Fraction-free Comprehensive LU Decomposition and Variants

Aishat Olagunju, Marc Moreno Maza, David Jeffrey

Western University

September 22, 2025

Ontario Research Centre for Computer Algebra

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.
- This yields various computational challenges, in particular:

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.
- This yields various computational challenges, in particular:
  1. (possibly too) many cases,

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.
- This yields various computational challenges, in particular:
  1. (possibly too) many cases,
  2. expression swell (either due to simplication or to no simplication)

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.
- This yields various computational challenges, in particular:
  1. (possibly too) many cases,
  2. expression swell (either due to simplication or to no simplication)

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.
- This yields various computational challenges, in particular:
  1. (possibly too) many cases,
  2. expression swell (either due to simplication or to no simplication)
  3. large running times caused by the above.

Overview

- We study the PLU decomposition of rectangular matrices with polynomial coefficients, depending on parameters, possibly subject to algebraic constraints.
- Computing such a parametric decomposition typically requires to split computations into cases.
- This yields various computational challenges, in particular:
  1. (possibly too) many cases,
  2. expression swell (either due to simplication or to no simplication)
  3. large running times caused by the above.
- This paper is an experimental comparison of different strategies for doing PLU.

Definition

LU Decomposition

Given a matrix A, it can be decomposed (factorized) into a permutation matrix P, a lower triangular matrix L and an upper triangular matrix U.

$$A = PLU$$

Definition

LU Decomposition

Given a matrix A, it can be decomposed (factorized) into a permutation matrix P, a lower triangular matrix L and an upper triangular matrix U.

$$A = PLU$$

This is the most common method, where the L matrix ends up with $1's$ on the diagonal.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Recursive PLU Steps

If the first column of A contains a non-zero entry a, we write

$$P_1 A = \left[\begin{array}{c|c} a & w^T \\ \hline v & A' \end{array}\right]$$

and set $c = 1/a$, otherwise we set $c = 0$.

Recursive PLU Steps

If the first column of A contains a non-zero entry a, we write

$$P_1 A = \left[\begin{array}{c|c} a & w^T \\ \hline v & A' \end{array}\right]$$

and set $c = 1/a$, otherwise we set $c = 0$.
We make a recursive call on A' and write:

$$P_1 A = \left[\begin{array}{c|c} 1 & 0 \\ \hline cv & I_{n-1} \end{array}\right] \left[\begin{array}{c|c} a & w^T \\ \hline 0 & A' - cvw^T \end{array}\right]$$

where $P'(A' - cvw^T) = L'U'$.

Recursive PLU Steps

If the first column of A contains a non-zero entry a, we write

$$P_1 A = \left[\begin{array}{c|c} a & w^T \\ \hline v & A' \end{array}\right]$$

and set $c = 1/a$, otherwise we set $c = 0$.
We make a recursive call on A' and write:

$$P_1 A = \left[\begin{array}{c|c} 1 & 0 \\ \hline cv & I_{n-1} \end{array}\right] \left[\begin{array}{c|c} a & w^T \\ \hline 0 & A' - cvw^T \end{array}\right]$$

where $P'(A' - cvw^T) = L'U'$.
Let $v' = P'v$, thus

$$\left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & P' \end{array}\right] P_1 A = \left[\begin{array}{c|c} 1 & 0 \\ \hline cv' & L' \end{array}\right] \left[\begin{array}{c|c} a & w^T \\ \hline 0 & U' \end{array}\right].$$

Matrices depending on parameters

There are parametric matrices that are of interest in practice. So it is natural to adapt linear algebra algorithms like LU decomposition, rank computation, Jordan, Hermite and Smith normal forms .

Matrices depending on parameters

There are parametric matrices that are of interest in practice. So it is natural to adapt linear algebra algorithms like LU decomposition, rank computation, Jordan, Hermite and Smith normal forms .

Moon's matrix about chaotic vibrations [1]

$$J = \begin{bmatrix} -a & -b & -d & 0 \\ b & -a & 0 & -d \\ d & 0 & -a & -b \\ 0 & d & b & -a \end{bmatrix}$$

Matrices depending on parameters

There are parametric matrices that are of interest in practice. So it is natural to adapt linear algebra algorithms like LU decomposition, rank computation, Jordan, Hermite and Smith normal forms .

Moon's matrix about chaotic vibrations [1]

$$J = \begin{bmatrix} -a & -b & -d & 0 \\ b & -a & 0 & -d \\ d & 0 & -a & -b \\ 0 & d & b & -a \end{bmatrix}$$

5X5 Kac Murdock Szegö Matrix [2]

$$A5 = \begin{bmatrix} 1 & -p & 0 & 0 & 0 \\ -p & p^2+1 & -p & 0 & 0 \\ 0 & -p & p^2+1 & -p & 0 \\ 0 & 0 & -p & p^2+1 & -p \\ 0 & 0 & 0 & -p & 1 \end{bmatrix}$$

Comprehensive LU Decomposition: Example

Given an input matrix $\mathbf{A}$ and an initially empty constructible set cs:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ ca & 0 & a & 0 \\ 0 & -ca & 0 & -a \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$\mathbf{ComprehensiveLU}(\mathbf{A}, \mathbf{cs})$ returns:

Comprehensive LU Decomposition: Example

Given an input matrix $\mathbf{A}$ and an initially empty constructible set cs:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ ca & 0 & a & 0 \\ 0 & -ca & 0 & -a \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$\mathbf{ComprehensiveLU}(\mathbf{A}, \mathbf{cs})$ returns:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -ca & 1 & 0 \\ ca & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -a \\ 0 & 0 & 0 & 0 \end{bmatrix} \left\{ a = 0 \right. \quad or \left\{ \begin{array}{l} c - 1 = 0 \\ a \neq 0 \end{array} \right.$$

Comprehensive LU Decomposition: Example

Given an input matrix $\mathbf{A}$ and an initially empty constructible set cs:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ ca & 0 & a & 0 \\ 0 & -ca & 0 & -a \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$\mathbf{ComprehensiveLU}(\mathbf{A}, \mathbf{cs})$ returns:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -ca & 1 & 0 \\ ca & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ ca & 0 & 1 & 0 \\ 0 & -ca & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -a \\ 0 & 0 & 0 & 0 \end{bmatrix} \left\{ a = 0 \right. \quad or \left\{ \begin{array}{l} c - 1 = 0 \\ a \neq 0 \end{array} \right. \qquad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -ac + a & 0 \\ 0 & 0 & 0 & -a \end{bmatrix} \left\{ \begin{array}{l} a \neq 0 \\ c - 1 \neq 0 \end{array} \right.$$

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

- we search for pivots must not be zero-divisors w.r.t. the constructible set, and this is where computations may split,

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

- we search for pivots must not be zero-divisors w.r.t. the constructible set, and this is where computations may split,
- thus the recursive calls may return several branches.

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

- we search for pivots must not be zero-divisors w.r.t. the constructible set, and this is where computations may split,
- thus the recursive calls may return several branches.

Specification

Input: $\mathbf{A}$ be an $\mathbf{m} \times \mathbf{n}$-matrix over $\mathbb{K}[X_1, \ldots, X_v]$ and W be a constructible set given by polynomials of $\mathbb{K}[X_1, \ldots, X_v]$.

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

- we search for pivots must not be zero-divisors w.r.t. the constructible set, and this is where computations may split,
- thus the recursive calls may return several branches.

Specification

Input: $\mathbf{A}$ be an $\mathbf{m} \times \mathbf{n}$-matrix over $\mathbb{K}[X_1, \ldots, X_v]$ and W be a constructible set given by polynomials of $\mathbb{K}[X_1, \ldots, X_v]$.

Output: ComprehensiveLU$(\mathbf{A}, W)$ returns a list of tuples $[\mathbf{P}_i, \mathbf{Q}_i, \mathbf{L}_i, \mathbf{U}_i, W_i]$, called *branches*, for $1 \leq i \leq e$:

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

- we search for pivots must not be zero-divisors w.r.t. the constructible set, and this is where computations may split,
- thus the recursive calls may return several branches.

Specification

Input: $\mathbf{A}$ be an $\mathbf{m} \times \mathbf{n}$-matrix over $\mathbb{K}[X_1, \ldots, X_v]$ and W be a constructible set given by polynomials of $\mathbb{K}[X_1, \ldots, X_v]$.

Output: ComprehensiveLU$(\mathbf{A}, W)$ returns a list of tuples $[\mathbf{P}_i, \mathbf{Q}_i, \mathbf{L}_i, \mathbf{U}_i, W_i]$, called *branches*, for $1 \leq i \leq e$:

1. the W_i's form a partition of W.

ComprehensiveLU is an adaptation of RecursiveLU to matrices depending on parameters [3]. The main enhancements are:

- we search for pivots must not be zero-divisors w.r.t. the constructible set, and this is where computations may split,
- thus the recursive calls may return several branches.

Specification

Input: $\mathbf{A}$ be an $\mathbf{m} \times \mathbf{n}$-matrix over $\mathbb{K}[X_1, \ldots, X_v]$ and W be a constructible set given by polynomials of $\mathbb{K}[X_1, \ldots, X_v]$.

Output: ComprehensiveLU$(\mathbf{A}, W)$ returns a list of tuples $[\mathbf{P}_i, \mathbf{Q}_i, \mathbf{L}_i, \mathbf{U}_i, W_i]$, called *branches*, for $1 \le i \le e$:

1. the W_i's form a partition of W.
2. $\mathbf{A} = \mathbf{P}_i \mathbf{L}_i \mathbf{U}_i \mathbf{Q}_i$ holds at every point of W_i.

Constructible Sets

In a nutshell, a *constructible set* is the solution set of a disjunction of conjunctions of polynomial equations and inequations.

Constructible Sets

In a nutshell, a *constructible set* is the solution set of a disjunction of conjunctions of polynomial equations and inequations.

Details

- $\mathbb{K}[X_1, \ldots, X_n]$: polynomials in $X_1, \ldots, X_n$ over the field $\mathbb{K}$ [4].

Constructible Sets

In a nutshell, a *constructible set* is the solution set of a disjunction of conjunctions of polynomial equations and inequations.

Details

- $\mathbb{K}[X_1, \ldots, X_n]$: polynomials in $X_1, \ldots, X_n$ over the field $\mathbb{K}$ [4].
- A *regular system* of $\mathbb{K}[X_1, \ldots, X_n]$ is a pair $S = [T, h]$ where $T \subset \mathbb{K}[X_1, \ldots, X_n]$ is a regular chain and $h \in \mathbb{K}[X_1, \ldots, X_n]$.

Constructible Sets

In a nutshell, a *constructible set* is the solution set of a disjunction of conjunctions of polynomial equations and inequations.

Details

- $\mathbb{K}[X_1, \ldots, X_n]$: polynomials in $X_1, \ldots, X_n$ over the field $\mathbb{K}$ [4].
- A *regular system* of $\mathbb{K}[X_1, \ldots, X_n]$ is a pair $S = [T, h]$ where $T \subset \mathbb{K}[X_1, \ldots, X_n]$ is a regular chain and $h \in \mathbb{K}[X_1, \ldots, X_n]$.
- The zero set $Z(S)$ of S is the subset of the "zero set" $W(T)$ of T where h does not vanish.

Constructible Sets

In a nutshell, a *constructible set* is the solution set of a disjunction of conjunctions of polynomial equations and inequations.

Details

- $\mathbb{K}[X_1, \ldots, X_n]$: polynomials in $X_1, \ldots, X_n$ over the field $\mathbb{K}$ [4].
- A *regular system* of $\mathbb{K}[X_1, \ldots, X_n]$ is a pair $S = [T, h]$ where $T \subset \mathbb{K}[X_1, \ldots, X_n]$ is a regular chain and $h \in \mathbb{K}[X_1, \ldots, X_n]$.
- The zero set $Z(S)$ of S is the subset of the "zero set" $W(T)$ of T where h does not vanish.
- For every constructible set C one can compute regular systems $S_1, \ldots, S_e$ so that $C = Z(S_1) \cup \cdots \cup Z(S_e)$.

Constructible Sets

In a nutshell, a *constructible set* is the solution set of a disjunction of conjunctions of polynomial equations and inequations.

Details

- $\mathbb{K}[X_1, \ldots, X_n]$: polynomials in $X_1, \ldots, X_n$ over the field $\mathbb{K}$ [4].
- A *regular system* of $\mathbb{K}[X_1, \ldots, X_n]$ is a pair $S = [T, h]$ where $T \subset \mathbb{K}[X_1, \ldots, X_n]$ is a regular chain and $h \in \mathbb{K}[X_1, \ldots, X_n]$.
- The zero set $Z(S)$ of S is the subset of the "zero set" $W(T)$ of T where h does not vanish.
- For every constructible set C one can compute regular systems $S_1, \ldots, S_e$ so that $C = Z(S_1) \cup \cdots \cup Z(S_e)$.
- Given two constructible sets C_1, C_2 represented by regular systems, one can deduce a regular system representation for the sets $C_1 \setminus C_2$ and $C_1 \cap C_2$ and $C_1 \cup C_2$.

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination
  - algebraic simplication.

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination
  - algebraic simplication.
- This is of great importance in polynomial system solving especially for resultant matrices (Sylvester, Dixon, Caley, etc.)

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination
  - algebraic simplication.
- This is of great importance in polynomial system solving especially for resultant matrices (Sylvester, Dixon, Caley, etc.)
- Often the variables of these resultant matrices satisfy constraints (possibly given by constructible sets)

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination
  - algebraic simplication.
- This is of great importance in polynomial system solving especially for resultant matrices (Sylvester, Dixon, Caley, etc.)
- Often the variables of these resultant matrices satisfy constraints (possibly given by constructible sets)
- This leads to the following research questions:

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination
  - algebraic simplication.
- This is of great importance in polynomial system solving especially for resultant matrices (Sylvester, Dixon, Caley, etc.)
- Often the variables of these resultant matrices satisfy constraints (possibly given by constructible sets)
- This leads to the following research questions:
  - When should we simplify the intermediate matrices against those constraints? At each step? Only in the final step?

The motivations of this paper

- To develop and implement LU decomposition of polynomial matrices, one may consider:
  - fraction-free elimination
  - algebraic simplication.
- This is of great importance in polynomial system solving especially for resultant matrices (Sylvester, Dixon, Caley, etc.)
- Often the variables of these resultant matrices satisfy constraints (possibly given by constructible sets)
- This leads to the following research questions:
  - When should we simplify the intermediate matrices against those constraints? At each step? Only in the final step?
  - Should we use these simplifications in conjunction with fraction-free techniques or not?

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination
  - with or without intermediate simplifications

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination
  - with or without intermediate simplifications

  against the baseline method of our SYNASC 2024 paper.
- We evaluated their performance based on key computational metrics:

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination
  - with or without intermediate simplifications

  against the baseline method of our SYNASC 2024 paper.
- We evaluated their performance based on key computational metrics:
  - running time,

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination
  - with or without intermediate simplifications

  against the baseline method of our SYNASC 2024 paper.
- We evaluated their performance based on key computational metrics:
  - running time,
  - number of case branches generated,

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination
  - with or without intermediate simplifications

  against the baseline method of our SYNASC 2024 paper.
- We evaluated their performance based on key computational metrics:
  - running time,
  - number of case branches generated,
  - output size (measures in ASCII characters),

The contributions of this paper

- We conducted a systematic comparative experimentation of variants of comprehensive LU:
  - with or without fraction-free elimination
  - with or without intermediate simplifications

  against the baseline method of our SYNASC 2024 paper.
- We evaluated their performance based on key computational metrics:
  - running time,
  - number of case branches generated,
  - output size (measures in ASCII characters),
  - dimension and degree of constructible sets.

Review of Bareiss Algorithm: pseudo-code

Recall that Bareiss algorithm uses (exact) division only to keep the intermediate entries smaller.

Review of Bareiss Algorithm: pseudo-code

Recall that Bareiss algorithm uses (exact) division only to keep the intermediate entries smaller. For simplicity of presentation, the pseudo-code below assumes all leading principal minors of A are non-zero

Review of Bareiss Algorithm: pseudo-code

Recall that Bareiss algorithm uses (exact) division only to keep the intermediate entries smaller. For simplicity of presentation, the pseudo-code below assumes all leading principal minors of A are non-zero

Algorithm 3 Compute leading principal minors in-place

Input: A — an $n \times n$ matrix with all leading principal minors $[A]_{k,k} \neq 0$

$A_{0,0} \leftarrow 1$

for $k \leftarrow 1$ to $n-1$ **do**

  for $i \leftarrow k+1$ to n **do**

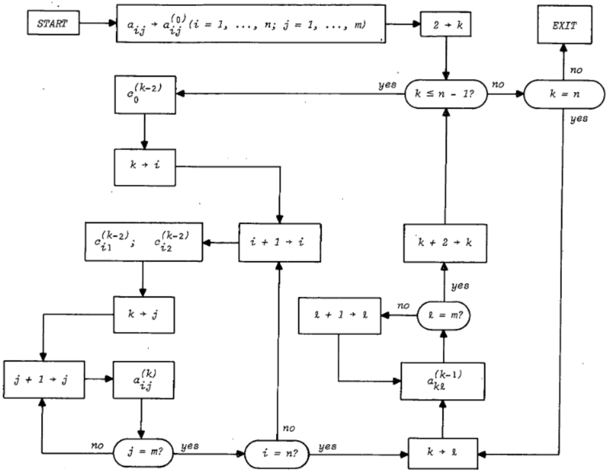
    for $j \leftarrow k+1$ to n **do**

      $A_{i,j} \leftarrow \dfrac{A_{i,j}A_{k,k} - A_{i,k}A_{k,j}}{A_{k-1,k-1}}$ ▷ exact division

    $A_{i,k} \leftarrow 0$

Output: Modified A with $A_{k,k} = [A]_{k,k}$, and $A_{n,n} = \det(A)$

Review of Bareiss Algorithm: Bareiss flow-chart



Fraction-Free LU Factorization

The fraction-free LU form is given by:

$$PA = LD^{-1}U, \quad \text{where}$$

1. the matrices L, D, U are constructed from the determinants of the submatrices $A_{i,j}^{(k)}$ generated during Bareiss algorithm [5], and
2. P and Q are the row and column permutation matrices, respectively.

Fraction-Free LU Factorization

The fraction-free LU form is given by:

$$PA = LD^{-1}U, \quad \text{where}$$

1. the matrices L, D, U are constructed from the determinants of the submatrices $A_{i,j}^{(k)}$ generated during Bareiss algorithm [5], and
2. P and Q are the row and column permutation matrices, respectively.

$$L = \begin{bmatrix} A_{1,1}^{(0)} & & & \\ A_{2,1}^{(0)} & A_{2,2}^{(1)} & & \\ \vdots & \vdots & \ddots & \\ A_{n,1}^{(0)} & A_{n,2}^{(1)} & \cdots & A_{n,n}^{(n-1)} \end{bmatrix}, \quad U = \begin{bmatrix} A_{1,1}^{(0)} & A_{1,2}^{(0)} & \cdots & A_{1,m}^{(0)} \\ & A_{2,2}^{(1)} & \cdots & A_{2,m}^{(1)} \\ & & \ddots & \vdots \\ & & & A_{n,m}^{(n-1)} \end{bmatrix}$$

$$D = \begin{bmatrix} A_{1,1}^{(0)} & 0 & \cdots & 0 \\ 0 & A_{1,1}^{(0)}A_{2,2}^{(1)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A_{n-1,n-1}^{(n-2)}A_{n,n}^{(n-1)} \end{bmatrix}$$

Specification

Two strategies for simplifying our matrix entries w.r.t. constructible sets.

$$A \in \mathscr{M}_{m\times n}(\mathbb{K}[\underline{x}]),\ \ W \subseteq \mathbb{K}^n$$

$$[P_i, L_i, U_i, D_i, Q_i, W_i]$$

Specification

Two strategies for simplifying our matrix entries w.r.t. constructible sets.

$$A \in \mathscr{M}_{m\times n}(\mathbb{K}[\underline{x}]), \;\; W \subseteq \mathbb{K}^n$$

$$[P_i, L_i, U_i, D_i, Q_i, W_i]$$

- Strategy 1:

Specification

Two strategies for simplifying our matrix entries w.r.t. constructible sets.

$$A \in \mathscr{M}_{m \times n}(\mathbb{K}[\underline{x}]), \;\; W \subseteq \mathbb{K}^n$$

1: Bareiss PLU over $\mathbb{K}[\underline{x}]$[1]

$$[P_i, L_i, U_i, D_i, Q_i, W_i]$$

- Strategy 1:
  1: W is only used to select pivots that are not zero-divisors.

Specification

Two strategies for simplifying our matrix entries w.r.t. constructible sets.

$$A \in \mathscr{M}_{m\times n}(\mathbb{K}[\underline{x}]),\ \ W \subseteq \mathbb{K}^n$$

1: Bareiss PLU over $\mathbb{K}[\underline{x}]^1$

$$[P_i, L_i, U_i, D_i, Q_i, W_i] \longrightarrow \text{Result}$$

2: Simplification modulo W_i

- Strategy 1:
  1. W is only used to select pivots that are not zero-divisors.
  2. The entries of the matrices L_i, D_i, U_i are simplified w.r.t. W_i after the whole fraction-free comprehensive LU is completed.

Specification

Two strategies for simplifying our matrix entries w.r.t. constructible sets.

$$A \in \mathscr{M}_{m\times n}(\mathbb{K}[\underline{x}]),\ \ W \subseteq \mathbb{K}^n$$

1: Bareiss PLU over $\mathbb{K}[\underline{x}]^1$ (↙)

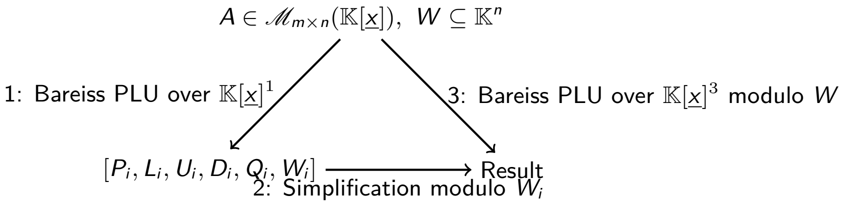
$$[P_i, L_i, U_i, D_i, Q_i, W_i] \longrightarrow \text{Result}$$

2: Simplification modulo W_i

- Strategy 1:
  1: W is only used to select pivots that are not zero-divisors.
  2: The entries of the matrices L_i, D_i, U_i are simplified w.r.t. W_i after the whole fraction-free comprehensive LU is completed.
- Strategy 2:

Specification

Two strategies for simplifying our matrix entries w.r.t. constructible sets.



- Strategy 1:
  1: W is only used to select pivots that are not zero-divisors.
  2: The entries of the matrices L_i, D_i, U_i are simplified w.r.t. W_i after the whole fraction-free comprehensive LU is completed.
- Strategy 2:
  3: The entries of the intermediate L_i, D_i, U_i are simplified w.r.t. W_i (after reach row reduction and after each exact division).

Details of the simplification

When we set the `simplified` flag to `true`, the following occur at each branch of the computation:

1. We "deconstruct" the constructible set Wi into a list of regular systems $[[T1, H1], ..., [Tk, Hk]]$.

Details of the simplification

When we set the `simplified` flag to `true`, the following occur at each branch of the computation:

1. We "deconstruct" the constructible set Wi into a list of regular systems $[[T1, H1], ..., [Tk, Hk]]$.
2. We make sure that each regular chain T_j is normalized (i.e, a Gröbner basis) which may split the computations.

Details of the simplification

When we set the `simplified` flag to `true`, the following occur at each branch of the computation:

1. We "deconstruct" the constructible set Wi into a list of regular systems $[[T1, H1], ..., [Tk, Hk]]$.
2. We make sure that each regular chain T_j is normalized (i.e, a Gröbner basis) which may split the computations.
3. We compute the normal form of each matrix L_i, D_i, U_i w.r.t. each T_j.

Details of the simplification

When we set the `simplified` flag to `true`, the following occur at each branch of the computation:

1. We "deconstruct" the constructible set Wi into a list of regular systems $[[T1, H1], ..., [Tk, Hk]]$.
2. We make sure that each regular chain T_j is normalized (i.e, a Gröbner basis) which may split the computations.
3. We compute the normal form of each matrix L_i, D_i, U_i w.r.t. each T_j.

Details of the simplification

When we set the `simplified` flag to `true`, the following occur at each branch of the computation:

1. We "deconstruct" the constructible set Wi into a list of regular systems $[[T1, H1], ..., [Tk, Hk]]$.
2. We make sure that each regular chain T_j is normalized (i.e, a Gröbner basis) which may split the computations.
3. We compute the normal form of each matrix L_i, D_i, U_i w.r.t. each T_j.
4. To perform an exact division $\frac{a}{b} \mod T_j$ we compute the inverse of b w.r.t. T_j, which may split the computations.

We implemented the algorithms in MAPLE The algorithms output a list of cases along with their corresponding constraints. To illustrate these methods, we selected examples from various papers, and ran these examples over $\mathbb{K}[X_1, \ldots, X_v]$ with initial with initial constraints in MAPLE 2024 using an `HP` Pavilion x360 PC with `Windows` 11.

Examples: same as those from our SYNASC 2024 paper plus:

- Sylvester matrices with input constraints hence representative of sub-resultant chain computations (E_{14}, E_{17})
- Vandermonde matrices, hence representative of evaluation-interpolation problems.(E_{20}, E_{24})
- Toeplitz matrices, since LU decomposition gives a quick method for computing the determinant of such matrices (E_{23})

Algorithmic Variants Compared:

Variant	Fraction-Free	Simplification
CLU	No	No
SCLU	No	Yes
FFLU	Yes	No
SFFLU	Yes	Yes

Performance comparison: Time, number of cases(branches), output size

Example	CLU	SCLU	FFLU	SFFLU	Best Performer
E_5	**0.38s** 7 2406	0.46s 12 3525	0.52s 9 3532	0.46s 9 2846	**CLU**
E_{14}	**0.23**s 3 1382	0.25s 7 2801	0.25s 7 3595	0.37s 7 3007	**CLU**
E_{17}	10.94s 3 2500	**5.04s** 18 13281	14.08s 15 18253	11.91s 18 19156	**SCLU**
E_{20}	18.59s 4 4637	**0.29s** 4 2037	12.37s 4 16118	1.66s 4 2112	**SCLU**
E_{23}	10.69s 12 9957	**5.03s** 20 9647	11.64s 22 22584	6.42s 20 10269	**SCLU**
E_{24}	20.46s 2 3386	**0.44s** 2 2810	886.11s 2 62673	7.01s 2 2949	**SCLU**
E_4	0.39s 9 2121	0.39s 11 2393	0.41s 11 3013	**0.35s** 9 2163	**SFFLU**

Conclusions

- No single variant dominates across all test cases.
- For most simple test cases (less than a 1 sec for all methods) CLU (that is, no simplification, no fraction-free) works best.
- For most harder test cases, SCLU (simplification, but no fraction-free) works best (time and size).
- For some of these harder test cases, CLU alone does better than FFCLU (fraction-free, but no simplification)
- For all harder test cases, SFFLU (simplification + fraction-free) comes second.
- Our results illustrate the fact that
  1. using fraction-free methods is not always helpful (think of subresultant chain with no defective subresultants),
  2. it is hard to build examples where systematic simplification is a very bad idea.

[1] F. C. Moon, *Chaotic Vibrations: An Introduction for Applied Scientists and Engineers.*

Spherical Pendulum, 1987.

[2] R. Corless, M. Moreno Maza, and S. E. Thornton, "Jordan Canonical Form with Parameters from Frobenius Form with Parameters," *Mathematical Aspects of Computer and Information Sciences*, 2017.

[3] A. Olagunju, M. Maza, and D. Jeffrey, "Comprehensive LU Decomposition and True Path," *SYNASC*, pp. 17–24, 09 2024.

[4] P. Aubry, D. Lazard, and M. Moreno Maza, "On the theories of triangular sets," *J. Symbolic Computation*, vol. 28, pp. 105–124, 1999.

[5] W. Zhou and D. Jeffrey, "Fraction-free matrix factors: New forms for lu and qr factors," *Frontiers of Computer Science in China*, vol. 2, pp. 67–80, 03 2008.