix algorithms from this (and other) toolbox(es), particularly for overdetermined systems of multivariate polynomial equations. In comparison to other `MATLAB` toolboxes that use the Macaulay matrix, `MacaulayLab` is a clear improvement: while being built modular and offering many different solution approaches and options, it is computationally more efficient and numerically more robust than its historical predecessors `RootFinding` and `PNLA`.

It is important to stress that `MacaulayLab` is designed to be independent of the polynomial basis and monomial ordering, which provides an easy way of solving problems that are given in a different representation (without needing to change the representation first). This can be very useful in applications where the results depend on the chosen polynomial basis or monomial ordering.

Additionally, the `MacaulayLab` toolbox also contains a database with many test problems. We believe that the inclusion of this comprehensive set of test problems could prove to be highly valuable for its users.

Acknowledgments

The authors want to thank several colleagues for testing the software packages and providing them with useful comments and suggestions. We acknowledge in that regard the help from Benoît Legat, Hans van Rooij, Katrien De Cock, Lukas Vanpoucke, Oscar Mauricio Agudelo, Sarthak De, Sem Viroux, and Sibren Lagauw.

This work was supported in part by the KU Leuven under Grant No. IBOF/23/064, C3/20/117, and C3I/21/00316; in part by the Fonds Wetenschappelijk Onderzoek under Grant No. S005319 and Grant No. T001919N; in part by the Departement Economie, Wetenschap & Innovatie via the Flanders AI Research Program; in part by the Vlaams Agentschap Innoveren & Ondernemen under Grant No. HBC/2021/0076; and in part by the European Research Council under Grant No. 885682.

References

[1] Oscar M. Agudelo, Christof Vermeersch, and Bart De Moor. 2021. Globally Optimal $\mathcal{H}_2$-Norm Model Reduction: A Numerical Linear Algebra Approach. *IFAC-PapersOnLine* 54, 9 (2021), 564–571. Part of special issue: 24th International Symposium on Mathematical Theory of Networks and Systems (MTNS).

[2] Daniel J. Bates, Jonathan D. Hauenstein, Sommese Andrew J., and Charles W. Wampler, II. 2013. Bertini: Software for Numerical Algebraic Geometry. https://bertini.nd.edu.

[3] Kim Batselier. 2013. *A Numerical Linear Algebra Framework for Solving Problems with Multivariate Polynomials.* Ph. D. Dissertation. KU Leuven, Leuven, Belgium.

[4] Kim Batselier, Philippe Dreesen, and Bart De Moor. 2014. On the Null Spaces of the Macaulay Matrix. *Linear Algebra Appl.* 460 (2014), 259–289.

[5] Matías R. Bender. 2022. Solving Sparse Polynomial Systems Using Gröbner Bases and Resultants. In *Proc. of the 2022 International Symposium on Symbolic and Algebraic Computation (ISSAC).* Villeneuve-d'Ascq, France, 21–30.

[6] Matías R. Bender and Simon Telen. 2022. *Yet another Eigenvalue Algorithm for Solving Polynomial Systems.* Technical Report. Technische Universität Berlin, Berlin, Germany.

[7] Paul Breiding and Sascha Timme. 2018. HomotopyContinuation.jl: A Package for Homotopy Continuation in Julia. In *Proc. of the 6th International Conference on Mathematical Software*, James H. Davenport, Manuel Kauers, George Labahn, and Josef Urban (Eds.). South Bend, IN, USA, 458–465.

[8] Tian-ran Chen, Tsung-Lin Lee, and Tien-Yien Li. 2014. Hom4PS-3: A Parallel Numerical Solver for Systems of Polynomial Equations Based on Polyhedral Homotopy Continuation Methods. In *Proc. of the 4th International Congress on Mathematical Software (ICMS)*. Seoul, South Korea, 183–190.

[9] Robert M. Corless, Gianni M. Patrizia, and Barry M. Trager. 1997. A Reordered Schur Factorization Method for Zero-Dimensional Polynomial Systems with Multiple Roots. In *Proc. of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC)*. Maui, HI, USA, 133–140.

[10] David A. Cox. 2020. *Applications of Polynomial Systems.* Regional Conference Series in Mathematics, Vol. 134. Conference Board of the Mathematical Sciences, Providence, RI, USA.

[11] Bart De Moor. 2019. Least Squares Realization of LTI Models Is an Eigenvalue Problem. In *Proc. of the 18th European Control Conference (ECC)*. Naples, Italy, 2270–2275.

[12] Bart De Moor. 2020. Least Squares Optimal Realisation of Autonomous LTI Systems Is an Eigenvalue Problem. *Communications in Information and Systems* 20, 2 (2020), 163–207.

[13] Elizabeth D. Dolan and Jorge J. Moré. 2002. Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming* 91 (2002), 201–213.

[14] Philippe Dreesen. 2013. *Back to the Roots: Polynomial System Solving Using Linear Algebra.* Ph. D. Dissertation. KU Leuven, Leuven, Belgium.

[15] Jean-Charles Faugère, Patrizia Gianni, Daniel Lazard, and Teo Mora. 1993. Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. *Journal of Symbolic Computation* 16, 4 (1993), 329–344.

[16] Jean-Charles Faugère, Françoise Levy-dit Vehel, and Ludovic Perret. 2008. Cryptanalysis of MinRank. In *Proc. of the 28th Annual International Cryptology Conference*. Santa Barbara, CA, USA, 280–296.

[17] Takayuki Gunji, Sunyoung Kim, Katsuki Fujisawa, and Masakuzu Kojima. 2006. PHoMpara – Parallel Implementation of the Polyhedral Homotopy Continuation Method for Polynomials Systems. *Computing* 77 (2006), 387–411.

[18] Takayuki Gunji, Sunyoung Kim, Masakuzu Kojima, Akiko Takeda, Katsuki Fujisawa, and Tomohiko Mizutani. 2004. PHoM – a Polyhedral Homotopy Continuation Method for Polynomial Systems. *Computing* 73 (2004), 57–77.

[19] Haoze He, Daniel Kressner, and Bor Plestenjak. 2024. Randomized Methods for Computing Joint Eigenvalues, with Applications to Multiparameter Eigenvalue Problems and Root Finding. *Numerical Algorithms* (2024), 1–32.

[20] Michiel E. Hochstenbach, Tomaž Košir, and Bor Plestenjak. 2024. Numerical Methods for Rectangular Multiparameter eigenvalue Problems, with Applications to Finding Optimal ARMA and LTI Models. *Numerical Linear Algebra with Applications* 31, 2 (2024), e2540:1–23.

[21] Sibren Lagauw, Oscar M. Agudelo, and Bart De Moor. 2023. Globally Optimal SISO H_2-Norm Model Reduction Using Walsh's Theorem. *IEEE Control System Letters* 7 (2023), 1670–1675.

[22] Anton Leykin. 2011. Numerical Algebraic Geometry. *Journal of Software for Algebra and Geometry* 3 (2011), 5–10.

[23] Tien-Yien Li. 1997. Numerical Solution of Multivariate Polynomial Systems by Homotopy Continuation Methods. *Acta Numerica* 6 (1997), 399–436.

[24] Alexander P. Morgan. 1987. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice-Hall, Englewood Cliffs, NJ, USA.

[25] Bor Plestenjak. 2023. MultiParEig: Toolbox for Multiparameter Eigenvalue Problems. https://mathworks.com/matlabcentral/fileexchange/47844-multipareig. (Visited on July 8, 2023).

[26] Bor Plestenjak and Michiel E. Hochstenbach. 2016. Roots of Bivariate Polynomial Systems via Determinantal Representations. *SIAM Journal on Scientific Computing* 38, 2 (2016), 765–788.

[27] Boris Shapiro. 2010. Algebro-Geometric Aspects of Heine–Stieltjes Theory. *Journal of the London Mathematical Society* 83, 1 (2010), 36–56.

[28] Andrew J. Sommese, Jan Verschelde, and Charles W. Wampler, II. 2001. Numerical Decomposition of the Solution Sets of Polynomial Systems into Irreducible Components. *SIAM J. Numer. Anal.* 38, 6 (2001), 2022–2046.

[29] Andrew J. Sommese and Charles W. Wampler, II. 2005. *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science.* World Scientific Publishing Company, Singapore.

[30] Simon Telen, Bernard Mourrain, and Marc Van Barel. 2018. Solving Polynomial Systems via Truncated Normal Forms. *SIAM J. Matrix Anal. Appl.* 39, 3 (2018), 1421 –1447.

[31] Llyod N. Trefethen. 2011. Six Myths of Polynomial Interpolation and Quadrature. *Mathematics Today* (2011), 184–188.

[32] Lloyd N. Trefethen. 2019. *Approximation Theory and Approximation Practice* (extended ed.). SIAM, Philadelphia, PA, USA.

[33] Jeroen Vanderstukken and Lieven De Lathauwer. 2021. Systems of Polynomial Equations, Higher-Order Tensor Decompositions and Multidimensional Harmonic Retrieval: An Unifying Framework. Part I: The Canonical Polyadic Decomposition. *SIAM J. Matrix Anal. Appl.* 42, 2 (2021), 883–912.

[34] Christof Vermeersch and Bart De Moor. 2019. Globally Optimal Least-Squares ARMA Model Identification Is an Eigenvalue Problem. *IEEE Control Systems Letters* 3, 4 (2019), 1062–1067.

[35] Jan Verschelde. 1996. *Homotopy Continuation Methods for Solving Polynomial Systems.* Ph. D. Dissertation. KU Leuven, Leuven, Belgium.

[36] Jan Verschelde. 1999. Algorithm 795: PHCpack: A General-Purpose Solver for Polynomial Systems by Homotopy Continuation. http://homepages.math.uic.edu/~jan/PHCpack/phcpack.html. *ACM Trans. Math. Software* 25, 2 (1999), 251–276.

[37] Layne T. Watson, Stephen C. Billups, and Alexander P. Morgan. 1987. Algorithm 652: HOMPACK: A Suite of Codes for Globally Convergent Homotopy Algorithms. *ACM Trans. Math. Software* 13, 3 (1987), 281–310.