Comprehensive *LU* factors of polynomial matrices

A.C. Camargos Couto[1][0000−0002−1252−2880], M. Moreno Maza[2,3], D. Linder[4], D.J. Jeffrey[3][0000−0002−2161−6803], and R. M. Corless[3][0000−0003−0515−1572]

[1] ORCCA, University of Western Ontario, London ON, Canada
[2] Maplesoft, Waterloo, Ontario, Canada

Abstract. The comprehensive LU decomposition of a parametric matrix consists of a case analysis of the LU factors for each specialization of the parameters. Special cases can be discontinuous with respect to the parameters, the discontinuities being triggered by zero pivots encountered during factorization. For polynomial matrices, we describe an implementation of comprehensive LU decomposition in Maple, using the `RegularChains` package.

Keywords: Parametric linear algebra · LU decomposition · Regular chains.

1 Introduction

Decomposing a matrix A into lower and upper triangular factors L and U is one of the fundamental operations in linear algebra. It is implemented in Maple's `LinearAlgebra` package as `LUDecomposition`. For polynomial matrices, the function takes the usual Computer Algebra option of returning only a generic factorization. Thus, for example,

$$A_1 = \begin{bmatrix} 1-x & 2 & 3 \\ 2-x & 5 & 6 \\ x & 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{x-2}{x-1} & 1 & 0 \\ \frac{-x}{x-1} & \frac{5x-3}{3x-1} & 1 \end{bmatrix} \begin{bmatrix} 1-x & 2 & 3 \\ 0 & \frac{3x-1}{x-1} & \frac{3x}{x-1} \\ 0 & 0 & -\frac{2}{3x-1} \end{bmatrix}. \tag{1}$$

The special cases $x = 1, 3/2$ make the elements singular. The importance of recomputing singular cases is established in [5], and, in the context of differential elimination, in [7]. We remark that special cases for LU factoring do not always occur when pivots are zero, because sometimes alternative pivots can lead to the same factoring. Indeed special cases can be *exactly detected* by Maple's existing `LUDecomposition` function through a special syntax implementing the algorithm of [5], which is not the default because its output is not just a simple answer, as we discuss below. See [8] for an example of the syntax. Because re-computation is necessary in those special cases by that method, which allows comprehensive computation but is not itself comprehensive, we do not directly compare our present implementation to that syntax.

Symbolic computing in the presence of parameters has been the subject of discussion over many years [1]. Early systems, such as Macsyma, often asked a user interactively for information regarding a parameter, while other approaches used provisos, case analyses, error messages, etc. An important distinction is that between *comprehensive* approaches and *generic* approaches. In a comprehensive approach, a system will attempt to identify and compute all possible special cases, in contrast to a generic approach which selects one expression, implying conditions (which may not be stated) on the parameters.

Comprehensive solutions have been defined and used in several areas of mathematics. In algebraic geometry, a comprehensive Gröbner Basis was defined in [6], and a comprehensive triangular system based on regular chains was defined in [9]. In the MAPLE package `DEtools`, the `rifsimp` program offers a `casesplit` option, which is equivalent to a comprehensive analysis. A comprehensive solution of linear systems was presented in [14]. Computer Algebra systems have tended to avoid comprehensive results for several reasons. First, there is the difficulty of continuing a computation using a comprehensive result; secondly, there has been a fear that the number of cases will multiply exponentially and overwhelm the system. Although this could happen, there are many problems for which a comprehensive solution is possible and desirable.

2 Preliminaries

The implementation is based on MAPLE's `RegularChains` library, which we briefly describe in this section. The notion of a *regular chain*, introduced independently in [2] and [4], is closely related to that of a triangular decomposition of a polynomial system. Broadly speaking, a *triangular decomposition* of a polynomial system S is a set of simpler (in a precise sense) polynomial systems $S_1, \ldots, S_e$ such that a point p is a solution of S if, and only if, p is a solution of (at least) one of the systems $S_1, \ldots, S_e$.

If one wishes to describe all the solutions of S, those simpler systems are required to be regular chains. We refer to [3, 10] for a formal presentation on the concepts of a regular chain.

Multivariate polynomials. Let $\mathbb{K}$ be a field. If $\mathbb{K}$ is an ordered field, then we assume that it is a real closed field such as the field $\mathbb{R}$ of real numbers. Otherwise, we assume that $\mathbb{K}$ is algebraically closed, like the field $\mathbb{C}$ of complex numbers. Let $X_1 < \cdots < X_s$ be $s \geqslant 1$ ordered variables. We denote by $\mathbb{K}[X_1, \ldots, X_s]$ the ring of polynomials in the variables $X_1, \ldots, X_s$ with coefficients in $\mathbb{K}$. For a non-constant polynomial $p \in \mathbb{K}[X_1, \ldots, X_s]$, the greatest variable in p is called the *main variable* of p, denoted by $\mathrm{mvar}(p)$, and the leading coefficient of p w.r.t. $\mathrm{mvar}(p)$ is called the *initial* of p, denoted by $\mathrm{init}(p)$.

Regular chains. A set R of non-constant polynomials in $\mathbb{K}[X_1, \ldots, X_s]$ is called a *triangular set*, if for all $p, q \in R$ with $p \neq q$ we have $\mathrm{mvar}(p) \neq \mathrm{mvar}(p)$. A variable X_i is said to be *free* w.r.t. R if there exists no $p \in R$ such that $\mathrm{mvar}(p)=X_i$. For a nonempty triangular set R, we define the *saturated ideal* $\mathrm{sat}(R)$ of R to be the $\mathrm{ideal}(R):h_R^\infty$, where h_R is the product of the initials of the

polynomials in R. The saturated ideal of the empty triangular set is defined as the trivial ideal $\langle 0 \rangle$. From now on, R denotes a triangular set of $\mathbb{K}[X_1, \ldots, X_s]$. The ideal $\mathrm{sat}(R)$ has several properties, and in particular it is unmixed [11]. We denote its height, that is, the number of polynomials in R, by e, thus $\mathrm{sat}(R)$ has dimension $s - e$. Let $X_{i_1} < \cdots < X_{i_e}$ be the main variables of the polynomials in R. We denote by r_j the polynomial of R whose main variable is X_{i_j} and by h_j the initial of r_j. Thus h_R is the product $h_1 \cdots h_e$. We say that R is a *regular chain* whenever R is empty or, $\{r_1, \ldots, r_{e-1}\}$ is a regular chain and h_e is regular modulo the saturated ideal $\mathrm{sat}(\{r_1, \ldots, r_{e-1}\})$.

Constructible sets. Let $F \subset \mathbb{K}[X_1, \ldots, X_s]$ be a set of polynomials and $g \in \mathbb{K}[X_1, \ldots, X_s]$ be a polynomial. We denote by $V(F) \subseteq \mathbb{K}^s$ the *zero set* or *affine variety* of F, that is, the set of points in the affine space $\mathbb{K}^s$ at which every polynomial $f \in F$ vanishes. If F consists of a single polynomial f, we write $V(f)$ instead of $V(F)$. We call a *constructible set* any subset of $\mathbb{K}^s$ of the form $V(F) \backslash V(g)$. Let $R \subset \mathbb{K}[X_1, \ldots, X_s]$ be a regular chain and let $h \in \mathbb{K}[X_1, \ldots, X_s]$ be a polynomial. We say that the pair $[R, h]$ is a *regular system* whenever h is regular modulo $\mathrm{sat}(R)$ and $V(h_R) \subseteq V(h)$ holds. We write $Z(R, h)$ for $V(R) \backslash V(h)$. One should observe that for a regular system $[R, h]$ the zero set $Z(R, h)$ is necessarily not empty. Regular systems provide an encoding for constructible sets. More precisely, there exists a finite family $\mathcal{T}$ of regular systems $[R_1, h_1], \ldots, [R_e, h_e]$ of $\mathbb{K}[X_1, \ldots, X_s]$ such that

$$V(F) \backslash V(g) \;=\; Z(R_1, h_1) \;\cup\; \cdots \;\cup\; Z(R_e, h_e).$$

We call $\mathcal{T}$ a *triangular decomposition* of the constructible set $V(F) \backslash V(g)$. Encoding constructible sets with regular systems has another benefit. It leads to efficient algorithms for performing set-theoretic operations on constructible sets; see [9]. These operations, as well as the above mentioned triangular decomposition algorithms, are part of the `RegularChains` library [12, 13] distributed with the Maple CAS.

3 Comprehensive LU method

We consider the LU factoring of matrices with multivariate polynomial entries, using partial pivoting. The pivots are analysed with the `RegularChains` library in Maple. Care is taken to identify cases where zero pivots do not, after all, lead to distinct LU factors. Considering that constructible sets represent the solution set of a polynomial, if there exists cases where the pivot is zero in a step of LU decomposition, constructible sets are used to represent their equations. Subsequently, these equations are used to express the constraints of validity of each solution branch (e.g.: $x = 3/2$ for the example shown in section 1)

For the decomposition to be comprehensive (i.e., span all possible scenarios), we need to conduct the row reductions on all possible *unique* cases that may arise. Therefore, at each step, a pivot's constructible sets are analysed. Let CS_1 express the solution set of $pivot \neq 0$. This constructible set can be built with the `GeneralConstruct` command from the `RegularChains` library in Maple :

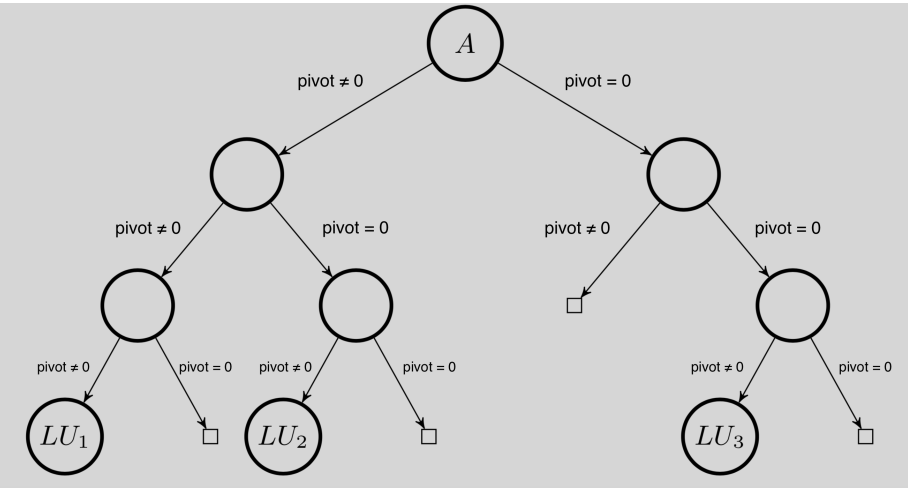

Fig. 1. Steps towards the comprehensive solution. Each root-to-leaf path represents a distinct LU decomposition of A. On each step, the calculation is split between two potential branches. Square nodes represents non-unique, and therefore dropped, cases.

```
>> GeneralConstruct([],[A(k,k)],R);
```

Where the first and second arguments express equations and inequations to build the constructible set from, and the third argument is a polynomial ring. In order to build CS_1, we would give `GeneralConstruct` one inequation that represents the condition $pivot \neq 0$, and no equations. If CS_1 is nonempty, there are cases in which the natural matrix pivot can be used for the row reduction operation; so this operation is recorded in a branch ($pivot \neq 0$ in Figure 1). Let CS_2 express the solution set of an inequation $pivot = 0$. Similarly to the other case, CS_2 can be built with the `GeneralConstruct` command:

```
>> GeneralConstruct([A(k,k)],[],R);
```

However, in this case, an equation $pivot = 0$ is passed as argument to the function, and no inequations are used. If CS_2 is non-empty, we look for an alternative pivot in the same column that has an empty CS_2 (this way guaranteeing that there will be no cases where division by zero is possible). The alternative pivot is used to build the permutation matrix and the original CS_2 value is saved, so that we can keep track of the exception case conditions. The alternative operation is recorded in a second branch ($pivot = 0$ in Figure 1).

This process is repeated iteratively on each step of the LU factoring, every time splitting the result in two possible cases, and this way forming an incomplete binary tree (incomplete because we only keep the unique leaves). The result is a group of solutions and their constraints, where the joining of all solution's constructible sets form a partition of the variable domain space.

4 Implementation in Maple

We have written a Maple procedure `ComprehensiveLU(A,R,opt)` to implement the method described. The arguments are A, a square matrix with polynomial elements, R, a descriptor of the polynomial ring containing the elements (the procedure `PolynomialRing` in the `RegularChains` library), and `opt`, to select different displays of the results. The results are returned as a list of lists. Each list consists of a factoring (P, L, U), and a constructible set, specifying the conditions. It is our intention to add the `ComprehensiveLU` procedure as part of the `LinearAlgebra` library from MAPLE in the future.

The options available for the printing of conditions are constructible sets (the default), prettyprinting and programmable. To present some examples below, we have unpacked the output using the prettyprinting option, for easier reading. We have also confined our examples to small matrices with only a few polynomial entries. In each example, the first case corresponds to the generic result, and equals the result returned by the Maple command `LinearAlgebra[LUDecomposition]`.

We return to the introductory example (1).

$$\begin{bmatrix} 1-x & 2 & 3 \\ 2-x & 5 & 6 \\ x & 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{x-2}{x-1} & 1 & 0 \\ \frac{-x}{x-1} & \frac{5x-3}{3x-1} & 1 \end{bmatrix} \begin{bmatrix} 1-x & 2 & 3 \\ 0 & \frac{3x-1}{x-1} & \frac{3x}{x-1} \\ 0 & 0 & -\frac{2}{3x-1} \end{bmatrix}, \begin{cases} x-1 \neq 0 \ , \\ 3x-1 \neq 0 \ . \end{cases} \tag{2}$$

The permutation matrix is I and is omitted. The two conditions are not returned by Maple. The special cases are

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 5/2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2/3 & 2 & 3 \\ 0 & 2 & -1/2 \\ 0 & 0 & -3/2 \end{bmatrix} \text{ , when } \{ 3x-1=0 \ .$$

and

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 \\ 0 & 2 & 4 \\ 0 & 0 & -1 \end{bmatrix} \text{ , when } \{ x-1=0 \ .$$

A multivariate example shows how the number of conditions increases as the number of parameters increases.

$$A = \begin{bmatrix} a & 2b & 3 \\ d & -2 & 6 \\ 7 & 3 & 2 \end{bmatrix} \ . \tag{3}$$

The generic case (also returned by Maple without conditions) is

$$A = \begin{bmatrix} 1 & 0 & 0 \\ \frac{d}{a} & 1 & 0 \\ \frac{7}{a} & \frac{-3a+14b}{2bd+2a} & 1 \end{bmatrix} \begin{bmatrix} a & 2b & 3 \\ 0 & \frac{-2bd-2a}{a} & \frac{-3d+6a}{a} \\ 0 & 0 & \frac{(4d-84)b+22a-9d-42}{2bd+2a} \end{bmatrix}, \begin{cases} a \neq 0 \ , \\ a+bd \neq 0 \ . \end{cases} \tag{4}$$

There are 3 special cases, and it is interesting to note that they uncover additional constraints.

$$A = \begin{bmatrix} 1\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \end{bmatrix} \begin{bmatrix} 1 & 0\,0 \\ \frac{-7}{bd} & 1\,0 \\ -1/b & 0\,1 \end{bmatrix} \begin{bmatrix} -bd & 2b & 3 \\ 0 & \frac{3d+14}{d} & \frac{2bd+21}{bd} \\ 0 & 0 & \frac{6b+3}{b} \end{bmatrix}, \begin{cases} a + bd = 0 \\ b \neq 0 \\ d \neq 0 \end{cases} \tag{5}$$

The second case is

$$A = \begin{bmatrix} 0\,0\,1 \\ 0\,1\,0 \\ 1\,0\,0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ d/7 & 1 & 0 \\ 0 & \frac{-14b}{3d+14} & 1 \end{bmatrix} \begin{bmatrix} 7 & 3 & 2 \\ 0 & -\frac{3d}{7} - 2 & -\frac{2d}{7} + 6 \\ 0 & 0 & \frac{-4bd+84b+9d+42}{3d+14} \end{bmatrix}, \begin{cases} a = 0 \\ 3d + 14 \neq 0 \end{cases} \tag{6}$$

Lastly,

$$A = \begin{bmatrix} 0\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \end{bmatrix} \begin{bmatrix} 1 & 0\,0 \\ 0 & 1\,0 \\ -2/3 & 0\,1 \end{bmatrix} \begin{bmatrix} 7 & 3 & 2 \\ 0 & 2b & 3 \\ 0 & 0 & 22/3 \end{bmatrix}, \begin{cases} a = 0\,, \\ 3d + 14 = 0 \end{cases} \tag{7}$$

4.1 Efficiency

The implementation uses the `RegularChains` library, which is more efficient at performing polynomial arithmetic than the older `LUDecomposition` procedure. In order to perform comparison tests, we created a set of input matrices with polynomial elements up to degree 5, and measured the computation time for the LU factoring of each configuration. To ensure a fair efficiency analysis, the comparison test restricts the `ComprehensiveLU` program to computing only the generic case, in order to keep it comparable with the MAPLE library. The results are shown in Figure 2. See below the script for the efficiency comparison test:

```
cf := proc(d) randpoly([x_1, x_2, x_3], dense, degree = d); end;
t1 := []: t2 := []: xd := []:
for d from 1 to 2 do      # coeff. degree
  m := 3; n:= 3;   ## order of the matrix
  xd := [op(xd),d];
  A := Matrix([[cf(d), cf(d), cf(d)], [1, cf(d), cf(d)],
  [cf(d), 2, 4]]):
  R := PolynomialRing([x_3, x_2, x_1]):
  t_1 :=  x_1^n + x_1 + 1;   t_2 :=  x_2^2  - x_1 - 1;
  t_3 :=  x_3^2 - x_2 - 1;
  cs := GeneralConstruct([t_1, t_2, t_3], [], R):
  t := time(): ComprehensiveLU(A, R, cs);
  t1 := [op(t1),time() - t]:
```

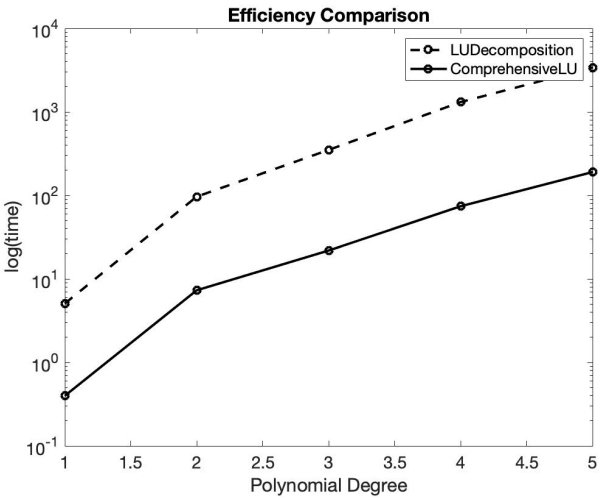

Fig. 2. Computation time vs polynomial degree for LUDecomposition of Maple and the present ComprehensiveLU. Recursion levels were introduced, with ComprehensiveLU being restricted to computing only the generic case.

```
  printf("with \%g degree polynomials, ComprehensiveLU took
      \%g seconds to compute the result \n",d, time() - t);
  a_1:=RootOf(t_1, x_1): a_2:=RootOf(y^2 - a_1 + 1, y):
  a_3:=RootOf(z^2 - a_2 + 1, z):
  B := eval(A, [x_1 = a_1, x_2 = a_2, x_3 = a_3]);
  t := time():   LUDecomposition(B);
  t2 := [op(t2),time() - t]:
  printf("with \%g degree polynomials, LUDecomposition took
      \%g seconds to compute the result \n",d, time() - t);
end do:
```

The experiment consists of an LU factoring of 3×3 matrices with random polynomials. In order to make the computations algebraically challenging, recurrence levels were established to define the polynomial variables, which obey additional polynomial relationships with highest degree equal to 3. The experiment script loops over values of the random polynomial degree d ranging from 1 to 5 and records the time it took both algorithms to compute the final result in each iteration. The plot in figure 2 illustrates the comparison findings.

5 Conclusion

In parametric linear algebra, LU decomposition can be a discontinuous operation if the pivots encountered throughout the factorization are polynomials with roots

defined in the problem's domain space. The discontinuity equations define special cases that we carefully consider in this project.

Our aim is to provide a comprehensive tool for computing LU factors of parametric matrices in MAPLE. We have shown that the main existing procedure in Maple's `LinearAlgebra` library, `LUDecomposition`, can only decompose generic cases of parametric matrices in an explicit way. Therefore, our algorithmic procedure `ComprehensiveLU` can be seen as a complement to the existing library function.

References

1. Corless, R.M., Jeffrey, D.J.: Well... it isn't quite that simple. SIGSAM Bulletin, **26**(3), 2–6 (1992).
2. M. Kalkbrener: Three contributions to elimination theory. Johannes Kepler University, Linz, (1991).
3. P. Aubry and D. Lazard and Moreno Maza, M.: On the Theories of Triangular Sets. J. Symb. Comp., **28**(1-2), 105-124, (1999).
4. L. Yang and J. Zhang: Searching dependency between algebraic equations: an algorithm applied to automated reasoning. International Atomic Energy Agency, IC/89/263, Miramare, Trieste, Italy, (1991).
5. Corless, R.M., Jeffrey, D.J.: The Turing factorization of a rectangular matrix. SIGSAM Bulletin, **31**(3), 20–30 (1997).
6. Weispfenning, V.: Comprehensive Grobner Bases. J. Symbolic Computation, **14**, 1–29 (1992).
7. Reid, G.: Algorithms for reducing a system of PDEs to standard form, determining the dimension of its solution space and calculating its Taylor series solution. European J. Appl. Math. **2**, 293–318 (1991).
8. Jeffrey, D.J., Corless R.M.: Linear Algebra in Maple, Chapter 89 in the CRC Handbook of Linear Algebra, Leslie Hogben, ed, Chapman & Hall/CRC, 2nd ed (2013)
9. Chen, C., Golubitsky, O., Lemaire, F., Moreno Maza, M., Pan, W.: Comprehensive Triangular Decomposition. In: Ganzha V.G., Mayr E.W., Vorozhtsov E.V. (eds) Computer Algebra in Scientific Computing. CASC 2007. Lecture Notes in Computer Science, vol 4770. Springer, Berlin, Heidelberg (2007).
10. Changbo Chen and Moreno Maza, M.: Algorithms for computing triangular decomposition of polynomial systems. J. Symb. Comput., **47**(6), 610-642 (2012).
11. F. Boulier, Lemaire, F. and Moreno Maza, M.: Well Known Theorems on Triangular Systems and the D5 Principle. Proceedings of Transgressive Computing 2006, J.-G. Dumas et al., editors, Granada, Spain, (2006)
12. Chen, C., Davenport, J.H., Lemaire, F., Moreno Maza, M., Phisanbut, N., Xia, B., Xiao, R., Xie, Y.: Solving semi-algebraic systems with the RegularChains library in Maple. Proceedings of the Fourth Internationa Conference on Mathematical Aspects of Computer Science and Information Sciences (MACIS 2011), Edited by Stefan Raschau, pp. 38–51, (2011).
13. F. Lemaire and M. Moreno Maza and Y. Xie: The RegularChains Library in Maple 10. Proceedings of Maple Summer Conference'05, Ilias S. Kotsireas editor, Waterloo, Canada, (2005).
14. Sit, W.Y.: An algorithm for solving parametric linear systems. J. Symb. Comp. **13**, 353–394 (1992)