REPORT ON OpenDreamKit DELIVERABLE D5.12

Exact linear algebra algorithms and implementations. Library maintenance and close integration in mathematical software for LINBOX library

CLÉMENT PERNET AND JEAN-GUILLAUME DUMAS

Due on	31/08/2018 (M36)
Delivered on	31/08/2018
Lead	Université Grenoble Alpes (UGA)
Progress on and finalization of this deliverable has been tracked publicly at: https://github.com/OpenDreamKit/OpenDreamKit/issues/110	

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #110 ON 2018-08-31

- **WP5:** High Performance Mathematical Computing
- **Lead Institution:** Universite Grenoble Alpes
- **Due:** 2018-08-31 (month 36)
- **Nature:** Demonstrator
- **Task:** T5.3 (#101) LinBox
- **Proposal:** p. 52

CONTEXT

Computational linear algebra is a key tool delivering high computing throughput to applications requiring large scale compuations. In numerical computing, dealing with floating point arithmetic and approximations, a long history of efforts has lead to the design of a full stack of technology for numerical HPC: from the design of stable and fast algorithms, to their implementation in standardized libraries such as LAPACK and BLAS, and their parallelization on shared memory servers or supercomputers with distributed memory. On the other hand, computational mathematics relies on linear algebra with exact arithmetic, i.e. multiprecision integers and rationals, finite fields, etc. This leads to significant differences in the algorithmic and implementations approaches. Over the last 20 years, a continuous stream of research has improved the exact linear algebra algorithmic and simultaneously, software projects, such as LinBox and fflas-ffpack were created to deliver, a similar set of kernel linear algebra routines as LAPACK but of exact arithmetic.

GOAL OF THE DELIVERABLE

This deliverable aims at taking a major step forward in the advancement of this technology stack for exact linear algebra: the development of new application frameworks, new algorithms, their careful implementation as high performance kernels in a standardized library. As a demonstrator for the usability of this building block for the development of virtual research environment, a key outcome of this deliverable is a tight integration of the libraries LinBox and fflas-ffpack into the software SageMath.

CONTENTS

1. ALGORITHMIC INNOVATIONS

1.1. Dense Gaussian elimination

Gaussian elimination is among the most commonly used computing kernel in both numerical and exact linear algebra, as it is used for solving linear systems, and more specifically in exact linear algebra for computing nullspace basis, rank and rank profiles, determinants, characteristic polynomials, etc.

Rank deficient LU decomposition. Before the start of the project, we had a series of contributions in the development of fast Gaussian elimination algorithms in the context of exact linear algebra, namely

(1) identifying and connecting all triangular matrix decomposition, revealing a crucial invariant, the rank profile, and echelon forms.
(2) proposing block recursive (by slab or tile splitting) algorithms computing these factorization in the best known time complexities, and without additional memory footprint;
(3) introducing a new matrix invariant, the rank profile matrix, summarizing all information on the row and column rank profiles of the matrix and of all of its leading submatrices;
(4) a block recursive and a base case iterative algorithm which implementation in `fflas-ffpack` sets the state of the art in terms of computing efficiency in sequential and parallel multi-threaded contexts, and competes with the efficiency of the equivalent numerical Gaussian elimination routine;
(5) an exhaustive study of the required conditions on the pivoting strategy for a Gaussian elimination algorithm to reveal the rank profile matrix invariant.

A first contribution to this deliverable, published in Dumas et al. [2017d], improves over the previous results on the rank profile matrix in the following ways:

(1) a new probabilistic algorithms to compute the rank profile matrix invariant in $\tilde{O}(r^\omega + mn)$ instead of $O(mnr^{\omega-2})$;
(2) a generalization of the existing algorithms to produce the full row and column echelon form of the matrix
(3) an exhaustive study on how does the notion of rank profile matrix generalizes over arbitrary rings.

Symmetric triangular factorization. When the input matrix is symmetric, a variety of symmetric factorizations and algorithms is known, depending on the ability to extract square roots (for the Cholesky factorization), the rank structure and the pivoting strategy to be used.

If the block iterative algorithms with partial or full pivoting are well studied in numerical linear algebra, the recursive block algorithms were only approached recently and with strong restrictions on the pivoting strategy[1]. We proposed in Dumas and Pernet [2018] (see Appendix A) a generalization of the Aasen algorithm in a block recursive structure, producing a LDLT factorization with partial pivoting. In particular, this algorithm is able to also compute the rank profile matrix invariant and enjoy a reduction of its time complexity to that of matrix multiplication: $O(mnr^{\omega-2})$.

	Gen. rank prof. $r=n$				Gen. rank prof. $r=n$		Random RPM $r=n/2$	
n	`dgetrf`	`dsytrf`	`dsytrf_rk`	`dsytrf_aa`	PLUQ	LDLT	PLUQ	LDLT
100	1.17e-04	1.31e-04	1.37e-04	9.21e-05	4.64e-04	3.23e-04	3.20e-04	3.80e-04
200	3.73e-04	4.39e-04	5.51e-04	4.48e-04	1.87e-03	8.80e-04	1.58e-03	1.33e-03
500	3.31e-03	3.78e-03	4.88e-03	3.87e-03	1.73e-02	5.21e-03	1.88e-02	7.92e-03
1000	2.19e-02	2.09e-02	2.58e-02	2.05e-02	8.84e-02	2.30e-02	6.27e-02	3.18e-02
2000	0.145	0.127	0.154	0.127	0.438	0.127	0.274	0.150
5000	2.005	1.604	1.871	1.598	3.904	1.591	2.431	1.294
10000	14.948	11.981	13.396	12.008	24.115	10.904	14.775	7.894

TABLE 1. Comparing computation time (s) of numerical routines with the symmetric (LDLT) and unsymmetric (PLUQ) triangular decompositions. Matrices with rank r, generic rank profile or rank profile matrix uniformly random.

Table 1 compares the computation time of the finite field symmetric decomposition algorithm with several other routines: the unsymmetric elimination over a finite field (running the PLUQ algorithm of Dumas et al. [2017d]) and the numerical elimination routines in double precision of LAPACK provided by OpenBLAS: `dgetrf` (LU decomposition), and 3 variants of `dsytrf` (symmetric LDLT decomposition with pivoting). These experiments first confirm a speed-up factor of about 2 between the symmetric and unsymmetric case over a finite field, which is the expected gain. On small dimensions, the numercial routines perform best, as the quadratic modular reductions is penalizing the routines over a finite field. However, this is compensated by use of Strassen's algorithm, making our implementation outperform LAPACK's best `dsytrf` for larger dimension, even when the rank profile is not generic.

1.2. Quasiseparable matrices

Exploiting some structure in a matrix to speed up computations is a whole field in linear algebra algorithmic. Quasiseparable matrices are structured by a bounding condition on the rank of any of their submatrices below or above the main diagonal. It is a well studied field in numerical linear algebra, as these matrices occur several major applications, such as solving particle interaction, or generalized eigenvalues problems. In exact linear algebra this class of structured matrices seemed to be absent from the literature and software ecosystem.

We introduced this class to the field in Pernet [2016] and Pernet and Storjohann [2018] (see Appendix A) where we contributed with two new storage formats for these matrices and the related algorithms to compute with. The key innovations there are the following:

(1) the first reduction in time complexity for the basic arithmetic with these matrices to the fast matrix multiplication complexity: $O(ns^{\omega-1})$ where ω is the exponent of matrix multiplication, and s is the order of quasiseparability.

[1] See for instance LAPACK Working Note 294, Dec. 2017, and references therein.

(2) the first flat (i.e. non-hierarchical) compact representation for these matrices reaching the best space and time complexities. This was made possible thanks to a non-trivial connection with the notion of rank profile matrix, which we developed in Dumas et al. [2017d].

1.3. **Outsourced computing security**

A more exploratory aspect of our contribution deals with the design of secure protocols for outsourced or multiparty computations. With the emergence of huge computing infrastructures on the Cloud, large scale computations are likely to no longer be handled in a controlled environment but instead to be delegated to third party infrastructures. This raises several problems regarding the privacy in the data, and the trust in the result.

In this context we have started investigations in two directions: the design of certificates of correctness ensuring trust and of multiparty computation protocols, ensuring privacy.

1.3.1. *Certificates.* In this setting, the provider of computing resources is called a prover and has to convince his client (denoted the verifier) that the result which he computed is correct. Most approaches to interactive certification rely on generic techniques for circuit transformations and are therefore mostly relevant for theoretical result on asymptotic complexities.

Alternatively our approach is to design problem specific certification protocols taking advantage of the algebraic nature of the computation and reducing the use of costly cryptographic primitives. A first instance, for matrix vector products is proposed in Dumas and Zucca [2017].

In Dumas et al. [2017c] (see Appendix A), we propose dedicated interactive certificates for the determinant, echelon forms and the rank profile matrix, achieving linear communication and verifier complexity and no overhead for the prover.

We then proposed in Lucas et al. [2018] a series of certificates for most elementary linear algebra computations over univariate polynomial modules within the same tight complexity estimates. The approach is twofold: for problems which can be embedded over the field of fraction (such as the determinant, the rank, the solution of a linear system), the certification reduces to that of a random projection of the problem over the base field. For problems specific to the module of polynomial matrices, we designed a certificate for rowspace membership that is used as a building block for all other problems (Hermite and Popov form, saturation basis, kernel basis, etc).

Lastly we propose in Dumas et al. [2017a] a new result that shows that *polynomial* time and space certificates are attainable for linear algebra in exponential sizes.

1.3.2. *Secure multiparty computation.* In this framework, several users need to perform a large computation with one or several matrices that are split into into shares between them. The end goal is to perform this computation ensuring correctness and resilience (malicious players can not perturb the computation wihtout being detected) while preserving privacy (each share should remain secret for the other players). Here again generic approaches do exist but usually suffer from large overheads making them unpractical on large scales. Our approach is also to design problem specific protocols for scalar products Dumas et al. [2017b], and matrix multiplication Dumas et al. [2018], based on Strassen's algorithm. The approach combines the use of semi-homomorphic encryption schemes (such as Paillier's or Naccach-Stern's) and random masking.

2. SOFTWARE RELEASES AND INTEGRATION

The software aspects of this deliverable involve the LinBox library and its integration as a computing kernel in the SAGE software. The LinBox ecosystem provides high performance exact linear algebra kernels through and ecosystem of three libraries:

Givaro: a library in charge of field and ring arithmetic: finite fields, multiprecision integers and rationals, univariate polynomials, etc.

fflas-ffpack: a library dedicated to dense linear algebra over finite fields. It offers a similar interface as the numerical BLAS and LAPACK libraries whenever they make sense over a finite field.

LinBox: a library for exact linear with dense, sparse or black-box matrices. It is based on Givaro and fflas-ffpack, offers a higher level interface to the core routines of fflas-ffpack and builds on top of them more sophisticated algorithms, such as Smith Normal Forms, Diophantine and rational solvers, etc.

2.1. LinBox ecosystem

During the course of the project 4 versions of givaro have been released,

givaro-4.0.1: released 24 Feb 2016
givaro-4.0.2: released 30 Jul 2016
givaro-4.0.3: released 17 Nov 2016
givaro-4.0.4: released 23 Nov 2017
givaro-4.1.0: in preparation (expected Oct 2018)

6 versions of `fflas-ffpack`,

fflas-ffpack-2.2.0: released 23 Feb 2016
fflas-ffpack-2.2.1: released 8 Apr 2016
fflas-ffpack-2.2.2: released 30 Jul 2016
fflas-ffpack-2.3.0: released 17 Nov 2017
fflas-ffpack-2.3.1: released 23 Nov 2017
fflas-ffpack-2.3.2: released 21 Dec 2017
fflas-ffpack-2.4.0: in preparation (expected Oct 2018)

and 4 versions of LinBox

linbox-1.4.0: released 25 Feb 2016
linbox-1.4.1: released 8 Apr 2016
linbox-1.4.2: released 30 Jul 2016
linbox-1.5.0: released 17 Nov 2017
linbox-1.5.1: released 23 Nov 2017
linbox-1.5.2: released 8 Dec 2017
linbox-1.6.0: in preparation (expected Oct 2018)

Over these releases, the goal was twofold: a strong improvement in the code quality improving its robustness and its maintainability and the addition of new features, either new algorithms for existing interfaces or extending the scope of problem being addressed.

The robustness of the code has been drastically increased by our adoption of more involved standard for the test suite, the increase of the coverage for these test-suite and the intensive use of a two continuous integration frameworks: `ci.inria` (`http://ci.inria.fr/linbox`), a Jenkins based solution run by Inria, and `travis` (`https://travis-ci.org/linbox-team/`).

Among the new features introduced, we list below the most significant ones:

- The fully functional implementation of the new symmetric triangular factorization resulting from our research in Dumas and Pernet [2018] achieving world's best performance for this task, and competing with the computational efficiency of the corresponding numerical routines in LAPACK;
- the fully functional algorithms computing the rank profile matrix and related echelon forms, following our research in Dumas et al. [2017d];

- a broader support of compiler and architectures, including a better detection and use of SIMD vector instruction sets;
- a parallel implementation of the triangular matrix inverse;
- the deployment of randomized certificates acting as optional sanity checks;
- a rewrite of the prime field code in `Givaro`

2.2. Integration in SAGE

The improvement of the integration of LinBox in SAGE and related issues have been conducted through the following tickets (available on https://trac.sagemath.org/ticket):

#22872 Enhanced LinBox interface.: This meta-ticket gathers information on all distinct tickets contributing to the improvement of the integration of LinBox in SAGE.
#22924 Cleaning of linbox for dense integer matrices: (positive review, released in SAGE 8.0)
#24544 cleaning linbox declarations + reimplement modn interface: (remains one bug on OSX)
#22970 Use `fmpq_mat_t` for rational dense matrices: (positive review, released in SAGE 8.0)
#24544 Cleaning linbox declarations + reimplement modn interface: (needs work)
#23214 Enhance sparse integer matrix with linbox: (needs work)
#24214 Upgrade to givaro-4.0.4 fflas-ffpack-2.3.2 and LinBox-1.5.2: (positive review, released in SAGE 8.2)
#17635 Update Givaro, FFLAS-FFPACK and LinBox: (positive review, released in SageMath 7.4)
#23391 Test timeout in FFPACK::CharPoly: (closed defect)
#25353 fflas and linbox broken with gcc 8.1.0: (closed defect, fixed in SAGE 8.3)
#21579 Errors calculating characteristic polynomials of rational matrices: (closed defect, fixedin SAGE 8.2)
#23919 Sage library: standardize on C99 and C++11: (closed defect, released in SAGE 8.1)
#21578 Problem with fflas.h on Cygwin since #17635: (closed defect, released in SAGE 7.4)

Besides maintaining the version of the three libraries `LinBox, fflas-ffpack` and `Givaro` up to date within SAGE, a large amount of the work in these tickets dealt with cleaning up the interfaces, improving maintainability and efficiency of the interraction and extending the coverage in features accessible for the end user of SAGE.

LIST OF PUBLICATIONS DELIVERED

Jean-Guillaume Dumas and Clément Pernet. Symmetric indefinite triangular factorization revealing the rank profile matrix. In *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, pages 151–158, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5550-6. doi:10.1145/3208976.3209019.

Jean-Guillaume Dumas and Vincent Zucca. Prover efficient public verification of dense or sparse/structured matrix-vector multiplication. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 2017, 22nd Australasian Conference on Information Security and Privacy*, volume 10343 of *Lecture Notes in Computer Science*, pages 115–134. Springer, July 2017. URL http://hal.archives-ouvertes.fr/hal-01503870.

Jean-Guillaume Dumas, Erich L. Kaltofen, Gilles Villard, and Lihong Zhi. Polynomial time interactive proofs for linear algebra with exponential matrix dimensions and scalars given by polynomial time circuits. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 125–132, New York, NY, USA, 2017a. ACM. ISBN 978-1-4503-5064-8. doi:10.1145/3087604.3087640.

Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Dual protocols for private multi-party matrix multiplication and trust computations. *Computers & Security*, (71):51–70, November 2017b. doi:10.1016/j.cose.2017.04.013.

Jean-Guillaume Dumas, David Lucas, and Clément Pernet. Certificates for triangular equivalence and rank profiles. In *Proceedings of the 2017 ACM on International Symposium on Symbolic*

and Algebraic Computation, ISSAC '17, pages 133–140, New York, NY, USA, 2017c. ACM. ISBN 978-1-4503-5064-8. doi:10.1145/3087604.3087609.

Jean-Guillaume Dumas, Clément Pernet, and Ziad Sultan. Fast computation of the rank profile matrix and the generalized bruhat decomposition. *Journal of Symbolic Computation*, 83:187 – 210, 2017d. ISSN 0747-7171. doi:10.1016/j.jsc.2016.11.011. Special issue on the conference ISSAC 2015: Symbolic computation and computer algebra.

Jean-Guillaume Dumas, Julio Fenner, Pascal Lafourcade, David Lucas, Clément Pernet, and Maxime Puys. Secure multi-party matrix multiplication based on strassen-winograd algorithm. Technical report, 2018. hal-01781554.

David Lucas, Vincent Neiger, Clément Pernet, Daniel S. Roche, and Johan Rosenkilde. Interactive certificates for polynomial matrices with sub-linear communication. Technical report, 2018. arXiv-1807.01272 [cs.SC].

Clément Pernet. Computing with quasiseparable matrices. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, pages 389–396, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4380-0. doi:10.1145/2930889.2930915.

Clément Pernet and Arne Storjohann. Time and space efficient generators for quasiseparable matrices. *Journal of Symbolic Computation*, 85:224 – 246, 2018. ISSN 0747-7171. doi:10.1016/j.jsc.2017.07.010. 41th International Symposium on Symbolic and Alge-braic Computation (ISSAC'16).

APPENDIX A. SELECTION OF RESEARCH ARTICLES PUBLISHED IN INTERNATIONAL JOURNALS OR CONFERENCES

Symmetric Indefinite Triangular Factorization Revealing the Rank Profile Matrix*

Jean-Guillaume Dumas
Université Grenoble Alpes
Laboratoire Jean Kuntzmann, CNRS, UMR 5224
38058 Grenoble, CEDEX 9, France
Jean-Guillaume.Dumas@univ-grenoble-alpes.fr

Clément Pernet
Université Grenoble Alpes
Laboratoire Jean Kuntzmann, CNRS, UMR 5224
38058 Grenoble, CEDEX 9, France
Clement.Pernet@univ-grenoble-alpes.fr

ABSTRACT

We present a novel recursive algorithm for reducing a symmetric matrix to a triangular factorization which reveals the rank profile matrix. That is, the algorithm computes a factorization $\mathbf{P}^T\mathbf{AP} = \mathbf{LDL}^T$ where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is lower triangular with a unit diagonal and $\mathbf{D}$ is symmetric block diagonal with 1×1 and 2×2 antidiagonal blocks. This algorithm requires $O(n^2r^{\omega-2})$ arithmetic operations, with n the dimension of the matrix, r its rank and ω an admissible exponent for matrix multiplication. Furthermore, experimental results demonstrate that our algorithm has very good performance: its computational speed matches that of its numerical counterpart and is twice as fast as the unsymmetric exact Gaussian factorization. By adapting the pivoting strategy developed in the unsymmetric case, we show how to recover the rank profile matrix from the permutation matrix and the support of the block-diagonal matrix. We also note that there is an obstruction in characteristic 2 for revealing the rank profile matrix, which requires to relax the shape of the block diagonal by allowing the 2-dimensional blocks to have a non-zero bottom-right coefficient. This relaxed decomposition can then be transformed into a standard $\mathbf{PLDL}^T\mathbf{P}^T$ decomposition at a negligible cost.

ACM Reference Format:
Jean-Guillaume Dumas and Clément Pernet. 2018. Symmetric Indefinite Triangular Factorization Revealing the Rank Profile Matrix. In *ISSAC '18: 2018 ACM International Symposium on Symbolic and Algebraic Computation, July 16–19, 2018, New York, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3208976.3209019

1 INTRODUCTION

Computing a triangular factorization of a symmetric matrix is a commonly used routine to solve symmetric linear systems, or to compute the signature of symmetric bilinear forms. Besides the fact that it is expected to save half of the arithmetic cost of a standard (non-symmetric) Gaussian elimination, it can also recover invariants, such as the signature, specific to symmetric matrices, and thus, e.g., be used to certify positive or negative definiteness or semidefiniteness [13, Corollary 1].

*This work is partly funded by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

ISSAC '18, July 16–19, 2018, New York, NY, USA

ACM ISBN 978-1-4503-5550-6/18/07…$15.00
https://doi.org/10.1145/3208976.3209019

It is a fundamental computation in numerical linear algebra, and is therefore most often presented in the setting of real matrices. When the matrix is positive definite, the Cholesky factorization can be defined: $\mathbf{A} = \mathbf{LL}^T$, where $\mathbf{L}$ is lower triangular for which square roots of diagonal elements have to be extracted. Alternatively, gathering the diagonal elements in a central diagonal matrix yields the LDLT factorization $\mathbf{A} = \mathbf{LDL}^T$ which no longer requires square roots. Similarly as for the LU decomposition, it is only defined for matrices with generic rank profile, i.e. having their $r = \text{rank}(\mathbf{A})$ first leading principal minors non-zero. For arbitrary matrices, symmetric permutations may lead to the former situations: $\mathbf{PAP}^T = \mathbf{LDL}^T$. However, this is unfortunately not always the case. For instance there is no permutation $\mathbf{P}$ such that $\mathbf{P}^T\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]\mathbf{P}$ has an LDLT factorization with a diagonal $\mathbf{D}$. This led to a series of generalizations where the matrix $\mathbf{D}$ was replaced first by a tridiagonal symmetric matrix by Parlett and Reid [14], improved by Aasen [1], achieving half the arithmetic cost of Gaussian elimination. Bunch and Kaufman then replaced this tridiagonal matrix by a block diagonal composed of 1 or 2-dimensional antidiagonal blocks to obtain an *indefinite triangular factorization.*

Pivoting. In numerical linear algebra, the choice of the permutation matrix is mainly driven by the need to ensure a good numerical quality of the decomposition. Bunch and Parlett [6] use a full pivoting technique, requiring a cubic number of tests. Bunch and Kaufman's pivoting strategy, implemented in LAPACK, uses a partial pivoting requiring only a quadratic number of tests.

In the context of exact linear algebra, for instance when computing over a finite field, numerical stability is no longer an issue. However, the computation of echelon forms and rank profiles, central in many applications, impose further constraints on the pivoting. A characterization of the requirements for the pivoting strategy is given in [10, 11] so that a PLUQ decomposition can reveal these rank profiles and echelon forms in the non-symmetric case. In particular, it is shown that pivot selection minimizing the lexicographic order on the coordinate of the pivot, combined with row and column rotations to move the pivot to the diagonal, enable the computation of the rank profile matrix, an invariant from which all rank profile information or the row and the column echelon form can be recovered.

Recursive algorithms. As in numerical linear algebra, we try to gather arithmetic operations in level 3 BLAS operations (matrix multiplication based), for it delivers the best computation throughput. Numerical software often use tiled implementations, especially when the pivoting is more constrained by the symmetry [12, 16], or in order to define communication avoiding variants [4]. In exact

linear algebra sub-cubic matrix multiplication, such as Strassen's algorithm, can be extensively used with no numerical instability issues. This led to the design of recursive algorithms, which was proven successful in the unsymmetric case, including for shared memory parallel computations [9].

Contribution. The contribution here is to propose a recursive algorithm producing a symmetric factorization PLDLTPT over any field, from which the rank profile matrix of the input can be recovered. This algorithm is a recursive variant of Bunch and Kaufman's algorithm [5] where the pivoting strategy has been replaced by the one developed previously by the authors in the unsymmetric case [11]. Compared to the recursive adaptation of Aasen's algorithm in [15], our algorithm leads to a similar data partitioning but does not suffer from an arithmetic overhead with respect to Aasen's algorithm. Our algorithm has time complexity $O(n^2 r^{\omega-2})$ where ω is an admissible exponent for matrix multiplication and r is the rank of the input matrix of dimension n. With $\omega = 3$, the leading constant in the time complexity is 1/3, matching that of the best alternative algorithms based on cubic time linear algebra.

In Section 2 we show that in characteristic two the rank profile matrix can not always be revealed by a symmetric factorization with antidiagonal blocks: sometimes antitriangular blocks are also required. Then we recall in Section 3 the main required level 3 linear algebra subroutines. In Section 4 we present the main recursive algorithm. An alternative iterative Crout variant is presented in Section 5 to be used as a base case in the recursion. We finally show, in Section 6, experiments of the resulting implementation over a finite field. They demonstrate the efficiency of cascading the recursive algorithm with the base case variant, especially with matrices involving a lot of pivoting. They finally confirm a speed-up by a factor of about 2 compared to the state of the art unsymmetric Gaussian elimination.

2 THE SYMMETRIC RANK PROFILE MATRIX

2.1 The pivoting matrix

Theorem 1 recalls the definition of the rank profile matrix (RPM).

Theorem 1 ([10]). *Let $\mathbf{A} \in \mathbb{F}^{m\times n}$. There exists a unique $m \times n$ $\{0,1\}$-matrix $\mathcal{R}_\mathbf{A}$ with r 1's in rook placement of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of $\mathbf{A}$. This matrix is called the* rank profile matrix of $\mathbf{A}$.

Lemma 1. *A symmetric matrix has a symmetric rank profile matrix.*

Proof. Otherwise, the rank of some leading submatrix of $\mathbf{A}$ and the same leading submatrix of $\mathbf{A}^T$ would be different which is absurd. □

Also, any symmetric matrix has a triangular decomposition $\mathbf{A} = \mathbf{PLDL}^T\mathbf{P}^T$ where $\mathbf{L}$ is unit lower triangular, $\mathbf{P}$ is a permutation matrix and $\mathbf{D}$ is block diagonal, formed by 1-dimensional scalar blocks or 2-dimensional antidiagonal blocks of the form $\left[\begin{smallmatrix} 0 & x \\ x & 0 \end{smallmatrix}\right]$. We further define the support matrix of such a block diagonal matrix as the localization of the non-zero elements:

Definition 1. *The* support matrix *of a block diagonal matrix $\mathbf{D}$, formed by 1-dimensional scalar blocks or 2-dimensional blocks of the form $\left[\begin{smallmatrix} 0 & x \\ x & 0 \end{smallmatrix}\right]$ is the block diagonal $\{0,1\}$-matrix $\mathbf{\Psi_D}$ such that $\mathbf{D} = \mathbf{\Psi_D}\overline{\mathbf{D}}$, with $\overline{\mathbf{D}}$ a diagonal matrix.*

Definition 2. *The* pivoting matrix *of a $\mathbf{PLDL}^T\mathbf{P}^T$ decomposition is the matrix $\mathbf{\Pi} = \mathbf{P\Psi_D P}^T$.*

Definition 3. *A $\mathbf{PLDL}^T\mathbf{P}^T$ decomposition is said to* reveal the rank profile matrix *of a symmetric matrix $\mathbf{A}$ if its pivoting matrix equals the rank profile matrix of $\mathbf{A}$.*

2.2 Antitriangular blocks in characteristic two

In zero or odd characteristic, we show next that one can always find such a PLDLTPT decomposition revealing the rank profile matrix. In characteristic two, however, this is not always possible.

Lemma 2. *In characteristic 2, there is no symmetric indefinite elimination revealing the rank profile matrix of $\mathbf{A} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right]$.*

Proof. Let $\mathbf{J}$ be the 2×2 anti-diagonal identity matrix. This is also the rank profile matrix of $\mathbf{A}$. Now, we let $\mathbf{L} = \left[\begin{smallmatrix} 1 & 0 \\ x & 1 \end{smallmatrix}\right], \overline{\mathbf{D}} = \left[\begin{smallmatrix} y & 0 \\ 0 & z \end{smallmatrix}\right]$. As the permutation matrices involved, $\mathbf{P}$ and $\mathbf{\Psi_D}$, can only be either the identity matrix or $\mathbf{J}$, there are then four cases:

(1) $\mathbf{A} = \mathbf{L}\cdot\overline{\mathbf{D}}\cdot\mathbf{L}^T = \left[\begin{smallmatrix} y & xy \\ xy & x^2y+z \end{smallmatrix}\right]$, but $y = 0$ and $\text{rank}(\overline{\mathbf{D}}) = \text{rank}(\mathbf{A}) = 2$ are incompatible.

(2) $\mathbf{A} = \mathbf{J}\cdot\mathbf{L}\cdot\overline{\mathbf{D}}\cdot\mathbf{L}^T\cdot\mathbf{J}^T = \left[\begin{smallmatrix} x^2y+z & xy \\ xy & y \end{smallmatrix}\right]$, but $\mathbf{I} = \mathbf{J}\cdot\mathbf{I}\cdot\mathbf{J} \neq \mathbf{J} = \mathcal{R}_\mathbf{A}$.

(3) $\mathbf{A} = \mathbf{L}\cdot\overline{\mathbf{D}}\cdot\mathbf{J}\cdot\mathbf{L}^T = \left[\begin{smallmatrix} 0 & y \\ z & xy+xz \end{smallmatrix}\right]$, but we need $y = z$ for the symmetry and then $2xy = 0 \neq 1$ in characteristic 2.

(4) $\mathbf{A} = \mathbf{J}\cdot\mathbf{L}\cdot\overline{\mathbf{D}}\cdot\mathbf{J}\cdot\mathbf{L}^T\cdot\mathbf{J}^T = \left[\begin{smallmatrix} xy+xz & z \\ y & 0 \end{smallmatrix}\right]$ but the bottom right coefficient of $\mathbf{A}$ is non zero.

□

However, one can generalize the PLDLTPT decomposition to a block diagonal matrix $\mathbf{D}$ having 2-dimensional blocks of the form $\left[\begin{smallmatrix} 0 & c \\ c & d \end{smallmatrix}\right]$ (lower antitriangular). Then the notion of support matrix can be generalized: for such a block diagonal matrix $\mathbf{D}$, $\mathbf{\Psi_D}$ is the block diagonal $\{0,1\}$ matrix such that $\mathbf{D} = \mathbf{\Psi_D}\overline{\mathbf{D}}$, with $\overline{\mathbf{D}}$ an upper triangular bidiagonal matrix (or equivalently such that $\mathbf{D} = \overline{\mathbf{D}}\mathbf{\Psi_D}$, with $\overline{\mathbf{D}}$ lower triangular bidiagonal). For instance $\left[\begin{smallmatrix} 0 & c \\ c & d \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} c & d \\ 0 & c \end{smallmatrix}\right] = \left[\begin{smallmatrix} c & 0 \\ d & c \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

With these generalized definitions, we show in Section 4, that there exists RPM-revealing PLDLTPT decompositions.

2.3 Antitriangular decomposition

Then, such a generalized decomposition can always be further reduced to a strict PLDLTPT decomposition by eliminating each of the antitriangular blocks. For this, the observation is that in characteristic two, a symmetric lower antitriangular 2×2 block is invariant under any symmetric triangular transformation: $\left[\begin{smallmatrix} 1 & \\ x & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} & c \\ c & d \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & x \\ & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} & c \\ c & 2cx+d \end{smallmatrix}\right] \equiv \left[\begin{smallmatrix} & c \\ c & d \end{smallmatrix}\right] \mod 2 = \left[\begin{smallmatrix} 1 & \\ & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} & c \\ c & d \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & \\ & 1 \end{smallmatrix}\right]$. Thus for each

2×2 block in a tridiagonal decomposition, the corresponding 2×2 diagonal block in $\mathbf{L}$ can be replaced by $\mathbf{I}_2$, via a multiplication by $\left[\begin{smallmatrix} 1 & \\ -x & 1 \end{smallmatrix}\right]$.

Further, we have that: $\left[\begin{smallmatrix} & c \\ c & d \end{smallmatrix}\right] = \mathbf{J}\left[\begin{smallmatrix} 1 & \\ c/d & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} d & \\ & -c^2/d \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & c/d \\ & 1 \end{smallmatrix}\right]\mathbf{J}$. Now $\mathbf{J}$ commutes with the identity $\mathbf{I}_2$ matrix. Therefore we have that: $\left[\begin{smallmatrix} 1 & \\ x & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & \\ -x & 1 \end{smallmatrix}\right]\mathbf{J}\left[\begin{smallmatrix} 1 & \\ c/d & 1 \end{smallmatrix}\right] = \mathbf{J}\left[\begin{smallmatrix} 1 & \\ x & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & \\ -x & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & \\ c/d & 1 \end{smallmatrix}\right]$.

Thus, to eliminate the antitriangular blocks, create a triangular matrix $\mathbf{L}_J$ that starts as the identity and where its $i, i+1$ blocks corresponding to a $\left[\begin{smallmatrix} 1 & \\ x & 1 \end{smallmatrix}\right]$ block in $\mathbf{L}$ is a $\left[\begin{smallmatrix} 1 & \\ c/d-x & 1 \end{smallmatrix}\right]$ block (associated to an antitriangular $\left[\begin{smallmatrix} & c \\ c & d \end{smallmatrix}\right]$ block, with $d \neq 0$, in $\mathbf{D}$). Then replace the triangular matrix $\mathbf{L}$ by $\tilde{\mathbf{L}} = \mathbf{L}\cdot\mathbf{L}_J$. Also, modify the diagonal matrix $\mathbf{D}$, to $\tilde{\mathbf{D}}$ such that the $\left[\begin{smallmatrix} & c \\ c & d \end{smallmatrix}\right]$ blocks of $\mathbf{D}$ are replaced by $\left[\begin{smallmatrix} d & \\ & -c^2/d \end{smallmatrix}\right]$ blocks in $\tilde{\mathbf{D}}$. Finally, create a permutation matrix $\mathbf{P}_J$, starting from the identity matrix, where each identity block at position $i, i+1$ corresponding to an antitriangular block in $\mathbf{D}$ is replaced by $\mathbf{J}$. Then $\tilde{\mathbf{P}} = \mathbf{P}\cdot\mathbf{P}_J$.

From this we have now a symmetric PLDLTPT factorization, $\mathbf{A} = \tilde{\mathbf{P}}\tilde{\mathbf{L}}\tilde{\mathbf{D}}\tilde{\mathbf{L}}^T\tilde{\mathbf{P}}^T$, with purely 1×1 and 2×2 antidiagonal blocks in $\tilde{\mathbf{D}}$ (but then a direct access to the rank profile matrix, $\mathbf{P}\Psi_{\mathbf{D}}\mathbf{P}^T$, might not be possible from $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{D}}$).

In the following we present some building blocks and then algorithms computing an RPM-revealing symmetric indefinite triangular factorization.

3 BUILDING BLOCKS

We recall here some of the standard algorithms from the BLAS3 [7] and LAPACK [2] interfaces and generalization thereof [3], which will be used to define the main block recursive symmetric elimination.

- **gemm** $(\mathbf{C}, \mathbf{A}, \mathbf{B})$**:** general matrix multiplication. Computes $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{AB}$.
- **trmm** $(\mathbf{U}, \mathbf{B})$**:** multiply a triangular and a rectangular matrix inplace. Computes $\mathbf{B} \leftarrow \mathbf{UB}$ where $\mathbf{B}$ is $m \times n$ and $\mathbf{U}$ is upper or lower triangular.
- **trmm** $(\mathbf{C}, \mathbf{U}, \mathbf{B})$**:** multiply a triangular and a rectangular matrix. Computes $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{UB}$ where $\mathbf{B}$ and $\mathbf{C}$ are $m \times n$ and $\mathbf{U}$ is upper or lower triangular. This is an adaptation of the BLAS3 `trmm` to leave the $\mathbf{B}$ operand unchanged.
- **trsm** $(\mathbf{U}, \mathbf{B})$**:** solve a triangular system with matrix right hand-side. Computes $\mathbf{B} \leftarrow \mathbf{U}^{-1}\mathbf{B}$ where $\mathbf{B}$ is $m \times n$ and $\mathbf{U}$ is upper or lower triangular.
- **syrdk** $(\mathbf{C}, \mathbf{A}, \mathbf{D})$**:** symmetric rank k update with diagonal scaling. Computes the upper or lower triangular part of the symmetric matrix $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{ADA}^T$ where $\mathbf{A}$ is $n \times k$ and D is diagonal or block diagonal.
- **syrd2k** $(\mathbf{C}, \mathbf{A}, \mathbf{D}, \mathbf{B})$**:** symmetric rank $2k$ update with diagonal scaling. Computes the upper or lower triangular part of the symmetric matrix $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{ADB}^T - \mathbf{BDA}^T$ where $\mathbf{A}$ and $\mathbf{B}$ are $n \times k$, and $\mathbf{D}$ is diagonal or block diagonal.

In addition, we use **scal** for multiplication by a diagonal matrix, **dadd** for matrix addition and diagonal scaling, and we need to introduce the **trssyr2k** routine solving Problem 1:

Problem 1. *Let $\mathbb{F}$ be a field of characteristic different than 2. Given a symmetric matrix $\mathbf{C} \in \mathbb{F}^{n\times n}$ and a unit upper triangular matrix $\mathbf{U} \in \mathbb{F}^{n\times n}$, find an upper triangular matrix $X \in \mathbb{F}^{n\times n}$ such that $\mathbf{X}^T\mathbf{U} + \mathbf{U}^T\mathbf{X} = \mathbf{C}$.*

In characteristic 2, the diagonal of $X^T\mathbf{U} + \mathbf{U}^TX$ is always zero for any matrix X and $\mathbf{U}$, hence Problem 1 has no solution as soon as $\mathbf{C}$ has a non-zero diagonal element.

However in characteristic zero or odd, Algorithm 1 presents a recursive implementation of this routine, and is in the same time a constructive proof of the existence of such a solution. Note that it performs a division by 2 in line 2, and therefore requires that the base field does not have characteristic two.

Algorithm 1 `trssyr2k` $(\mathbf{U}, \mathbf{C})$

Require: $\mathbf{U}$, $n \times n$ full-rank upper triangular
Require: $\mathbf{C}$, $n \times n$, symmetric
Ensure: $\mathbf{C} \leftarrow \mathbf{X}$, $n \times n$ upper triangular, s.t. $\mathbf{X}^T\mathbf{U} + \mathbf{U}^T\mathbf{X} = \mathbf{C}$
1: **if** $m = 1$ **then**
2: **return** $\mathbf{C}_{1,1} \leftarrow \frac{1}{2}\mathbf{C}_{1,1}\cdot\mathbf{U}_{1,1}^{-1}$;
3: **end if**
4: Split $\mathbf{C} = \left[\begin{smallmatrix} \mathbf{C}_1 & \mathbf{C}_2 \\ \mathbf{C}_2^T & \mathbf{C}_3 \end{smallmatrix}\right]$, $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ & \mathbf{U}_3 \end{smallmatrix}\right]$, $\mathbf{C}_1$ and $\mathbf{U}_1$ are $\left\lfloor \frac{n}{2} \right\rfloor \times \left\lfloor \frac{n}{2} \right\rfloor$
5: Find $\mathbf{X}_1$ s.t. $\mathbf{X}_1^T\mathbf{U}_1 + \mathbf{U}_1^T\mathbf{X}_1 = \mathbf{C}_1$ $\{$`trssyr2k` $(\mathbf{U}_1, \mathbf{C}_1)\}$
6: $\mathbf{D}_2 \leftarrow \mathbf{C}_2 - \mathbf{X}_1^T\mathbf{U}_2$ $\{$`trmm` $(\mathbf{X}_1^T, \mathbf{U}_2)\}$
7: $\mathbf{X}_2 \leftarrow \mathbf{U}_1^{-T}\mathbf{D}_2$ $\{$`trsm` $(\mathbf{U}_1^T, \mathbf{D}_2)\}$
8: $\mathbf{D}_3 \leftarrow \mathbf{C}_3 - (\mathbf{X}_2^T\mathbf{U}_2 + \mathbf{U}_2^T\mathbf{X}_2)$ $\{$`syrd2k` $(\mathbf{X}_2, \mathbf{U}_2)\}$
9: Find $\mathbf{X}_3$ s.t. $\mathbf{X}_3^T\mathbf{U}_3 + \mathbf{U}_3^T\mathbf{X}_3 = \mathbf{D}_3$ $\{$`trssyr2k` $(\mathbf{U}_3, \mathbf{D}_3)\}$

Remark 1. *Note that algorithm 1 computes the solution $\mathbf{X}$ in place on the symmetric storage of $\mathbf{C}$: by induction $\mathbf{X}_1$ and $\mathbf{X}_3$ overwrite $\mathbf{C}_1$ and $\mathbf{C}_3$, and $\mathbf{X}_2$ overwrites $\mathbf{C}_2$ according to the specifications of the generalized* `trmm` *routine.*

Lemma 3. *Algorithm* `trssyr2k` *is correct and runs in $O(n^\omega)$ arithmetic operations.*

Proof. Using the notations of Algorithm 1, let $\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 \\ & \mathbf{X}_3 \end{bmatrix}$. Then expanding $\mathbf{X}^T\mathbf{U} + \mathbf{U}^T\mathbf{X}$ gives

$$\begin{bmatrix} \mathbf{X}_1^T\mathbf{U}_1 + \mathbf{U}_1^T\mathbf{X}_1 & \mathbf{X}_1^T\mathbf{U}_2 + \mathbf{U}_1^T\mathbf{X}_2 \\ (\mathbf{X}_1^T\mathbf{U}_2 + \mathbf{U}_1^T\mathbf{X}_2)^T & \mathbf{X}_2^T\mathbf{U}_2 + \mathbf{U}_2^T\mathbf{X}_2 + \mathbf{X}_3^T\mathbf{U}_3 + \mathbf{U}_3^T\mathbf{X}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{C}_2 \\ \mathbf{C}_2^T & \mathbf{C}_3 - \mathbf{D}_3 + \mathbf{X}_3^T\mathbf{U}_3 + \mathbf{U}_3^T\mathbf{X}_3 \end{bmatrix} = \mathbf{C}.$$

which proves the correction by induction. The arithmetic cost satisfies a recurrence of the form $T(n) = 2T(n/2) + Cn^\omega$ and is therefore $T(n) = O(n^\omega)$. □

4 A BLOCK RECURSIVE ALGORITHM

4.1 Sketch of the recursive algorithm

The design of a block recursive PLDLTPT algorithm is based on the generalization of the 2×2 case into a block 2×2 block algorithm. While scalars could be either 0 or invertible, the difficulty in elimination algorithms, is that a submatrix could be rank deficient but non-zero. We start here an overview of the recursive algorithm by considering that the leading principal block is either all zero

or invertible. We will later give the general presentation of the algorithm where its rank could be arbitrary.

Let $\mathbf{M} \in \mathbb{F}^{(m+n)\times(m+n)}$ be the symmetric matrix to be factorized. Consider its block decomposition $\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix}$ where $\mathbf{A} \in \mathbb{F}^{m\times m}$ and $\mathbf{C} \in \mathbb{F}^{n\times n}$ are also symmetric.

If $\mathbf{A}$ is full rank, then a recursive call will produce $\mathbf{A} = \mathbf{PLDL}^T\mathbf{P}^T$, and $\mathbf{M}$ can thus be decomposed as:

$$\mathbf{M} = \begin{bmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{G} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix} \begin{bmatrix} \mathbf{L}^T & \mathbf{G}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{P}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix},$$

where $\mathbf{G}$ is such that $\mathbf{PLDG}^T = \mathbf{B}$ and $\mathbf{Z} = \mathbf{C} - \mathbf{GDG}^T$. Thus $\mathbf{G}$ can be computed as the transpose of $\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{P}^{-1}\mathbf{B}$ which can be obtained by a call to `trsm`, some permutations and a diagonal scaling. Then $\mathbf{Z}$ is computed by a call to `syrdk`. A second recursive call will then decompose $\mathbf{Z}$ and lead to the final factorization of $\mathbf{M}$.

Now if $\mathbf{A}$ is the zero matrix, one is reduced to factorize the matrix $\mathbf{M} = \left[\begin{smallmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{smallmatrix}\right]$. In order to recover the rank profile matrix, one has to first look for pivots in $\mathbf{B}$ before considering the block $\mathbf{C}$. Therefore diagonal pivoting is not an option here. Then the matrix $\mathbf{B}$, which we assume has full rank for the moment, can be decomposed in a *PLDUQ* factorization ($\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, $\mathbf{L}$ and $\mathbf{U}$ are respectively unit lower and unit upper triangular, $\mathbf{D}$ is diagonal). We then need to distinguish two cases depending on whether the field characteristic is two or not.

4.1.1 Zero or odd characteristic case. If the characteristic is zero or odd, $\mathbf{M}$ can thus be decomposed as:

$$\mathbf{M} = \begin{bmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^T \end{bmatrix} \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{G} & \mathbf{U}^T \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{D} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{L}^T & \mathbf{G}^T \\ \mathbf{0} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{P}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{bmatrix},$$

where $\mathbf{G}$ is such that $\mathbf{Q}^T(\mathbf{GDU}+\mathbf{U}^T\mathbf{DG}^T)\mathbf{Q} = \mathbf{C}$. To compute $\mathbf{G}$, one can first permute $\mathbf{C}$ to get $\mathbf{C}' = \mathbf{QCQ}^T$ (which remains symmetric) and then use a call to `trssyr2k`.

4.1.2 Characteristic two case. In characteristic two, the equation $\mathbf{GDU} + \mathbf{U}^T\mathbf{DG}^T = \mathbf{QCQ}^T$ in unknown $\mathbf{G}$ has in general no solution (as soon as $\mathbf{C}$ has a non-zero diagonal element).

However, one can still relax Problem 1 and allow the elimination to leave a diagonal of elements not zeroed out. Following Lemma 2, the idea is then to decompose $\mathbf{M}$ into a block tridiagonal form:

$$\mathbf{M} = \begin{bmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^T \end{bmatrix} \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{G} & \mathbf{U}^T \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{D} \\ \mathbf{D} & \mathbf{\Delta} \end{bmatrix} \begin{bmatrix} \mathbf{L}^T & \mathbf{G}^T \\ \mathbf{0} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{P}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{bmatrix},$$

where $\mathbf{\Delta}$ is a diagonal matrix and now $\mathbf{G}$ is such that $\mathbf{Q}^T(\mathbf{GDU} + \mathbf{U}^T\mathbf{DG}^T + \mathbf{U}^T\mathbf{\Delta U})\mathbf{Q} = C$. Therefore $\mathbf{\Delta}$ can be chosen such that the diagonal of $\mathbf{C}'' = \mathbf{QCQ}^T - \mathbf{U}^T\mathbf{\Delta U} = \mathbf{C}' - \mathbf{U}^T\mathbf{\Delta U}$ is zero. As $\mathbf{U}$ is unit upper triangular, a simple pass over its coefficients is sufficient to find such a $\mathbf{\Delta}$: let $\Delta_{ii} = \mathbf{C}'_{ii} - \sum_{j=1}^{i-1} \Delta_{jj}\mathbf{U}^2_{j,i}$. The algorithm is thus to permute $\mathbf{C}$ to get $\mathbf{C}'$; then compute $\mathbf{\Delta}$ with the recursive relation above and update $\mathbf{C}'' = \mathbf{C}' - \mathbf{U}^T\mathbf{\Delta U}$ with a `syrdk`. $\mathbf{C}''$ remains symmetric but with a zero diagonal and now `trssyr2k` can be applied.

4.2 The actual recursive algorithm

4.2.1 First phase: recursive elimination. In the general case, the leading matrices are not full rank, and we have to consider intermediate steps. For the symmetric matrix $M \in \mathbb{F}^{(m+n)\times(m+n)}$ of Section 4.1, its leading principal block $\mathbf{A} \in \mathbb{F}^{m\times m}$ has rank $r \leq m$. Thus its actual recursive decomposition is of the form:

$$\mathbf{A} = \mathbf{P}_1 \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{M}_1 \end{bmatrix} \mathbf{D}_1 \begin{bmatrix} \mathbf{L}_1^T & \mathbf{M}_1^T \end{bmatrix} \mathbf{P}_1^T,$$

where $\mathbf{L}_1 \in \mathbb{F}^{r\times r}$ is full rank unit lower triangular, $\mathbf{D}_1 \in \mathbb{F}^{r\times r}$ is block diagonal with 1 or 2-dimensional diagonal blocks, and $\mathbf{M}_1 \in \mathbb{F}^{(m-r)\times r}$. Therefore, forgetting briefly the permutations, the decomposition of M becomes:

$$\mathbf{M} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{M}_1 & \mathbf{I} & \mathbf{0} \\ \mathbf{G} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{D}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Y} \\ \mathbf{0} & \mathbf{Y}^T & \mathbf{Z} \end{bmatrix} \begin{bmatrix} \mathbf{L}_1^T & \mathbf{M}_1^T & \mathbf{G}^T \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix},$$

where $\mathbf{Y}$ is such that $\mathbf{B} = \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{M}_1 \end{bmatrix} \mathbf{D}_1\mathbf{G}^T + \begin{bmatrix} \mathbf{0} \\ \mathbf{Y} \end{bmatrix}$.

From this point on, there remains to factorize the submatrix $\left[\begin{smallmatrix} \mathbf{0} & \mathbf{Y} \\ \mathbf{Y}^T & \mathbf{Z} \end{smallmatrix}\right]$. This will be carried out by the algorithm described in the next section, working on a matrix with a zero leading principal submatrix. Supposing for now that this is possible, Algorithm 2 summarizes the whole procedure.

Algorithm 2 Recursive symmetric indefinite elimination

Require: $\mathbf{A} \in \mathbb{F}^{m\times m}$ and $\mathbf{C} \in \mathbb{F}^{n\times n}$ both symmetric, $\mathbf{B} \in \mathbb{F}^{m\times n}$.
Ensure: $\mathbf{P}$ permutation, $\mathbf{L}$ unit lower triangular, $\mathbf{D}$ block diagonal, s.t. $\left[\begin{smallmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{smallmatrix}\right] = \mathbf{PLDL}^T\mathbf{P}^T$.

1: Decompose $\mathbf{A} = \mathbf{P}_1 \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{M}_1 \end{bmatrix} \begin{bmatrix} \mathbf{D}_1 \end{bmatrix} \begin{bmatrix} \mathbf{L}_1^T & \mathbf{M}_1^T \end{bmatrix} \mathbf{P}_1^T$ {Alg. 2}
2: let $r = \text{rank}(\mathbf{A})$ s.t. $\mathbf{L}_1$ and $\mathbf{D}_1$ are $r \times r$.
3: $\mathbf{B}' = \mathbf{P}_1^T\mathbf{B}$ {perm $(\mathbf{P}_1^T, \mathbf{B})$}
4: Split $\mathbf{B}' = \begin{bmatrix} \mathbf{B}'_1 \\ \mathbf{B}'_2 \end{bmatrix}$ where $\mathbf{B}'_1$ is $r \times n$.
5: $\mathbf{X} \leftarrow \mathbf{L}_1^{-1}\mathbf{B}'_1$ {trsm $(\mathbf{L}_1, \mathbf{B}'_1)$}
6: $\mathbf{Y} \leftarrow \mathbf{B}'_2 - \mathbf{M}_1\mathbf{X}$ {gemm $(\mathbf{B}'_2, \mathbf{M}_1, \mathbf{X})$}
7: $\mathbf{G} \leftarrow \mathbf{X}^T\mathbf{D}_1^{-1}$ {scal $(\mathbf{X}^T, \mathbf{D}_1^{-1})$}
8: $\mathbf{Z} \leftarrow \mathbf{C} - \mathbf{GD}_1\mathbf{G}^T$ {syrdk $(\mathbf{C}, \mathbf{G}, \mathbf{D}_1)$}
9: Decompose $\begin{bmatrix} \mathbf{0} & \mathbf{Y} \\ \mathbf{Y}^T & \mathbf{Z} \end{bmatrix} = \mathbf{P}_2\mathbf{L}_2\mathbf{D}_2\mathbf{L}_2^T\mathbf{P}_2^T$ {Alg. 3}
10: $\mathbf{P} \leftarrow \left[\begin{smallmatrix} \mathbf{P}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_n \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} \mathbf{I}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_2 \end{smallmatrix}\right]$
11: $\mathbf{N}_1 \leftarrow \mathbf{P}_2^T \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{G} \end{bmatrix}$ {perm $(\mathbf{P}_2^T, \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{G} \end{bmatrix})$}
12: $\mathbf{L} \leftarrow \left[\begin{smallmatrix} \mathbf{L}_1 & \\ \mathbf{N}_1 & \mathbf{L}_2 \end{smallmatrix}\right]$
13: $\mathbf{D} \leftarrow \left[\begin{smallmatrix} \mathbf{D}_1 & \\ & \mathbf{D}_2 \end{smallmatrix}\right]$

The correctness of Algorithm 2 is stated in the following Theorem 2 and proven in the remaining of the current Section.

Theorem 2. *Algorithm 2 correctly computes a symmetric indefinite PLDLTPT factorization revealing the rank profile matrix.*

4.2.2 Second phase: off-diagonal pivoting. Consider $\mathbf{M} = \left[\begin{smallmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{smallmatrix}\right]$, where $\mathbf{B} \in \mathbb{F}^{m\times n}$, with $m \leq n$, has now an arbitrary rank $r \leq m$. Then its PLDUQ decomposition is of the form

$$\mathbf{B} = \mathbf{P}\begin{bmatrix}\mathbf{L}_1\\ \mathbf{M}_1\end{bmatrix}\begin{bmatrix}\mathbf{D}_1\end{bmatrix}\begin{bmatrix}\mathbf{U}_1 & \mathbf{V}_1\end{bmatrix}\mathbf{Q},$$

with $\mathbf{D}_1$ diagonal, and $\mathbf{L}_1$ and $\mathbf{U}_1$ unit square triangular matrices, all three of order r. Then consider a conformal block decomposition of $\mathbf{QCQ}^T = \left[\begin{smallmatrix} \mathbf{C}_1 & \mathbf{C}_2 \\ \mathbf{C}_2^T & \mathbf{C}_3 \end{smallmatrix}\right]$ where $\mathbf{C}_1$ is $r \times r$. It remains to eliminate $\mathbf{C}_1$ and $\mathbf{C}_2$ with the pivots found in B, which leads to the following factorization:

$$\mathbf{M} = \begin{bmatrix}\mathbf{P} & \\ & \mathbf{Q}^T\end{bmatrix}\begin{bmatrix}\mathbf{L}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{M}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{G}_1 & \mathbf{U}_1^T & \mathbf{0}\\ \mathbf{G}_2 & \mathbf{V}_1^T & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{D}_1 & \mathbf{0}\\ \mathbf{D}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{Z}\end{bmatrix}\times \begin{bmatrix}\mathbf{L}_1^T & \mathbf{M}_1^T & \mathbf{G}_1^T & \mathbf{G}_2^T\\ \mathbf{0} & \mathbf{0} & \mathbf{U}_1 & \mathbf{V}_1\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{P}^T & \\ & \mathbf{Q}\end{bmatrix} \tag{1}$$

where $\mathbf{G}_1$ satisfies

$$\mathbf{U}_1^T\mathbf{D}_1\mathbf{G}_1^T + \mathbf{G}_1\mathbf{D}_1\mathbf{U}_1 = \mathbf{C}_1 \tag{2}$$

and $\mathbf{G}_2 = (\mathbf{C}_2^T - \mathbf{V}_1^T\mathbf{D}_1\mathbf{G}_1^T)\mathbf{U}_1^{-1}\mathbf{D}_1^{-1}$ and $\mathbf{Z} = \mathbf{C}_3 - (\mathbf{V}_1^T\mathbf{D}_1\mathbf{G}_2^T + \mathbf{G}_2\mathbf{D}_1\mathbf{V}_1)$.

In order to produce an LDLT decomposition, there still remains to perform permutations to

(1) compact the leading elements of the lower triangular matrix into a $2r \times 2r$ invertible leading triangular submatrix,
(2) make the $\left[\begin{smallmatrix} \mathbf{D}_1 & \\ & \mathbf{D}_1 \end{smallmatrix}\right]$ matrix block diagonal with 1 or 2-dimensional diagonal blocks.

The permutation matrix

$$\mathbf{P}_c = \begin{bmatrix}\mathbf{I}_r & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{m-r} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_r & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{n-r}\end{bmatrix}, \tag{3}$$

corresponding to a block circular rotation, takes care of condition 1, while preserving precedence in the non-pivot rows. This is a requirement for the factorization to reveal the rank profile matrix [11, Theorem 20]. The decomposition becomes

$$N = \begin{bmatrix}\mathbf{P} & \\ & \mathbf{Q}^T\end{bmatrix}\mathbf{P}_c\begin{bmatrix}\mathbf{L}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{G}_1 & \mathbf{U}_1^T & \mathbf{0}\\ \mathbf{M}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{G}_2 & \mathbf{V}_1^T & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{D}_1 & \mathbf{0}\\ \mathbf{D}_1 & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{Z}\end{bmatrix}\times \begin{bmatrix}\mathbf{L}_1^T & \mathbf{G}_1^T & \mathbf{M}_1^T & \mathbf{G}_2^T\\ \mathbf{0} & \mathbf{U}_1 & \mathbf{0} & \mathbf{V}_1\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}\end{bmatrix}\mathbf{P}_c^T\begin{bmatrix}\mathbf{P}^T & \\ & \mathbf{Q}\end{bmatrix}. \tag{4}$$

In order to achieve Condition 2, we will transform the matrix $\left[\begin{smallmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{0} \end{smallmatrix}\right]$ into the block diagonal matrix $\mathrm{Diag}\left(\left[\begin{smallmatrix} 0 & d_i \\ d_i & 0 \end{smallmatrix}\right]\right)$ where d_i is the ith diagonal element in $\mathbf{D}_1$. To describe the process, we will focus on the matrix

$$\mathbf{M}_2 = \begin{bmatrix}\mathbf{L}_1 & \mathbf{0}\\ \mathbf{G}_1 & \mathbf{U}_1^T\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{D}_1\\ \mathbf{D}_1 & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{L}_1^T & \mathbf{G}_1^T\\ \mathbf{0} & \mathbf{U}_1\end{bmatrix} = \overline{\mathbf{L}}\cdot\overline{\Delta}\cdot\overline{\mathbf{L}}^T,$$

and consider a splitting in halves of the matrix $\mathbf{D}_1 = \left[\begin{smallmatrix} \mathbf{D}_{11} & \\ & \mathbf{D}_{12} \end{smallmatrix}\right]$ where $\mathbf{D}_{11}$ has order r_1 and $\mathbf{D}_{12}$ order r_2. This leads to the conformal decomposition

$$\begin{bmatrix}\mathbf{L}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{L}_{12} & \mathbf{L}_{13} & \mathbf{0} & \mathbf{0}\\ \mathbf{G}_{11} & \mathbf{G}_{14} & \mathbf{U}_{11}^T & \mathbf{0}\\ \mathbf{G}_{12} & \mathbf{G}_{13} & \mathbf{U}_{12}^T & \mathbf{U}_{13}^T\end{bmatrix}\begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{D}_{11} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{D}_{12}\\ \mathbf{D}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{D}_{12} & \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{L}_{11}^T & \mathbf{L}_{12}^T & \mathbf{G}_{11}^T & \mathbf{G}_{12}^T\\ \mathbf{0} & \mathbf{L}_{13}^T & \mathbf{G}_{14}^T & \mathbf{G}_{13}^T\\ \mathbf{0} & \mathbf{0} & \mathbf{U}_{11} & \mathbf{U}_{12}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{U}_{13}\end{bmatrix}$$

Then considering the permutation matrix

$$\mathbf{P}_d = \begin{bmatrix}\mathbf{I}_{r_1} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{r_2} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{r_1} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{r_2}\end{bmatrix},$$

one can form $\mathbf{P}_d^T\overline{\Delta}\mathbf{P}_d = \begin{bmatrix}\mathbf{0} & \mathbf{D}_{11} & \mathbf{0} & \mathbf{0}\\ \mathbf{D}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{D}_{12}\\ \mathbf{0} & \mathbf{0} & \mathbf{D}_{12} & \mathbf{0}\end{bmatrix}$ and $\mathbf{P}_d^T\overline{L}\mathbf{P}_d = \begin{bmatrix}\mathbf{L}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{G}_{11} & \mathbf{U}_{11}^T & \mathbf{G}_{14} & \mathbf{0}\\ \mathbf{L}_{12} & \mathbf{0} & \mathbf{L}_{13} & \mathbf{0}\\ \mathbf{G}_{12} & \mathbf{U}_{12}^T & \mathbf{G}_{13} & \mathbf{U}_{13}^T\end{bmatrix}$.

Applying this process recursively changes $\overline{\Delta}$ into the desired block diagonal form. Then the transformation of $\overline{\mathbf{L}}$ will remain lower triangular if and only if all $\mathbf{G}_{14}$ matrices are zero: this means that $\mathbf{G}_1$ must be lower triangular in the first place.

Finding $\mathbf{G}_1$ lower triangular satisfying Equation (2), is an instance of Problem 1 for which the routine `trssyr2k` provides a solution.

Note that the actual permutation to transform $\left[\begin{smallmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{0} \end{smallmatrix}\right]$ into a 2×2-blocks diagonal matrix is a permutation matrix, $\mathbf{P}_i$, resulting from the one by one interleaving of the rows of $[\,\mathbf{I}_r\ \mathbf{0}\,]$ and $[\,\mathbf{0}\ \mathbf{I}_r\,]$. If $\mathbf{e}_i = [\,0\ldots0\ 1\ 0\ldots0\,]^T$ is the i-th canonical vector, then:

$$\mathbf{P}_i = \begin{bmatrix}\mathbf{e}_1 & \mathbf{e}_{r+1} & \mathbf{e}_2 & \mathbf{e}_{r+2} & \ldots & \mathbf{e}_r & \mathbf{e}_{2r}\end{bmatrix}. \tag{5}$$

Similarly the triangular factor of the factorization is thus a one by one interleaving of the rows of $[\,\mathbf{L}_1\ \mathbf{0}\,]$ and $[\,\mathbf{G}_1\ \mathbf{U}_1^T\,]$ as well as a one by one interleaving of the columns $\left[\begin{smallmatrix} \mathbf{L}_1 \\ \mathbf{G}_1 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} \mathbf{0} \\ \mathbf{U}_1^T \end{smallmatrix}\right]$, which overall remains triangular.

Finally, a call to Algorithm 2 produces a factorization for the remaining Z block and a final block rotation,

$$\begin{bmatrix}\mathbf{I}_{2r} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{m-r}\\ \mathbf{0} & \mathbf{I}_{n-r} & \mathbf{0}\end{bmatrix},$$

moves the intermediate zero rows and columns to the bottom right. The full algorithm is presented in details in Algorithm 3 (for zero or odd characteristic, the characteristic two case being presented afterwards in Section 4.3).

4.3 Characteristic two

The case of the characteristic two can be handled similarly, just computing the extra diagonal and updating after the `PLDUQ` decomposition, as sketched in Section 4.1.2. Indeed, the only issue is the division by 2 in `trssyr2k`, which is removed if the diagonal of $\mathbf{C}_1'$ is zero. Therefore, Algorithm 2 is unchanged, the block diagonal matrix just has lower symmetric antitriangular 2×2 blocks instead of only antidiagonal ones. The only few additional operations appear in Algorithm 3 (lines 4-9, 12, 16 and 21).

Then the tridiagonal form with symmetric antitriangular 2×2 blocks thus obtained by Algorithm 3 can be used to either reveal the rank profile matrix (via computing $\mathbf{\Psi}_\mathbf{D}$, the support matrix of $\mathbf{D}$, and the pivoting matrix $\mathcal{R} = \mathbf{P}\mathbf{\Psi}_\mathbf{D}\mathbf{P}^T$) or a PLDLTPT factorization, both at an extra linear cost in the dimension, as shown in Section 2.3.

Algorithm 3 Rank deficient and zero leading principal symmetric elimination

Require: $\mathbf{C} \in \mathbb{F}^{n\times n}$ symmetric and $\mathbf{B} \in \mathbb{F}^{m\times n}$.

Ensure: $\mathbf{P}$ permutation, $\mathbf{L}$ unit lower triangular, $\mathbf{D}$ block-diagonal, s.t. $\begin{bmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix} = \mathbf{PLDL}^T\mathbf{P}^T$.

1: Decompose $\mathbf{B} = \mathbf{P}_B \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{M}_1 \end{bmatrix} \begin{bmatrix} \mathbf{D}_1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_1 & \mathbf{V}_1 \end{bmatrix} \mathbf{Q}$ {PLDUQ }

2: let $r = \text{rank}(\mathbf{B})$ s.t. $\mathbf{L}_1$, $\mathbf{D}_1$ and $\mathbf{U}_1$ are $r \times r$.

3: $\mathbf{C}' \leftarrow \mathbf{QCQ}^T = \begin{bmatrix} \mathbf{C}'_1 & \mathbf{C}'_2 \\ \mathbf{C}'^T_2 & \mathbf{C}'_3 \end{bmatrix}$ where $\mathbf{C}'_1$ is $r \times r$ {perm }

4: **if** characteristic($\mathbb{F}$) = 2 **then**

5: **for** $i = 1$ **to** r **do**

6: $\Delta_{ii} = \left(\mathbf{C}'_1\right)_{ii} - \sum_{j=1}^{i-1} \Delta_{jj} \left(\mathbf{U}_1\right)^2_{j,i}$

7: **end for**

8: $\mathbf{C}'_1 \leftarrow \mathbf{C}'_1 - \mathbf{U}_1^T\Delta\mathbf{U}_1$ {syrdk $(\mathbf{C}'_1, \mathbf{U}_1, \Delta)$}

9: **end if**

10: Find $\mathbf{X}$ s.t. $\mathbf{X}^T\mathbf{U}_1 + \mathbf{U}_1^T\mathbf{X} = \mathbf{C}'_1$ {trssyr2k $(\mathbf{U}_1, \mathbf{C}'_1)$}

11: $\mathbf{G}_1 \leftarrow \mathbf{X}^T\mathbf{D}_1^{-1}$ {scal $(\mathbf{X}^T, \mathbf{D}_1^{-1})$}

12: **if** characteristic($\mathbb{F}$) = 2 **then** $\mathbf{X} \leftarrow \mathbf{X} + \Delta\mathbf{U}_1$ **end if** {dadd $(\mathbf{X}, \Delta, \mathbf{U}_1)$}

13: $\mathbf{C}''_2 \leftarrow \mathbf{C}'_2 - \mathbf{X}^T\mathbf{V}_1$ {trmm $(\mathbf{X}^T, \mathbf{V}_1)$}

14: $\mathbf{Y} \leftarrow \mathbf{U}_1^{-T}\mathbf{C}''_2$ {trsm $(\mathbf{U}_1^T, \mathbf{C}''_2)$}

15: $\mathbf{Z} \leftarrow \mathbf{C}'_3 - (\mathbf{Y}^T\mathbf{V}_1 + \mathbf{V}_1^T\mathbf{Y})$ {syrd2k $(\mathbf{C}'_3, \mathbf{Y}, \mathbf{V}_1)$}

16: **if** characteristic($\mathbb{F}$) = 2 **then** $\mathbf{Z} \leftarrow \mathbf{Z} - (\mathbf{V}_1^T\Delta\mathbf{V}_1)$ **end if** {syrdk $(\mathbf{Z}, \mathbf{V}_1, \Delta)$}

17: $\mathbf{G}_2 \leftarrow \mathbf{Y}^T\mathbf{D}_1^{-1}$ {scal $(\mathbf{Y}^T, \mathbf{D}_1^{-1})$}

18: Decompose $\mathbf{Z} = \mathbf{P}_3\mathbf{L}_3\mathbf{D}_3\mathbf{L}_3^T\mathbf{P}_3^T$ {Alg. 2}

19: $\mathbf{P} \leftarrow \begin{bmatrix} \mathbf{P}_B & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^T \end{bmatrix} \begin{bmatrix} \mathbf{P}_c & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{n-r} \end{bmatrix} \begin{bmatrix} \mathbf{P}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{m+n-2r} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{m+r} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_3 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{2r} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{m-r} \\ \mathbf{0} & \mathbf{I}_{n-r} & \mathbf{0} \end{bmatrix}$ {With $\mathbf{P}_c$ from (3), and $\mathbf{P}_i$ from (5)}

20: $\mathbf{L} \leftarrow \left[\begin{array}{c|c} \mathbf{P}_i^T \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{G}_1 & \mathbf{U}_1^T \end{bmatrix} \mathbf{P}_i & \\ \hline \begin{bmatrix} \mathbf{G}_2 & \mathbf{V}_1^T \\ \mathbf{M}_1 & \mathbf{0} \end{bmatrix} \mathbf{P}_i & \begin{matrix} \mathbf{L}_3 \\ \mathbf{0} \end{matrix} \end{array}\right]$

21: **if** characteristic($\mathbb{F}$) = 2 **then** $\mathbf{D} \leftarrow \left[\begin{array}{c|c} \mathbf{P}_i^T \begin{bmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \Delta \end{bmatrix} \mathbf{P}_i & \\ \hline & \mathbf{D}_3 \end{array}\right]$ **else** $\mathbf{D} \leftarrow \left[\begin{array}{c|c} \mathbf{P}_i^T \begin{bmatrix} \mathbf{0} & \mathbf{D}_1 \\ \mathbf{D}_1 & \mathbf{0} \end{bmatrix} \mathbf{P}_i & \\ \hline & \mathbf{D}_3 \end{array}\right]$ **end if**

5 BASE CASE ITERATIVE VARIANT

The recursion of Algorithm 2 should not be performed all the way to a dimension 1 in practice. For implementations over a finite field, it would induce an unnecessary large number of modular reductions and a significant amount of data movement for the permutations. Instead, we propose in Algorithm 4 an iterative algorithm computing a PLDLTPT revealing the rank profile matrix to be used as a base case in the recursion.

This iterative algorithm has the following features:

(1) it uses a pivot search minimizing the lexicographic order (following the characterization in [11]): if the diagonal element of the current row is 0, the pivot is chosen as the first non-zero element of the row, unless the row is all zero, in which case, it is searched in the following row;

(2) the pivot is permuted with cyclic shifts on the row and columns, so as to leave the precedence in the remaining rows and columns unchanged.

(3) the update of the unprocessed part in the matrix is delayed following the scheme of a Crout elimination schedule [8]. It does not only improve efficiency thanks to a better data locality, but it also reduces the amount of modular reductions, over a finite field, as shown for the unsymmetric case in [9].

We denote by $\rho_{i,n}$ the cyclic shift permutation of order n moving element i to the first position: $\rho_{i,n} = (i, 0, 1, \ldots, i-1, i+1, \ldots n-1)$. Indices are 0 based, index ranges are excluding their upper bound. For instance, $\mathbf{A}_{i,0..r}$ denotes the r first elements of the $i+1$st row of $\mathbf{A}$, and $\mathbf{A}_{0..r,0..r}$ is the 0-dimensional matrix when $r = 0$.

6 EXPERIMENTS

We now report on experiments of an implementation of these algorithms in the FFLAS-FFPACK library [17], dedicated to dense linear algebra over finite fields. We used the version committed under the reference e12a998 of the master branch. It was compiled with gcc-5.4 and was linked with the numerical library OpenBLAS-0.2.18. Experiments are run on a single core of an Intel Haswell i5-4690, @3.5GHz. Computation speed is normalized as *effective Gfops*, an estimate of the number of field operations that an algorithm with classic matrix arithmetic would perform per second, divided by the computation time. For a matrix of order n and rank r, we defined this as:

$$\text{Effective Gfops} = (r^3/3 + n^2r - r^2n)/(10^9 \times \text{time}).$$

Algorithm 4 SYTRF Crout iterative base case

Require: $\mathbf{A} \in \mathbb{F}^{n\times n}$ symmetric
Ensure: $\mathbf{P}$, a permutation, $\mathbf{L}$, unit lower triangular and $\mathbf{D}$, block diagonal, such that $\mathbf{A} = \mathbf{PLDL}^T\mathbf{P}^T$

Denote $\mathbf{W} = \mathbf{A}$ {the working matrix}
$r \leftarrow 0$; $\mathbf{D} \leftarrow \mathbf{0}$
for $i = 0..n$ **do**
  Here $\mathbf{W} = \begin{bmatrix} \mathbf{L} & & & \\ \mathbf{M} & \mathbf{0} & \mathbf{0} & \\ \mathbf{N} & \mathbf{0} & \mathbf{A}_{i..n,i} & \mathbf{A}_{i..n,i+1..n} \end{bmatrix}$ with $\mathbf{L} \in \mathbb{F}^{r\times r}$
  $\mathbf{v} \leftarrow \mathbf{N}_{0,0..r} \times \mathbf{D}^{-1}_{0..r,0..r}$
  $\mathbf{c} \leftarrow \mathbf{A}_{i..n,i} - \mathbf{N} \times \mathbf{v}^T$
  if $\mathbf{c} = 0$ **then** Loop to next iteration **end if**
  Let j be the smallest index such that $x = c_j \neq 0$
  Denote $\mathbf{c} = \begin{bmatrix} \mathbf{0} & x & \mathbf{k} \end{bmatrix}^T$
  if $j = 0$ **then**
    $\begin{bmatrix} \mathbf{M} \\ \mathbf{N} \end{bmatrix} \leftarrow \rho_{j,n-r} \times \begin{bmatrix} \mathbf{M} \\ \mathbf{N} \end{bmatrix}$
    $\mathbf{W}_{r..n,r} \leftarrow x^{-1} \times \rho_{j,n-r} \times \begin{bmatrix} 0 \\ \mathbf{c} \end{bmatrix}$
    $\mathbf{P} \leftarrow \mathbf{P} \times \rho^T_{j,n-r}$; $\mathbf{D}_{r,r} \leftarrow x$; $r \leftarrow r+1$
  else
    $\mathbf{w} \leftarrow \mathbf{N}_{j,0..r} \times \mathbf{D}^{-1}_{0..r,0..r}$
    $\mathbf{d} \leftarrow \mathbf{A}_{i..n,j+i} - \mathbf{N} \times \mathbf{w}^T (= \begin{bmatrix} \mathbf{0} & x & \mathbf{g} & y & \mathbf{h} \end{bmatrix}^T)$
    Here $\mathbf{W} = \begin{bmatrix} \mathbf{L} & & & & & \\ \mathbf{M} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ & \mathbf{0} & \mathbf{0} & \mathbf{0} & x & \mathbf{k}^T \\ \mathbf{N} & \mathbf{0} & \mathbf{0} & \mathbf{F} & \mathbf{g} & \mathbf{J}^T \\ & \mathbf{0} & x & \mathbf{g}^T & y & \mathbf{h}^T \\ & \mathbf{0} & \mathbf{k} & \mathbf{J} & \mathbf{h} & * \end{bmatrix}$
    if characteristic($\mathbb{F}$) = 2 **then**
      $y' \leftarrow 0$; $\mathbf{h}' \leftarrow \mathbf{h} - yx^{-1}\mathbf{k}$
      $\mathbf{D}_{r..r+2,r..r+2} \leftarrow \begin{bmatrix} 0 & x \\ x & y \end{bmatrix}$
    else
      $y' \leftarrow y/2$; $\mathbf{h}' \leftarrow \mathbf{h} - y'x^{-1}\mathbf{k}$
      $\mathbf{D}_{r..r+2,r..r+2} \leftarrow \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix}$
    end if
    Perform cyclic symmetric row and column rotations to bring W to the form $\mathbf{W} = \left[\begin{array}{ccc|ccc} \mathbf{L} & & & & & \\ \mathbf{n}' & 1 & & & & \\ & x^{-1}y' & 1 & & & \\ \hline \mathbf{M} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{N}' & x^{-1}\mathbf{g} & \mathbf{0} & \mathbf{0} & \mathbf{F} & \mathbf{J}^T \\ & x^{-1}\mathbf{h}' & x^{-1}\mathbf{k} & \mathbf{0} & \mathbf{J} & * \end{array}\right]$
    Update $\mathbf{P}$ accordingly
    $r \leftarrow r+2$
  end if
end for

All experiments are over the 23-bit finite field $\mathbb{Z}/8388593\mathbb{Z}$ (timings with other primes of similar size are similar).

Figures 1 and 2 compare the computation speed of the pure recursive algorithm, the base case algorithm and a cascade of these two, with a threshold set to its optimum value from experiments on this machine. Remark that the pure recursive variant performs rather well with generic rank profile matrices, while matrices with uniformly random rank profile matrix make this variant very slow, due to an excessive amount of pivoting. As expected, the base case Crout variant speeds up these instances for small dimensions, but then its performance stagnates on large dimensions, due to poor cache efficiency. Lastly the cascade algorithm combines the benefits of the two variants and therefore performs best in all settings. We here used a threshold $n = 128$ for the experiments with random RPM matrices, but of only $n = 48$ for generic rank profile matrices, since the recursive variant becomes competitive much earlier. In most cases, the rank profile structure of given matrices is unknown a priori, making the setting of this threshold speculative. One could instead implement an introspective strategy, updating the threshold from experimenting with running instances.

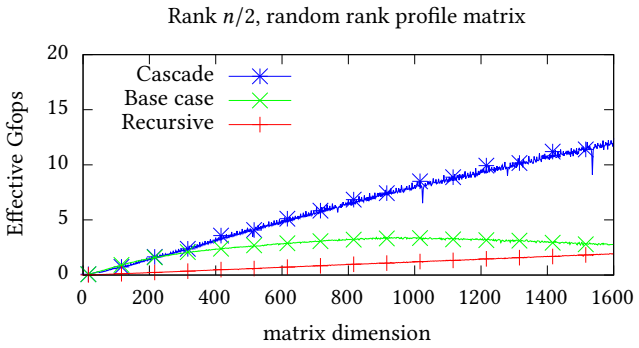


Figure 1: RPM revealing PLDLTPT: base case, pure recursive and cascading variants. Matrices with rank half the dimension and random RPM.

Table 1 compares the computation time of the finite field symmetric decomposition algorithm with several other routines: the unsymmetric elimination over a finite field (running the PLUQ algorithm of [11]) and the numerical elimination routines in double precision of LAPACK [2] provided by OpenBLAS: `dgetrf` (LU decomposition), and 3 variants of `dsytrf` [18] (symmetric LDLT decomposition with pivoting). These experiments first confirm a speed-up factor of about 2 between the symmetric and unsymmetric case over a finite field, which is the expected gain, looking at the constant in the time complexity. Note that on large instances, the PLUQ elimination performs better with random RPM instances than generic rank profiles, contrarily to the LDLT routine. This is due to the lesser amount of arithmetic operations when the RPM is random (some intermediate sub-matrices being rank deficient). On the other hand, these matrices generate more off-diagonal pivots, which cause more pivoting in LDLT than in PLUQ, explaining the slowdown for the symmetric case. On small dimensions, the numercial routines perform best as the quadratic modular reductions is penalizing the routines over a finite field. However, this is compensated by use of Strassen's algorithm, making our implementation outperform LAPACK's best `dsytrf` for larger dimension, even when the rank profile is not generic.

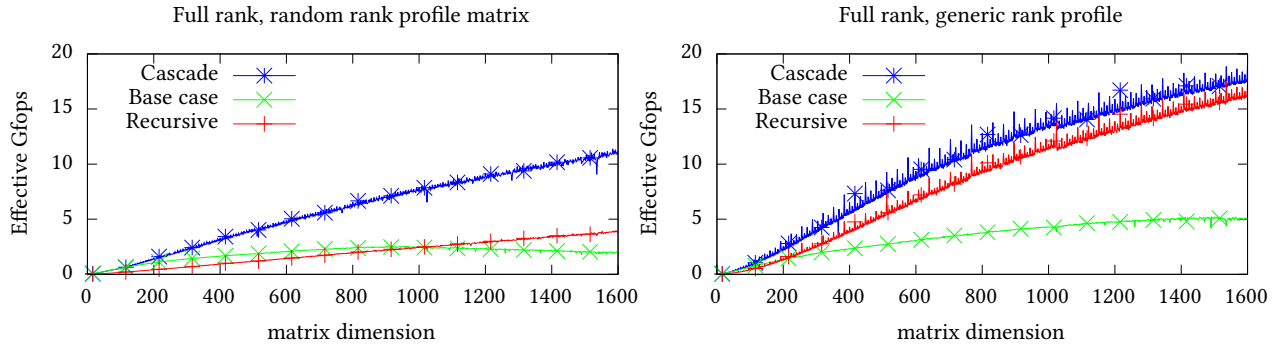


Figure 2: RPM revealing PLDLTPT. Matrices with full rank and random RPM (left) or generic rank profile (right).

n	Gen. rank prof. $r = n$				Gen. rank prof. $r = n$		Random RPM $r = n$		Random RPM $r = n/2$	
	dgetrf	dsytrf	dsytrf_rk	dsytrf_aa	PLUQ	LDLT	PLUQ	LDLT	PLUQ	LDLT
100	1.17e-04	1.31e-04	1.37e-04	9.21e-05	4.64e-04	3.23e-04	5.59e-04	5.22e-04	3.20e-04	3.80e-04
200	3.73e-04	4.39e-04	5.51e-04	4.48e-04	1.87e-03	8.80e-04	2.58e-03	1.59e-03	1.58e-03	1.33e-03
500	3.31e-03	3.78e-03	4.88e-03	3.87e-03	1.73e-02	5.21e-03	2.83e-02	9.85e-03	1.88e-02	7.92e-03
1000	2.19e-02	2.09e-02	2.58e-02	2.05e-02	8.84e-02	2.30e-02	9.95e-02	4.01e-02	6.27e-02	3.18e-02
2000	0.145	0.127	0.154	0.127	0.438	0.127	0.490	0.191	0.274	0.150
5000	2.005	1.604	1.871	1.598	3.904	1.591	3.849	1.744	2.431	1.294
10000	14.948	11.981	13.396	12.008	24.115	10.904	23.985	11.209	14.775	7.894

Table 1: Comparing computation time (s) of numerical routines with the symmetric (LDLT) and unsymmetric (PLUQ) triangular decompositions. Matrices with rank r, generic rank profile or rank profile matrix uniformly random.

ACKNOWLEDGMENT

We thank the referees for their valuable feedback and pointing out the recent work of [18].

REFERENCES

[1] J. O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT Numerical Mathematics*, 11(3):233–242, Sep 1971. doi:10.1007/BF01931804.

[2] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et al. *LAPACK Users' guide*. SIAM, 1999. URL: http://www.netlib.org/lapack/lug/lapack_lug.html.

[3] M. Baboulin, D. Becker, and J. Dongarra. A Parallel Tiled Solver for Dense Symmetric Indefinite Systems on Multicore Architectures. In *IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 14–24. IEEE, May 2012. doi:10.1109/IPDPS.2012.12.

[4] G. Ballard, D. Becker, J. Demmel, J. Dongarra, A. Druinsky, I. Peled, O. Schwartz, S. Toledo, and I. Yamazaki. Communication-Avoiding Symmetric-Indefinite Factorization. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1364–1406, Jan. 2014. doi:10.1137/130929060.

[5] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, 1977. doi:10.2307/2005787.

[6] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, Dec. 1971. doi:10.1137/0708060.

[7] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM TOMS*, 16(1):1–17, Mar. 1990. doi:10.1145/77626.79170.

[8] J. J. Dongarra, L. S. Duff, D. C. Sorensen, and H. A. V. Vorst. *Numerical Linear Algebra for High Performance Computers*. SIAM, 1998.

[9] J.-G. Dumas, T. Gautier, C. Pernet, J.-L. Roch, and Z. Sultan. Recursion based parallelization of exact dense linear algebra routines for gaussian elimination. *Parallel Computing*, 57:235 – 249, 2016. doi:10.1016/j.parco.2015.10.003.

[10] J.-G. Dumas, C. Pernet, and Z. Sultan. Computing the rank profile matrix. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '15, pages 149–156, New York, NY, USA, 2015. ACM. doi:10.1145/2755996.2756682.

[11] J.-G. Dumas, C. Pernet, and Z. Sultan. Fast computation of the rank profile matrix and the generalized Bruhat decomposition. *Journal of Symbolic Computation*, 83:187–210, Nov.–Dec. 2017. doi:10.1016/j.jsc.2016.11.011.

[12] E. Elmroth, F. G. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):3–45, 2004. doi:10.1137/S0036144503428693.

[13] E. L. Kaltofen, M. Nehring, and B. D. Saunders. Quadratic-time certificates in linear algebra. In A. Leykin, editor, *ISSAC'2011, Proceedings of the 2011 ACM International Symposium on Symbolic and Algebraic Computation, San Jose, California, USA*, pages 171–176. ACM Press, New York, June 2011. doi:10.1145/1993886.1993915.

[14] B. Parlett and J. K. Reid. On the solution of a system of linear equations whose matrix is symmetric but not definite. *BIT*, 10(3):386–397, 1970. doi:10.1007/BF01934207.

[15] M. Rozložník, G. Shklarski, and S. Toledo. Partitioned triangular tridiagonalization. *ACM Trans. Math. Softw.*, 37(4):38:1–38:16, Feb. 2011. doi:10.1145/1916461.1916462.

[16] G. Shklarski and S. Toledo. Blocked and recursive algorithms for triangular tridiagonalization. 2007. URL: http://www.cs.tau.ac.il/~stoledo/Bib/Pubs/ShklarskiToledo-Aasen.pdf.

[17] The FFLAS-FFPACK group. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package*, 2018. v2.3.2. https://github.com/linbox-team/fflas-ffpack.

[18] I. Yamazaki and J. Dongarra. Aasen's symmetric indefinite linear solvers in LAPACK. Technical Report 294, LAPACK Working Note, Dec. 2017. URL: http://www.netlib.org/lapack/lawnspdf/lawn294.pdf.

Journal of Symbolic Computation 85 (2018) 224–246

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Time and space efficient generators for quasiseparable matrices ☆

Clément Pernet [a], Arne Storjohann [b]

[a] Université Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, ÉNS de Lyon, Université de Lyon, Laboratoire de l'Informatique du Parallélisme, France
[b] David R. Cheriton School of Computer Science, University of Waterloo, Ontario, N2L 3G1, Canada

ARTICLE INFO

Article history:
Received 30 November 2016
Accepted 1 May 2017
Available online 17 July 2017

Keywords:
Quasiseparable
Hierarchically semiseparable
Rank profile matrix
Generalized Bruhat decomposition
Fast matrix arithmetic

ABSTRACT

The class of quasiseparable matrices is defined by the property that any submatrix entirely below or above the main diagonal has small rank, namely below a bound called the order of quasiseparability. These matrices arise naturally in solving PDE's for particle interaction with the Fast Multi-pole Method (FMM), or computing generalized eigenvalues. From these application fields, structured representations and algorithms have been designed in numerical linear algebra to compute with these matrices in time linear in the matrix dimension and either quadratic or cubic in the quasiseparability order. Motivated by the design of the general purpose exact linear algebra library LinBox, and by algorithmic applications in algebraic computing, we adapt existing techniques introduce novel ones to use quasiseparable matrices in exact linear algebra, where sub-cubic matrix arithmetic is available. In particular, we will show, the connection between the notion of quasiseparability and the rank profile matrix invariant, that we have introduced in 2015. It results in two new structured representations, one being a simpler variation on the hierarchically semiseparable storage, and the second one exploiting the generalized Bruhat decomposition. As a consequence, most basic operations, such as computing the quasiseparability orders, applying a vector, a block vector, multiplying two quasiseparable matrices together, inverting a quasiseparable

☆ This research was partly supported by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

E-mail addresses: clement.pernet@imag.fr (C. Pernet), astorjoh@cs.uwaterloo.ca (A. Storjohann).
URLs: http://ljk.imag.fr/membres/Clement.Pernet/ (C. Pernet), https://cs.uwaterloo.ca/~astorjoh/ (A. Storjohann).

http://dx.doi.org/10.1016/j.jsc.2017.07.010

matrix, can be at least as fast and often faster than previous existing algorithms.

1. Introduction

We consider the class of quasiseparable matrices, defined by a bounding condition on the ranks of the submatrices in their lower and upper triangular parts. These structured matrices originate mainly from two distinct application fields: computing generalized eigenvalues (Gohberg et al., 1985; Eidelman and Gohberg, 1999), and solving partial differential equations for particule simulation with the fast multipole method (Carrier et al., 1988). This class also arise naturally, as it includes the closure under inversion of the class of banded matrices. Among the several definitions used in the literature, we will use that of Eidelman and Gohberg (1999) for the class of quasiseparable matrices.

Definition 1. An $n \times n$ matrix M is (r_L, r_U)-quasiseparable if its strictly lower and upper triangular parts satisfy the following low rank structure: for all $1 \leq k \leq n-1$,

$$\operatorname{rank}(\mathsf{M}_{k+1..n,1..k}) \leq r_L, \tag{1}$$

$$\operatorname{rank}(\mathsf{M}_{1..k,k+1..n}) \leq r_U. \tag{2}$$

The values r_L and r_U are the quasiseparable orders of M.

Other popular classes of structured matrices like Toeplitz, Vandermonde, Cauchy, Hankel matrices and their block versions, enjoy a unified description through the powerful notion of displacement rank (Kailath et al., 1979). Consequently they benefit from space efficient representations (linear in the dimension n and in the displacement rank s), and time efficient algorithms to apply them to a vector, compute their inverse and solve linear systems: most operations have been reduced to polynomial arithmetic (Pan, 1990; Bini and Pan, 1994), and by incorporating fast matrix algebra, this cost has been reduced from $O^{\sim}(s^2 n)$ to $O^{\sim}(s^{\omega-1} n)$ by Bostan et al. (2008) (assuming that two $n \times n$ matrices can be multiplied in $O(n^{\omega})$ for $2.3728639 \leq \omega \leq 3$ Le Gall, 2014).

However quasiseparable matrices do not fit in the framework of rank displacement structures. Taking advantage of the low rank properties, mainly two types of structured representations have been developed together with corresponding dedicated algorithms to perform common linear algebra operations: the quasiseparable generators of Eidelman and Gohberg (1999); Vandebril et al. (2005; 2007), their generalization for finite block matrices by Eidelman and Gohberg (2005), that coincides with the sequentially semiseparable (SSS) representation of Chandrasekaran et al. (2005) and the hierarchically semiseparable representations (HSS) of Chandrasekaran et al. (2006); Xia et al. (2010). We refer to Vandebril et al. (2005), Vandebril et al. (2007) and Xia et al. (2010) for a broad bibliographic overview on the topic. Note also the alternative approach of Givens and unitary weights in Delvaux and Van Barel (2007).

Sequentially semiseparable representation The sequentially semiseparable representation used by Eidelman and Gohberg (1999); Vandebril et al. (2005; 2007); Eidelman et al. (2005); Boito et al. (2016) for a matrix M, consists of $(n-1)$ pairs of vectors $\mathsf{p}(i), \mathsf{q}(i)$ of size r_L, $(n-1)$ pairs of vectors $\mathsf{g}(i), \mathsf{h}(i)$ of size r_U, $n-1$ matrices $\mathsf{a}(i)$ of dimension $r_L \times r_L$, and $n-1$ matrices $\mathsf{b}(i)$ of dimension $r_U \times r_U$ and n scalars $d(i)$ such that

$$\mathsf{M}_{i,j} = \begin{cases} \mathsf{p}(i)^T \mathsf{a}_{ij}^{>} \mathsf{q}(j), & 1 \leq j < i \leq n \\ d(i), & 1 \leq i = j \leq n \\ \mathsf{g}(i)^T \mathsf{b}_{ij}^{<} \mathsf{h}(j), & 1 \leq i < j \leq n \end{cases}$$

where

$$\mathsf{a}^{>}_{ij} = \mathsf{a}(i-1)\ldots\mathsf{a}(j+1) \text{ for } j > i+1, \qquad \mathsf{a}_{j+1,j} = 1,$$
$$\mathsf{b}^{<}_{ij} = \mathsf{b}(i+1)\ldots\mathsf{b}(i-1) \text{ for } i > j+1, \qquad \mathsf{b}_{i,i+1} = 1.$$

For $s = \max(r_L, r_U)$, this representation, of size $O(n(r_L^2 + r_U^2)) = O(s^2 n)$ makes it possible to apply a vector in $O(s^2 n)$ field operations, multiply two quasiseparable matrices in time $O(s^3 n)$ and also compute the inverse of a strongly regular matrix in time $O(s^3 n)$ (Eidelman and Gohberg, 1999). Note that the inefficiency in size for this representation can be mitigated using the blocked version of this representation of Eidelman and Gohberg (2005).

The Hierarchically Semiseparable representation The Hierarchically Semiseparable representation was introduced in Chandrasekaran et al. (2006) and is related to the structure used in the Fast Multipole Method (Carrier et al., 1988). It is based on the splitting of the matrix in four quadrants, the use of rank revealing factorizations of its off-diagonal quadrants and applying the same scheme recursively on the diagonal blocks. A further compression is applied to represent all off-diagonal blocks as linear combinations (called translation operators) of blocks of a finer recursive order. While the space and time complexity of the HSS representation is depending on numerous parameters, the analysis in Chandrasekaran et al. (2006) seem to indicate that the size of an HSS representation is $O(sn)$, it can be applied to a vector in linear time in its size, and linear systems can be solved in $O(s^2 n)$. For the product of two HSS matrices, we could not find any better estimate than $O(s^3 n)$ given by Sheng et al. (2007).

Context and motivation The motivation here is to propose simplified and improved representations of quasiseparable matrices (in space and time). Our approach does not focus on numerical stability for the moment. Our first motivation is indeed to use these structured matrices in computer algebra where computing e.g. over a finite field or over multiprecision integers and rationals does not lead to any numerical instability. Hence we will assume throughout the paper that any Gaussian elimination algorithm mentioned has the ability to reveal ranks. In numerical linear algebra, a special care need to be taken for the pivoting of LU decompositions (Hwang et al., 1992; Pan, 2000), and QR or SVD decompositions are often preferred (Chan, 1987; Chandrasekaran and Ipsen, 1994). Part of the methods presented here, namely that of Section 5, rely on an arbitrary rank revealing matrix factorization and can therefore be applied to a setting with numerical instability. In the contrary, Section 6 relies on a class of Gaussian elimination algorithm that reveal the rank profile matrix, hence applying it to numerical setting is future work. This study is motivated by the design of new algorithms on polynomial matrices over a finite field, where quasiseparable matrices naturally occur, and more generally by the framework of the `LinBox` library (The LinBox Group, 2016) for black-box exact linear algebra.

Contribution This paper presents in further details and extends the results of Pernet (2016), while also fixing a mistake.[1] It proposes two new structured representations for quasiseparable matrices, a Recursive Rank Revealing (RRR) representation that can be viewed as a simplified version of the HSS representation of Chandrasekaran et al. (2006), and a representation based on the generalized Bruhat decomposition, which we name Compact Bruhat (CB) representation. The later one, is made possible by the connection that we make between the notion of quasiseparability and a matrix invariant, the rank profile matrix, that we introduced in Dumas et al. (2015) and applied to the generalized Bruhat decomposition in Dumas et al. (2016). More precisely, we show that the lower and upper triangular parts of a quasiseparabile matrix have a Generalized Bruhat decompositions off of which many coefficients can be shaved. The resulting structure of these decompositions allows to handle them within memory footprint and time complexity that does not depend on the rank but on the quasiseparable order (which can be arbitrarily lower). These two representations use respectively a

[1] Equation (9) in Pernet (2016) is missing the Left operators. The resulting algorithms are incorrect. This is fixed in section 6.2.

Table 1
Comparing the size and time complexities for basic operations of the proposed RRR and CB representations with the existing one SSS and HSS on an $n \times n$ quasiseparable matrix of order s.

	SSS	HSS	RRR	CB
Size	$O(s^2n)$	$O(sn)$	$O(sn \log \frac{n}{s})$	$O(sn)$
Construction	$O(s^2n^2)$	$O(sn^2)$	$O(s^{\omega-2}n^2)$	$O(s^{\omega-2}n^2)$
QSxVec	$O(s^2n)$	$O(sn)$	$O(sn \log \frac{n}{s})$	$O(sn)$
QSxTS	$O(s^3n)$	$O(s^2n)$	$O(s^{\omega-1}n \log \frac{n}{s})$	$O(s^{\omega-1}n)$
QSxQS	$O(s^3n)$	$O(s^3n)$	$O(s^{\omega-1}n \log^2 \frac{n}{s})$	$O(s^{\omega-2}n^2)$
LinSys	$O(s^3n)$	$O(s^2n)$	$O(s^{\omega-1}n \log^2 \frac{n}{s})$	

space $O(sn \log \frac{n}{s})$ (RRR) and $O(sn)$ (CB), hence improving over that of the SSS, $O(s^2n)$, and matching that of the HSS representation, $O(sn)$.

The complexity of applying a vector remains linear in the size of the representations. The main improvement in these two representations is in the complexity of applying them to matrices and computing the matrix inverse, where we replace by $s^{\omega-1}$ the s^3 factor of the SSS or the s^2 factor of the HSS representations.[2] Table 1 compares the two proposed representations with the SSS and the HSS in their the size, and the complexity of the main basic operations.

Outline Section 2 defines and recalls some preliminary notions on left triangular matrices and the rank profile matrix, that will be used in Section 3 and 6. Using the strong connection between the rank profile matrix and the quasiseparable structure, we first propose in Section 3 an algorithm to compute the quasiseparability orders (r_L, r_U) of any dense matrix in $O(n^2 s^{\omega-2})$ where $s = \max(r_L, r_U)$. Section 4 then describes the two proposed structured representations for quasiseparable matrices: the Recursive Rank Revealing representation (RRR), a simplified HSS representation based on a binary tree of rank revealing factorizations, and the Compact Bruhat representation (CB), based on the intermediate Bruhat representation. Section 5 then presents algorithms to compute an RRR representation, and perform the most common operations with it: applying a vector, a tall and skinny matrix, multiplying two quasiseparable matrices in RRR representation, and computing the inverse of a strongly regular RRR matrix. Section 6 presents algorithms to compute a Compact Bruhat representation, and multiply it with a vector, a tall and skinny matrix or a dense matrix.

Notations Throughout the paper, $\mathsf{A}_{i..j,k..l}$ will denote the sub-matrix of A of row indices between i and j and column indices between k and l. The matrix of the canonical basis, with a one at position (i, j) will be denoted by $\Delta^{(i,j)}$. We will denote the identity matrix of order n by I_n, the unit antidiagonal of dimension n by J_n and the zero matrix of dimension $m \times n$ by $\mathsf{0}_{m\times n}$.

2. Preliminaries

2.1. Left triangular matrices

We will make intensive use of matrices with non-zero elements only located above the main anti-diagonal. We will refer to these matrices as left triangular, to avoid any confusion with upper triangular matrices.

Definition 2. An $m \times n$ matrix A is left triangular if $\mathsf{A}_{i,j} = 0$ for all $i > n - j$.

[2] Note that most complexities for SSS and HSS in the literature are given in the form $O(n^2)$ or $O(n)$, considering the parameter s as a constant. The estimates given here, with the exponent in s, can be found in the proofs of the related papers or easily derived from the algorithms.

The left triangular part of a matrix A, denoted by Left(A) will refer to the left triangular matrix extracted from it. We will need the following property on the left triangular part of the product of a matrix by a triangular matrix.

Lemma 3. *Let* $\mathsf{A} = \mathsf{BU}$ *be an* $m \times n$ *matrix where* U *is* $n \times n$ *upper triangular. Then Left*(A) = *Left*(*Left*(B)U).

Proof. Let $\bar{\mathsf{A}} = \text{Left}(\mathsf{A}), \bar{\mathsf{B}} = \text{Left}(\mathsf{B})$. For $j \leq n - i$, we have $\bar{\mathsf{A}}_{i,j} = \sum_{k=1}^{n} \mathsf{B}_{i,k} \cdot \mathsf{U}_{k,j} = \sum_{k=1}^{j} \mathsf{B}_{i,k} \cdot \mathsf{U}_{k,j}$ as U is upper triangular. Now for $k \leq j \leq n - i$, $\mathsf{B}_{i,k} = \bar{\mathsf{B}}_{i,k}$, which proves that the left triangular part of A is that of Left(B)U. □

Applying Lemma 3 on A^T yields Lemma 4.

Lemma 4. *Let* $\mathsf{A} = \mathsf{LB}$ *be an* $m \times n$ *matrix where* L *is* $m \times m$ *lower triangular. Then Left*(A) = *Left*(L*Left*(B)).

Lastly, we will extend the notion of order of quasiseparability to left triangular matrices, in the natural way: the order of left quasiseparability is the maximal rank of any leading $k \times (n - k)$ sub-matrix. When no confusion may occur, we will abuse the definition and simply call it the order of quasiseparability.

2.2. PLUQ decomposition

We recall that for any $m \times n$ matrix A of rank r, there exist a PLUQ decomposition $\mathsf{A} = \mathsf{PLUQ}$ where P is an $m \times m$ permutation matrix, Q is an $n \times n$ permutation matrix, L is an $m \times r$ unit lower triangular matrix, and U is an $r \times n$ upper triangular matrix. matrix. It is not unique, but once the permutation matrices P and Q are fixed, the triangular factors L and U are unique, since the matrix $\mathsf{P}^T\mathsf{A}\mathsf{Q}^T$ has generic rank profile and therefore has a unique LU decomposition.

2.3. The rank profile matrix

We will use a matrix invariant, introduced in Dumas et al. (2015, Theorem 1) that summarizes the information on the ranks of any leading sub-matrices of a given input matrix.

Definition 5. (Dumas et al., 2015, Theorem 1) The rank profile matrix of an $m \times n$ matrix A of rank r is the unique $m \times n$ matrix $\mathcal{R}_{\mathsf{A}}$, with only r non-zero coefficients, all equal to one, located on distinct rows and columns such that any leading sub-matrices of $\mathcal{R}_{\mathsf{A}}$ has the same rank as the corresponding leading sub-matrix in A.

This invariant can be computed in just one Gaussian elimination of the matrix A, at the cost of $O(mnr^{\omega-2})$ field operations (Dumas et al., 2015), provided some conditions on the pivoting strategy being used. It is obtained from the corresponding PLUQ decomposition as the product

$$\mathcal{R}_{\mathsf{A}} = \mathsf{P}\begin{bmatrix} \mathsf{I}_r & \\ & \mathsf{0}_{(m-r)\times(n-r)} \end{bmatrix}\mathsf{Q}.$$

Remark 6. The notion of rank profile matrix originates from Bruhat's matrix decomposition for non-singular matrices in Bruhat (1956), where uniqueness of this permutation was established. It has then been generalized in Tyrtyshnikov (1997) and Manthey and Helmke (2007) for all matrices and in Malaschonok (2010) with the LEU decomposition, where it appears as the E factor. The connection to rank profiles introduced in Dumas et al. (2015) was the motivation for its name.

We also recall in Theorem 7 an important property of such PLUQ decompositions revealing the rank profile matrix.

Property 7 *(Dumas et al., 2016, Th. 24, Dumas et al., 2013, Th. 1).* Let $\mathsf{A} = \mathsf{PLUQ}$, a PLUQ decomposition revealing the rank profile matrix of A. Then, $\mathsf{P}\begin{bmatrix}\mathsf{L} & 0_{m\times(m-r)}\end{bmatrix}\mathsf{P}^T$ is lower triangular and $\mathsf{Q}^T\begin{bmatrix}\mathsf{U}\\ 0_{(n-r)\times n}\end{bmatrix}\mathsf{Q}$ is upper triangular.

Lemma 8. *The rank profile matrix invariant is preserved by multiplication*

1. *to the left with an invertible lower triangular matrix,*
2. *to the right with an invertible upper triangular matrix.*

Proof. Let $\mathsf{B} = \mathsf{LA}$ for an invertible lower triangular matrix L. Then for any $i \le m$, $j \le n$, $\text{rank}(\mathsf{B}_{1..i,1..j}) = \text{rank}(\mathsf{L}_{1..i,1..i}\mathsf{A}_{1..i,1..j}) = \text{rank}(\mathsf{A}_{1..i,1..j})$. Hence $\mathcal{R}_\mathsf{B} = \mathcal{R}_\mathsf{A}$. □

3. Computing the orders of quasiseparability

Let M be an $n \times n$ matrix of which one wants to determine the quasiseparable orders (r_L, r_U). Let L and U be respectively the lower triangular part and the upper triangular part of M.

Multiplying on the left by J_n, the unit anti-diagonal matrix, inverses the row order while multiplying on the right by J_n inverses the column order. Hence both $\mathsf{J}_n\mathsf{L}$ and $\mathsf{U}\mathsf{J}_n$ are left triangular matrices. Remark that conditions (1) and (2) state that all leading $k \times (n-k)$ sub-matrices of $\mathsf{J}_n\mathsf{L}$ and $\mathsf{U}\mathsf{J}_n$ have rank no greater than r_L and r_U respectively. We will then use the rank profile matrix of these two left triangular matrices to find these parameters.

3.1. From a rank profile matrix

First, note that the rank profile matrix of a left triangular matrix is not necessarily left triangular. For example, the rank profile matrix of $\begin{bmatrix}1&1&0\\1&0&0\\0&0&0\end{bmatrix}$ is $\begin{bmatrix}1&0&0\\0&1&0\\0&0&0\end{bmatrix}$. However, only the left triangular part of the rank profile matrix is sufficient to compute the left quasiseparable orders.

Suppose for the moment that the left-triangular part of the rank profile matrix of a left triangular matrix is given (returned by a function LT-RPM). It remains to enumerate all leading $k \times (n-k)$ sub-matrices and find the one with the largest number of non-zero elements. Algorithm 1 shows how to compute the largest rank of all leading sub-matrices of such a matrix. Run on $\mathsf{J}_n\mathsf{L}$ and $\mathsf{U}\mathsf{J}_n$, it returns successively the quasiseparable orders r_L and r_U.

Algorithm 1 QS-order.

Require: A, an $n \times n$ matrix
Ensure: $\max\{\text{rank}(\mathsf{A}_{1..k,1..n-k}) : 1 \le k \le n-1\}$
  $\mathcal{R} \leftarrow$ LT-RPM(A) ▷ The left triangular part of the rank profile matrix of A
  rows $\leftarrow$ (False,...,False)
  cols $\leftarrow$ (False,...,False)
  for all (i, j) such that $\mathcal{R}_{i,j} = 1$ **do**
    rows[i] $\leftarrow$ True
    cols[j] $\leftarrow$ True
  end for
  $s, r \leftarrow 0$
  for $i = 1 \ldots n-1$ **do**
    if rows[i] **then** $r \leftarrow r+1$
    if cols[$n-i+1$] **then** $r \leftarrow r-1$
    $s \leftarrow \max(s, r)$
  end for
  return s

This algorithm runs in $O(n)$ provided that the rank profile matrix R is stored in a compact way, e.g. using a vector of r pairs of pivot indices $[(i_1, j_1), \ldots, (i_r, j_r)]$.

3.2. Computing the rank profile matrix of a left triangular matrix

We now deal with the missing component: computing the left triangular part of the rank profile matrix of a left triangular matrix.

3.2.1. From a PLUQ decomposition

A first approach is to run any Gaussian elimination algorithm that can reveal the rank profile matrix, as described in Dumas et al. (2015). In particular, the PLUQ decomposition algorithm of Dumas et al. (2013) computes the rank profile matrix of A in $O(n^2 r^{\omega-2})$ where $r = \text{rank}(\mathsf{A})$. However this estimate may be pessimistic as it does not take into account the left triangular shape of the matrix. Moreover, it does not depend on the left quasiseparable order s but on the rank r, which could be much higher.

Remark 9. The discrepancy between the rank r of a left triangular matrix and its quasiseparable order arises from the location of the pivots in its rank profile matrix. Pivots located near the top left corner of the matrix are shared by many leading sub-matrices, and are therefore likely to contribute to the quasiseparable order. On the other hand, pivots near the main anti-diagonal can be numerous, but do not add up to a large quasiseparable order. As an illustration, consider the two following extreme cases:

1. a matrix A with generic rank profile. Then the leading $r \times r$ sub-matrix of A has rank r and the quasiseparable order is $s = r$;
2. the matrix with $n-1$ ones immediately above the main anti-diagonal. It has rank $r = n-1$ but quasiseparable order 1.

Remark 9 indicates that in the unlucky cases when $r \gg s$, the computation should reduce to instances of smaller sizes, hence a trade-off should exist between, on one hand, the discrepancy between r and s, and on the other hand, the dimension n of the problems. All contributions presented in the remaining of the paper are based on such trade-offs.

3.2.2. A dedicated algorithm

In order to reach a complexity depending on s and not r, we adapt in Algorithm 2 the tile recursive algorithm of Dumas et al. (2013), so that the left triangular structure of the input matrix is preserved and can be used to reduce the amount of computation. In this algorithm, the input matrix is modified in-place, and comments keep track of its current value. In particular, the upper and lower triangular factors obtained after a PLUQ decomposition are stored one above the other on the same storage, which is represented by the notation $\left[\mathsf{L}\backslash\mathsf{U}\right]$.

Algorithm 2 does not assume that the input matrix is left triangular, as it will be called recursively with arbitrary matrices, but guarantees to return the left triangular part of the rank profile matrix. While the top left quadrant A_1 is eliminated using any PLUQ decomposition algorithm revealing the rank profile matrix, the top right and bottom left quadrants are handled recursively.

Theorem 10. *Given an $n \times n$ input matrix A with left quasiseparable order s, Algorithm 2 computes the left triangular part of the rank profile matrix of A in $O(n^2 s^{\omega-2})$ field operations.*

Proof. First remark that

$$\mathsf{P}_1 \begin{bmatrix} \mathsf{D} \\ \mathsf{F} \end{bmatrix} = \underbrace{\mathsf{P}_1 \begin{bmatrix} {\mathsf{L}_1}^{-1} & \\ -\mathsf{M}_1 {\mathsf{L}_1}^{-1} & \mathsf{I}_{n-r_1} \end{bmatrix} {\mathsf{P}_1}^T}_{\mathsf{L}} \mathsf{P}_1 \begin{bmatrix} \mathsf{B}_1 \\ \mathsf{B}_2 \end{bmatrix} = \mathsf{L}\mathsf{A}_2.$$

Hence

$$\mathsf{L}\begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_2 \end{bmatrix} = \mathsf{P}_1 \left[\begin{array}{cc|c} \mathsf{U}_1 & \mathsf{V}_1 & \mathsf{D} \\ 0 & & \mathsf{F} \end{array}\right] \mathsf{Q}_1.$$

Algorithm 2 LT-RPM: Left Triangular part of the Rank Profile Matrix.

Require: A: an $n \times n$ matrix
Ensure: $\mathcal{R}$: the left triangular part of the RPM of A

1: **if** $n = 1$ **then return** [0]
2: Split $\mathsf{A} = \begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_2 \\ \mathsf{A}_3 & \end{bmatrix}$ where A_3 is $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$
3: Compute a PLUQ decomposition $\mathsf{A}_1 = \mathsf{P}_1 \begin{bmatrix} \mathsf{L}_1 \\ \mathsf{M}_1 \end{bmatrix} \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \end{bmatrix} \mathsf{Q}_1$ revealing the RPM
4: $\mathcal{R}_1 \leftarrow \mathsf{P}_1 \begin{bmatrix} \mathsf{I}_{r_1} & \\ & 0 \end{bmatrix} \mathsf{Q}_1$ where $r_1 = \text{rank}(\mathsf{A}_1)$.
5: $\begin{bmatrix} \mathsf{B}_1 \\ \mathsf{B}_2 \end{bmatrix} \leftarrow \mathsf{P}_1{}^T \mathsf{A}_2$
6: $\begin{bmatrix} \mathsf{C}_1 & \mathsf{C}_2 \end{bmatrix} \leftarrow \mathsf{A}_3 \mathsf{Q}_1{}^T$ ▷ Here $\mathsf{A} = \left[\begin{array}{cc|c} \mathsf{L}_1\backslash\mathsf{U}_1 & \mathsf{V}_1 & \mathsf{B}_1 \\ \mathsf{M}_1 & 0 & \mathsf{B}_2 \\ \hline \mathsf{C}_1 & \mathsf{C}_2 & \end{array}\right]$.
7: $\mathsf{D} \leftarrow \mathsf{L}_1{}^{-1} \mathsf{B}_1$
8: $\mathsf{E} \leftarrow \mathsf{C}_1 \mathsf{U}_1{}^{-1}$
9: $\mathsf{F} \leftarrow \mathsf{B}_2 - \mathsf{M}_1 \mathsf{D}$
10: $\mathsf{G} \leftarrow \mathsf{C}_2 - \mathsf{E}\mathsf{V}_1$ ▷ Here $\mathsf{A} = \left[\begin{array}{cc|c} \mathsf{L}_1\backslash\mathsf{U}_1 & \mathsf{V}_1 & \mathsf{D} \\ \mathsf{M}_1 & 0 & \mathsf{F} \\ \hline \mathsf{E} & \mathsf{G} & \end{array}\right]$.
11: $\mathsf{H} \leftarrow \mathsf{P}_1 \begin{bmatrix} 0_{r_1 \times \frac{n}{2}} \\ \mathsf{F} \end{bmatrix}$
12: $\mathsf{I} \leftarrow \begin{bmatrix} 0_{r_1 \times \frac{n}{2}} & \mathsf{G} \end{bmatrix} \mathsf{Q}_1$
13: $\mathcal{R}_2 \leftarrow \text{LT} - \text{RPM}(\mathsf{H})$
14: $\mathcal{R}_3 \leftarrow \text{LT} - \text{RPM}(\mathsf{I})$
15: **return** $\mathcal{R} \leftarrow \begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 \\ \mathcal{R}_3 & \end{bmatrix}$

From Theorem 7, the matrix L is lower triangular and by Lemma 8 the rank profile matrix of $\begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_2 \end{bmatrix}$ equals that of $\mathsf{P}_1 \left[\begin{array}{cc|c} \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \end{bmatrix} \mathsf{Q}_1 & & \mathsf{D} \\ & 0 & \mathsf{F} \end{array}\right]$. Now as U_1 is upper triangular and non-singular, this rank profile matrix is in turn that of $\mathsf{P}_1 \left[\begin{array}{cc|c} \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \end{bmatrix} \mathsf{Q}_1 & & 0 \\ & 0 & \mathsf{F} \end{array}\right]$ and its left triangular part is $\begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 \end{bmatrix}$.

By a similar reasoning, $\begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_3 \end{bmatrix}^T$ is the left triangular part of the rank profile matrix of $\begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_3 \end{bmatrix}^T$, which shows that the algorithm is correct.

Let s_1 be the left quasiseparable order of H and s_2 that of I. The number of field operations required to run Algorithm 2 is

$$T(n, s) = \alpha r_1^{\omega-2} n^2 + T_{\text{LT-RPM}}(n/2, s_1) + T_{\text{LT-RPM}}(n/2, s_2)$$

for a positive constant α. We will prove by induction that $T(n, s) \leq 2\alpha s^{\omega-2} n^2$.

Again, since L is lower triangular, the rank profile matrix of LA_2 is that of A_2 and the quasiseparable orders of the two matrices are the same. Now H is the matrix LA_2 with some rows zeroed out, hence s_1, the quasiseparable order of H is no greater than that of A_2 which is less or equal to s. Hence $\max(r_1, s_1, s_2) \leq s$ and we obtain $T(n, s) \leq \alpha s^{\omega-2} n^2 + 4\alpha s^{\omega-2} (n/2)^2 = 2\alpha s^{\omega-2} n^2$. □

4. New structured representations for quasiseparable matrices

In order to introduce fast matrix arithmetic in the algorithms computing with quasiseparable matrices, we introduce in this section three new structured representations: the Recursive Rank Revealing (RRR) representation, the Bruhat representation, and finally its compact version, the Compact Bruhat (CB) representation.

4.1. The Recursive Rank Revealing representation

This a simplified version of the HSS representation. It uses in the same manner a recursive splitting of the matrix in a quad-tree, and each off-diagonal block at each recursive level is represented by a rank revealing factorization.

Definition 11 *(RR: Rank revealing factorization).* A rank revealing factorization (RR) of an $m \times n$ matrix A of rank r is a pair of matrices L and R of dimensions $m \times r$ and $r \times n$ respectively, such that $\mathsf{A} = \mathsf{LR}$.

For instance, a PLUQ decomposition is a rank revealing factorization. One can either store explicitly the two factors PL and UQ or only consider the factors L and U keeping in mind that permutations need to be applied on the left and on the right of the product.

Definition 12 *(RRR: Recursive Rank Revealing representation).* A recursive rank revealing (RRR) representation of an $n \times n$ quasiseparable matrix $\mathsf{A} = \begin{bmatrix} \mathsf{A}_{11} & \mathsf{A}_{12} \\ \mathsf{A}_{21} & \mathsf{A}_{22} \end{bmatrix}$ of order s is formed by a rank revealing factorization of A_{12} and A_{21} and applies recursively for the representation of A_{11} and A_{22}.

A Recursive Rank Revealing representation forms a binary tree where each node correspond to a diagonal block of the input matrix, and contains the Rank Revealing factorization of its off-diagonal quadrants.

If A is (r_L, r_U)-quasiseparable, then all off-diagonal blocks in its lower part have rank bounded by r_L, and their rank revealing factorizations take advantage of this low rank until a block dimension $n/2^k \approx r_L$ where a dense representation is used. The same applies for the upper triangular part with quasiseparable order r_U. This representation uses $O(sn \log \frac{n}{s})$ space where $s = \max(r_L, r_U)$.

4.2. The Bruhat representation

This structured representation is closely related to the notion of the rank profile matrix and the LEU decomposition of Malaschonok (2010). Contrarily to the RRR or the HSS representations, it is not depending on a specific recursive cutting of the matrix. For this representation, and its compact version that will be studied in section 4.3, the lower and the upper triangular parts are represented independently. We will therefore treat them in a unified way, showing how to represent a left triangular matrix. Recall that if L is lower triangular and U is upper triangular then both $\mathsf{J_n L}$ and $\mathsf{U J_n}$ are left triangular.

Given a left triangular matrix A of quasiseparable order s and a PLUQ decomposition of it, revealing its rank profile matrix R, the Bruhat generator consists in the three matrices

$$\mathcal{L} = \text{Left}(\mathsf{P}\begin{bmatrix}\mathsf{L} & 0\end{bmatrix}\mathsf{Q}), \tag{3}$$

$$\mathcal{R} = \text{Left}(\mathsf{R}), \tag{4}$$

$$\mathcal{U} = \text{Left}(\mathsf{P}\begin{bmatrix}\mathsf{U} \\ 0\end{bmatrix}\mathsf{Q}). \tag{5}$$

Lemma 13 shows that these three matrices suffice to recover the initial left triangular matrix.

Lemma 13. $\mathsf{A} = \mathit{Left}(\mathcal{L}\mathcal{R}^T\mathcal{U})$

Proof. $\mathsf{A} = \mathsf{P}\begin{bmatrix}\mathsf{L} & 0_{m\times(n-r)}\end{bmatrix}\mathsf{Q}\mathsf{Q}^T\begin{bmatrix}\mathsf{U} \\ 0_{(n-r)\times n}\end{bmatrix}\mathsf{Q}$. From Theorem 7, the matrix $\mathsf{Q}^T\begin{bmatrix}\mathsf{U} \\ 0\end{bmatrix}\mathsf{Q}$ is upper triangular and the matrix $\mathsf{P}\begin{bmatrix}\mathsf{L} & 0\end{bmatrix}\mathsf{P}^T$ is lower triangular. Applying Lemma 3 yields $\mathsf{A} = \text{Left}(\mathsf{A}) = \text{Left}(\mathcal{L}\mathsf{Q}^T\begin{bmatrix}\mathsf{U} \\ 0\end{bmatrix}\mathsf{Q}) = \text{Left}(\mathcal{L}\mathsf{R}^T\mathsf{P}\begin{bmatrix}\mathsf{U} \\ 0\end{bmatrix}\mathsf{Q})$, where $\mathsf{R} = \mathsf{P}\begin{bmatrix}\mathsf{I}_r & \\ & 0\end{bmatrix}\mathsf{Q}$. Then, as $\mathcal{L}\mathsf{R}^T$ is the matrix $\mathsf{P}\begin{bmatrix}\mathsf{L} & 0\end{bmatrix}\mathsf{P}^T$ with some coefficients zeroed out, it is lower triangular, hence applying again Lemma 4 yields

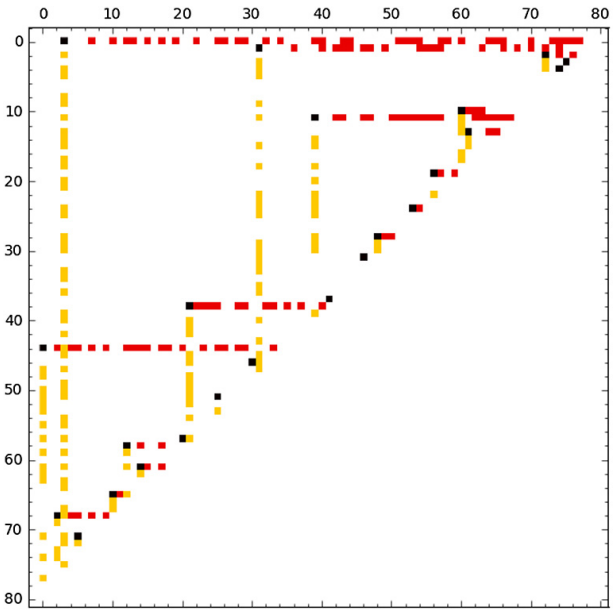


Fig. 1. Support of the $\mathcal{L}$ (yellow), $\mathcal{R}$ (black) and $\mathcal{U}$ (red) matrices of the Bruhat generator for a 80×80 left triangular matrix of quasiseparable order 5. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\mathsf{A} = \text{Left}(\mathcal{L}\mathsf{R}^T\mathcal{U}). \tag{6}$$

Consider any non-zero coefficient $e_{j,i}$ of R^T that is not in its the left triangular part, i.e. $j > n - i$. Its contribution to the product $\mathcal{L}\mathsf{R}^T$, is only of the form $\mathcal{L}_{k,j}e_{j,i}$. However the leading coefficient in column j of $\mathsf{P}\begin{bmatrix}\mathsf{L} & 0\end{bmatrix}\mathsf{Q}$ is precisely at position (i, j). Since $i > n - j$, this means that the j-th column of $\mathcal{L}$ is all zero, and therefore $e_{i,j}$ has no contribution to the product. Hence we finally have $\mathsf{A} = \text{Left}(\mathcal{L}\mathcal{R}^T\mathcal{U})$. □

We now analyze the space required by this generator.

Lemma 14. *Consider an $n \times n$ left triangular rank profile matrix R with quasiseparable order s. Then a left triangular matrix L all zero except at the positions of the pivots of R and below these pivots, does not contain more than $s(n - s)$ non-zero coefficients.*

Proof. Let $p(k) = \text{rank}(\mathsf{R}_{1..k,1..n-k})$. The value $p(k)$ indicates the number of non-zero columns located in the $k \times n - k$ leading sub-matrix of L and is therefore an upper bound on the number of non-zero elements in row k of L. Consequently the sum $\sum_{k=1}^{n-1} p(k)$ is an upper bound on the number of non-zero coefficients in L. Since $p(k) \leq s$, it is bounded by sn. More precisely, there is no more than k pivots in the first k columns and the first k rows, hence $p(k) \leq k$ and $p(n - k) \leq k$ for $k \leq s$. The bound becomes $s(s + 1) + (n - 2s - 1)s = s(n - s)$. □

Corollary 15. *The Bruhat generator $(\mathcal{L}, \mathcal{R}, \mathcal{U})$ uses $2s(n - s)$ field coefficients and $O(n)$ additional indices to represent a left triangular matrix.*

Proof. The leading column elements of $\mathcal{L}$ are located at the pivot positions of the left triangular rank profile matrix $\mathcal{R}$. Lemma 14 can therefore be applied to show that this matrix occupies no more than $s(n - s)$ non-zero coefficients. The same argument applies to the matrix $\mathcal{U}$. □

Fig. 1 illustrates this generator on a left triangular matrix of quasiseparable order 5. As the supports of $\mathcal{L}$ and $\mathcal{U}$ are disjoint, the two matrices can be shown on the same left triangular matrix. The pivots of $\mathcal{R}$ (black) are the leading coefficients of every non-zero row of $\mathcal{U}$ and non-zero column of $\mathcal{L}$.

Corollary 16. *Any (r_L, r_U)-quasiseparable matrix of dimension $n \times n$ can be represented by a generator using no more than $2n(r_L + r_U) + n - 2(r_L^2 - 2r_U^2)$ field elements and $2(r_L + r_U)$ indices.*

Proof. This estimate is obtained as the space required for the Bruhat representation of the upper and lower triangular parts of the matrix, with n coefficients for the main diagonal. The $2(r_L + r_U)$ indices correspond to the storage of the pivot positions of the two rank profile matrices. □

4.3. The compact Bruhat representation

The scattered structure of the Bruhat generator makes it not amenable to the use of fast matrix arithmetic. We therefore propose here a compact variation on it, called the compact Bruhat, that will be used to derive algorithms taking advantage of fast matrix multiplication. This structured representation relies on the generalized Bruhat decomposition described in Manthey and Helmke (2007), thanks to the connection with the rank profile matrix made in Dumas et al. (2016).

Theorem 17 *(Generalized Bruhat decomposition (Manthey and Helmke, 2007; Dumas et al., 2016)). For any $m \times n$ matrix A of rank r, there exist an $m \times r$ matrix C in column echelon form, an $r \times n$ matrix E in row echelon form, and an $r \times r$ permutation matrix R such that $\mathsf{A} = \mathsf{CRE}$.*

We will also need an additional structure on the echelon form factors.

Definition 18. Two non-zero columns of matrix are non-overlapping if one has its leading element below the trailing element of the other.

Definition 19. A matrix is s-overlapping if any sub-set of $s+1$ of its non-zero columns contains at least a pair that are non-overlapping.

The motivation for introducing this structure is that left triangular matrices of quasiseparable order s have a generalized Bruhat decomposition with echelon form factors C and E that are s-overlapping.

Theorem 20. *For any $n \times n$ left triangular matrix A of quasiseparable order s and of rank r, there is a generalized Bruhat decomposition of the form $\mathsf{A} = Left(\mathsf{CRE})$ where C and E^T are s-overlapping.*

Proof. Let $(\mathcal{L}, \mathcal{R}, \mathcal{U})$ be a Bruhat generator for A. The matrix $\mathcal{L}$ is s-overlapping: otherwise, there would be a subset S of $s+1$ of columns such that no pair of them is non-overlapping. Let $((i_1, j_1), \ldots, (i_{s+1}, j_{s+1}))$ be the coordinates of their leading elements sorted by increasing row index : $i_1 < i_2 < \cdots < i_{s+1}$. Since $\mathcal{L}$ is left triangular, $j_{s+1} \leq n - i_{s+1}$. The trailing elements of every other column of S must be below row i_{s+1}, hence, $j_k \leq n - i_{s+1}$ for all $k \leq s$ since $\mathcal{L}$ is left triangular. Consequently the $i_{s+1} \times (n - i_{s+1})$ leading submatrix of $\mathcal{L}$ contains $s+1$ pivots, a contradiction. The same reasoning applies to show that E^T is s-overlapping. Consider the permutation matrix $\mathcal{Q}$ such that $\mathcal{L}\mathcal{Q} = \begin{bmatrix} \mathsf{C} & 0_{m \times (n-r)} \end{bmatrix}$ is in column echelon form. Similarly let $\mathcal{P}$ be the permutation matrix such that $\mathcal{P}\mathcal{U} = \begin{bmatrix} \mathsf{E} \\ 0_{(m-r) \times n} \end{bmatrix}$, and remark that $\mathsf{R} = \begin{bmatrix} \mathsf{I}_r & 0 \end{bmatrix} \mathcal{Q}^T \mathcal{R}^T \mathcal{P}^T \begin{bmatrix} \mathsf{I}_r \\ 0 \end{bmatrix}$ is a permutation matrix and verifies $\mathsf{A} = \mathsf{Left(CRE)}$. □

The s-overlapping shape of the echelon form factors in the generalized Bruhat decomposition allow to further compress it as follows.

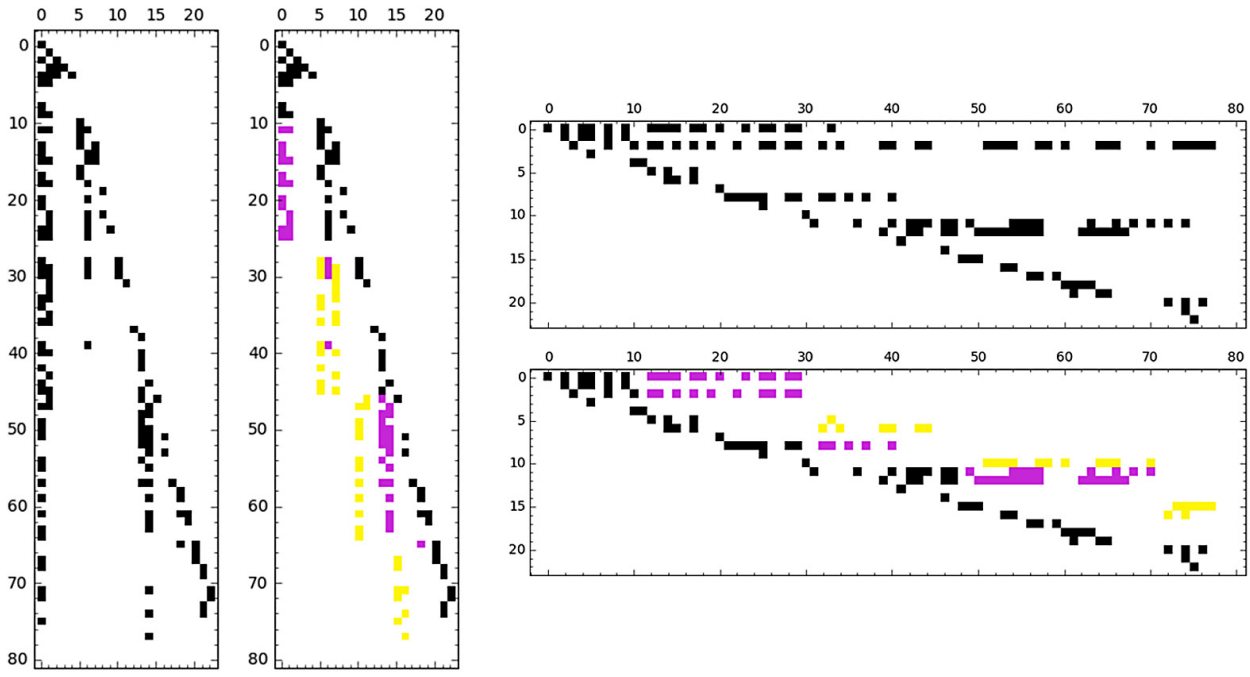


Fig. 2. Support of the matrices $\mathsf{C} = \mathcal{L}\mathcal{Q}$ (left), $\mathsf{E} = \mathcal{P}\mathcal{U}$ (top right) of the s-overlapping CRE decomposition of Theorem 20 applied to the matrix of Fig. 1. The Compression to the block bi-diagonal structure of the corresponding compact Bruhat generator is shown in the central and bottom right matrices. There, D is in black and S in magenta and yellow; those rows and columns moved at step 10 of Algorithm 3 are in yellow. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Proposition 21. *Any s-overlapping $m \times r$ matrix A can be written $\mathsf{A} = (\mathsf{D} + \mathsf{ST})\mathsf{P}$ where P is a permutation matrix, $\mathsf{T} \in \{0,1\}^{r\times r}$ has at most one non-zero element per row and $\mathsf{D} = Diag(\mathsf{D}_1, \ldots, \mathsf{D}_t)$, $\mathsf{S} = \begin{bmatrix} 0 & & & \\ \mathsf{S}_2 & 0 & & \\ & \ddots & \ddots & \\ & & \mathsf{S}_t & \end{bmatrix}$ where each D_i and S_i is $k_i \times s$, except D_t and S_t having possibly fewer columns than s and $\sum_{i=1}^{t} k_i = n$.*

Intuitively, the permutation P^T sorts the columns of A in increasing order of their leading row index. Cutting the columns in slices of dimension s makes AP^T block lower triangular. The block diagonal is D, and the remaining part can be folded into a block sub-diagonal matrix S thanks to the s-overlapping property.

Algorithm 3 is a constructive proof of Proposition 21, computing a compact representation of any s-overlapping matrix.

Proof. Since A is s-overlapping, there exists a permutation P such that $\mathsf{C} = \mathsf{AP}$ is block lower triangular, with blocks of column dimension s except possibly the last one of column dimension $\leq s$. Note that for every i, the dimensions of the blocks S_i and D_i are that of the block C_{ii}: $k_i \times s$. We then prove that there always exists a zero column to pick at step 9. In the first row of $\begin{bmatrix}\mathsf{C}_{i1} & \ldots & \mathsf{C}_{ii}\end{bmatrix}$, there is a non-zero element located in the block C_{ii}. As any non-zero column of $\begin{bmatrix}\mathsf{C}_{i1} & \ldots & \mathsf{C}_{i,i-1}\end{bmatrix}$ has a leading coefficient in A at a row index strictly lower than i, there can not be more than $s-1$ of them. These $s-1$ columns of $\begin{bmatrix}\mathsf{C}_{i1} & \ldots & \mathsf{C}_{i,i-1}\end{bmatrix}$ can all be gathered in the block $\mathsf{C}_{i,i-1}$ of column dimension s.

There only remains to show that ST is the matrix C of step 5. For every pair of indices (j,k) selected in loop 7, right multiplication by $(\mathsf{I}_n + \Delta^{(k,j)} - \Delta^{(k,k)})$ adds up column k to column j and zeroes out column k. On matrix S, this has the effect of reverting each operation done at step 10 in the reverse order of the loop 7. □

Algorithm 3 Compress-to-Block-Bidiagonal.

Require: A: an s-overlapping matrix
Ensure: D, S, T, P: such that $A = (D + ST)P$ as in Proposition 21.
1: $P \leftarrow$ a permutation sorting the columns of A by increasing row position of their leading coefficient.
2: $C \leftarrow AP\begin{bmatrix} I_r \\ 0 \end{bmatrix}$ where r is the number of non-zero columns in A
3: Split C in column slices of width s. ▷ $C = \begin{bmatrix} C_{11} & & & \\ C_{21} & C_{22} & & \\ \vdots & \vdots & \ddots & \\ C_{t1} & C_{t2} & \cdots & C_{tt} \end{bmatrix}$ where C_{ii} is $k_i \times s$ $\forall i < t$.
4: $D \leftarrow \mathrm{Diag}(C_{11}, \ldots, C_{tt})$
5: $C \leftarrow C - D = \begin{bmatrix} 0 & & & \\ C_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ C_{t1} & \cdots & C_{t,t-1} & 0 \end{bmatrix}$
6: $T \leftarrow I_n$
7: **for** $i = 3 \ldots t$ **do**
8: **for** each non-zero column j of $\begin{bmatrix} C_{i,i-2} \\ \cdots \\ C_{t,i-2} \end{bmatrix}$ **do**
9: Let k be a zero column of $\begin{bmatrix} C_{i,i-1} \\ \cdots \\ C_{t,i-1} \end{bmatrix}$
10: Move column j in $\begin{bmatrix} C_{i,i-2} \\ \vdots \\ C_{t,i-2} \end{bmatrix}$ to column k in $\begin{bmatrix} C_{i,i-1} \\ \vdots \\ C_{t,i-1} \end{bmatrix}$.
11: $T \leftarrow (I_n + \Delta^{(k,j)} - \Delta^{(k,k)}) \times T$
12: **end for**
13: **end for**
14: $S \leftarrow C = \begin{bmatrix} 0 & & & \\ C_{21} & 0 & & \\ & \ddots & \ddots & \\ & & C_{t,t-1} & 0 \end{bmatrix}$
15: **return** (D, S, T, P)

Proposition 22. *If an s-overlapping matrix A is in column echelon form, then, the structured representation* (D, S, T, P) *is such that* $P = I_r$ *and* $k_i \geq s$ $\forall i < t$.

Proof. The leading elements of each column are already sorted in a column echelon form, hence $P = I_r$. Then, each block C_{ii} contains s pivots, hence $k_i \geq s$. □

We can now define the compact Bruhat representation.

Definition 23. The compact Bruhat representation of an $n \times n$ s-quasiseparable left triangular matrix A is given by the tuples $(D^{C_A}, S^{C_A}, T^{C_A})$, $(D^{E_A}, S^{E_A}, T^{E_A})$ where $D^{C_A}, S^{C_A}, (D^{E_A})^T$ and $(S^{E_A})^T$ are $n \times r$ block diagonal, with blocks of column dimension s, and T^{C_A} and $(T^{E_A})^T$ are lower triangular $\{0, 1\}$-matrices with r coefficients equals to 1 placed on distinct rows, and a permutation matrix R^A such that

$$\begin{cases} C^A = D^{C_A} + S^{C_A} T^{C_A}, \\ E^A = D^{E_A} + T^{E_A} S^{E_A} \end{cases}$$

and $A = \mathrm{Left}(C^A R^A E^A)$ is a generalized Bruhat decomposition of A.

Fig. 2 illustrates the CRE factors and their folding into a compact Bruhat representation for the matrix used in the example of Fig. 1.

5. Computing with RRR representations

In this section, we will keep considering that the RRR representation is based on any rank revealing factorization (RR), which could originate from various matrix factorizations: PLUQ, CUP, PLE, QR, SVD, etc. We will assume that there exists an algorithm `RRF` computing such a rank revealing factorization. For instance, PLUQ, CUP, PLE decomposition algorithms can be used to compute such a factorization in time $T_{\mathtt{RRF}}(m,n,r) = O(mnr^{\omega-2})$ on an $m \times n$ matrix of rank r (Jeannerod et al., 2013).

5.1. Construction of the generator

The construction of the RRR representation simply consists in computing rank revealing factorizations of all off-diagonal submatrices in a binary splitting of the main diagonal. Let $T_{\mathtt{RRR}}(n,s)$ denote the cost of the computation of the binary tree generator for an $n \times n$ matrix of order of quasiseparability s. It satisfies the recurrence relation $T_{\mathtt{RRR}}(n,s) = 2T_{\mathtt{RRF}}(n,n,s) + 2T_{\mathtt{RRR}}(n/2,s)$ which solves in $T_{\mathtt{RRR}}(n,s) = O(s^{\omega-2}n^2)$.

5.2. Matrix-vector product

In the RRR representation, the application of a vector to the quasiseparable matrix takes the same amount of field operations as the number of coefficients used for its representation. This yields a cost of $O(n(r_L \log \frac{n}{r_L} + r_U \log \frac{n}{r_U})) = O(sn \log \frac{n}{s})$ field operations.

5.3. Auxiliary algorithms

In the following, we present a set of routines that will be used to build multiplication and inversion algorithms for RRR representations. Algorithm 4 expands a matrix from an RRR representation to a dense representation. The recurring relation $T_{\mathtt{RRRExpand}}(n,s) = 2T_{\mathtt{RRRExpand}}(n/2,s) + O(n^2 s^{\omega-2})$

Algorithm 4 `RRRExpand`: expands an RRR representation into a dense representation.

Require: A, an $n \times n$ s-quasiseparable matrix in an RRR representation,
Ensure: $\mathsf{B} \leftarrow \mathsf{A}$ in a dense representation.
1: **if** $n \leq s$ **then return** $\mathsf{B} \leftarrow \mathsf{A}$
2: **end if**
3: $\mathsf{B}_{11} \leftarrow \mathtt{RRRExpand}(\mathsf{A}_{11})$
4: $\mathsf{B}_{22} \leftarrow \mathtt{RRRExpand}(\mathsf{A}_{22})$
5: $\mathsf{B}_{12} \leftarrow \mathsf{L}^{\mathsf{A}}_{12}\mathsf{R}^{\mathsf{A}}_{12}$
6: $\mathsf{B}_{21} \leftarrow \mathsf{L}^{\mathsf{A}}_{21}\mathsf{R}^{\mathsf{A}}_{21}$
7: **return** $\mathsf{B} \leftarrow \begin{bmatrix} \mathsf{B}_{11} & \mathsf{B}_{12} \\ \mathsf{B}_{21} & \mathsf{B}_{22} \end{bmatrix}$

for $n > s$ yields directly

$$T_{\mathtt{RRRExpand}}(n,s) = O(n^2 s^{\omega-2}).$$

Algorithm 5 multiplies two rank revealing factorizations and outputs the result in a rank revealing factorization. As L^{A} and L^{X} have full column rank, so is their product. Hence the $\mathsf{L}^{\mathsf{C}}\mathsf{R}^{\mathsf{C}}$ is a rank revealing factorization of the product. The resulting cost (assuming $s \leq t$ without loss of generality) is

Algorithm 5 `RRxRR`: multiplies two matrices stored as rank revealing factorization.

Require: A, an $m \times k$ matrix of rank $\leq s$ in an RR representation $\mathsf{L}^{\mathsf{A}} \times \mathsf{R}^{\mathsf{A}}$,
Require: B, an $k \times n$ matrix of rank $\leq t$ in an RR representation $\mathsf{L}^{\mathsf{B}} \times \mathsf{R}^{\mathsf{B}}$,
Ensure: $\mathsf{C} \leftarrow \mathsf{A} \times \mathsf{B}$ in an RR representation $\mathsf{L}^{\mathsf{C}} \times \mathsf{R}^{\mathsf{C}}$.
1: $\mathsf{X} \leftarrow \mathsf{R}^{\mathsf{A}}\mathsf{L}^{\mathsf{B}}$
2: $(\mathsf{L}^{\mathsf{X}}, \mathsf{R}^{\mathsf{X}}) \leftarrow \mathtt{RRF}(\mathsf{X})$ ▷ Computes the RR factorization $\mathsf{X} = \mathsf{L}^{\mathsf{X}} \times \mathsf{R}^{\mathsf{X}}$
3: $\mathsf{L}^{\mathsf{C}} \leftarrow \mathsf{L}^{\mathsf{A}}\mathsf{L}^{\mathsf{X}}$
4: $\mathsf{R}^{\mathsf{C}} \leftarrow \mathsf{R}^{\mathsf{X}}\mathsf{R}^{\mathsf{B}}$

$$T_{\texttt{RRxRR}}(m,k,n,s,t) = O(s^{\omega-2}tk) + T_{\texttt{RRF}}(s,t) + O(r_{\mathsf{X}}^{\omega-2}(ms+nt)).$$

With $n = \Theta(m) = \Theta(k)$, this is $T_{\texttt{RRxRR}}(n,s,t) = O((s+t)^{\omega-1}n)$.

Algorithm 6 adds two rank revealing factorizations. It first stacks together the left sides and the right sides of the rank revealing factorizations of the two terms. The resulting factorization may not reveal the rank as the inner dimension may be larger. Therefore, a rank revealing factorization of each factor is first computed, before invoking RRxRR to obtain an RR representation of their product. Assuming $n = \Theta(m)$, the time complexity is

Algorithm 6 RR+RR: adds two matrices stored as rank revealing factorization.

Require: A, an $m \times n$ matrix of rank $\leq s$ in an RR representation $\mathsf{L}^{\mathsf{A}} \times \mathsf{R}^{\mathsf{A}}$
Require: B, an $m \times n$ matrix of rank $\leq t$ in an RR representation $\mathsf{L}^{\mathsf{B}} \times \mathsf{R}^{\mathsf{B}}$
Ensure: $\mathsf{D} \leftarrow \mathsf{A} + \mathsf{B}$ in an RR representation $\mathsf{L}^{\mathsf{D}} \times \mathsf{R}^{\mathsf{D}}$.
1: $\mathsf{X} \leftarrow \begin{bmatrix} \mathsf{L}^{\mathsf{A}} & \mathsf{L}^{\mathsf{B}} \end{bmatrix}$; $\mathsf{Y} \leftarrow \begin{bmatrix} \mathsf{R}^{\mathsf{A}} \\ \mathsf{R}^{\mathsf{B}} \end{bmatrix}$
2: $(\mathsf{L}^{\mathsf{X}}, \mathsf{R}^{\mathsf{X}}) \leftarrow \texttt{RRF}(\mathsf{X})$ ▷ $\mathsf{X} = \mathsf{L}^{\mathsf{X}} \times \mathsf{R}^{\mathsf{X}}$, $r_{\mathsf{X}} = \text{rank}(\mathsf{X})$; L^{X} is $m \times r_{\mathsf{X}}$ and R^{X} is $r_{\mathsf{X}} \times n$
3: $(\mathsf{L}^{\mathsf{Y}}, \mathsf{R}^{\mathsf{Y}}) \leftarrow \texttt{RRF}(\mathsf{Y})$ ▷ $\mathsf{Y} = \mathsf{L}^{\mathsf{Y}} \times \mathsf{R}^{\mathsf{Y}}$, $r_{\mathsf{Y}} = \text{rank}(\mathsf{Y})$; L^{Y} is $m \times r_{\mathsf{Y}}$ and R^{Y} is $r_{\mathsf{Y}} \times n$
4: $\mathsf{D} \leftarrow \texttt{RRxRR}(\mathsf{X}, \mathsf{Y})$ ▷ Computes an RR representation of the product $\mathsf{D} = \mathsf{XY}$

$$T_{\texttt{RR+RR}}(n,s,t) = 2T_{\texttt{RRF}}(n,s+t) + O((s+t)^{\omega-1}n) = O((s+t)^{\omega-1}n).$$

Algorithm 7 adds a quasiseparable matrix in RRR representation with a matrix in RR representation. The time complexity satisfies the recurring relation

Algorithm 7 RRR+RR: adds a quasiseparable matrix in RRR representation and a rank revealing factorization.

Require: A, an $n \times n$ s-quasiseparable matrix in an RRR representation
Require: B, an $n \times n$ matrix of rank $\leq t$ in an RR representation $\mathsf{L}^{\mathsf{B}} \times \mathsf{R}^{\mathsf{B}}$
Ensure: $\mathsf{C} \leftarrow \mathsf{A} + \mathsf{B}$ in an RRR representation.
1: **if** $n \leq s + t$ **then**
2: **return** $\mathsf{C} \leftarrow \texttt{RRRExpand}(\mathsf{A}) + \mathsf{L}^{\mathsf{B}} \times \mathsf{R}^{\mathsf{B}}$
3: **end if**
4: Split the matrices as $\begin{bmatrix} \mathsf{C}_{11} & \mathsf{C}_{12} \\ \mathsf{C}_{21} & \mathsf{C}_{22} \end{bmatrix} \leftarrow \begin{bmatrix} \mathsf{A}_{11} & \mathsf{A}_{12} \\ \mathsf{A}_{21} & \mathsf{A}_{22} \end{bmatrix} + \begin{bmatrix} \mathsf{B}_{11} & \mathsf{B}_{12} \\ \mathsf{B}_{21} & \mathsf{B}_{22} \end{bmatrix}$.
5: $\mathsf{C}_{11} \leftarrow \texttt{RRR} + \texttt{RR}(\mathsf{A}_{11}, \mathsf{B}_{11})$ ▷ $\mathsf{C}_{11} \leftarrow \mathsf{A}_{11} + \mathsf{B}_{11}$
6: $\mathsf{C}_{22} \leftarrow \texttt{RRR} + \texttt{RR}(\mathsf{A}_{22}, \mathsf{B}_{22})$ ▷ $\mathsf{C}_{11} \leftarrow \mathsf{A}_{22} + \mathsf{B}_{22}$
7: $\mathsf{C}_{12} \leftarrow \texttt{RR} + \texttt{RR}(\mathsf{A}_{12}, \mathsf{B}_{12})$ ▷ $\mathsf{C}_{12} \leftarrow \mathsf{A}_{12} + \mathsf{B}_{12}$
8: $\mathsf{C}_{21} \leftarrow \texttt{RR} + \texttt{RR}(\mathsf{A}_{21}, \mathsf{B}_{21})$ ▷ $\mathsf{C}_{21} \leftarrow \mathsf{A}_{21} + \mathsf{B}_{21}$
9: **return** $\mathsf{C} \leftarrow \begin{bmatrix} \mathsf{C}_{11} & \mathsf{C}_{12} \\ \mathsf{C}_{21} & \mathsf{C}_{22} \end{bmatrix}$

$$\begin{cases} T_{\texttt{RRR+RR}}(n,s,t) &= 2T_{\texttt{RRR+RR}}(n/2,s,t) + 2T_{\texttt{RR+RR}}(n/2,s,t) \quad \text{for } n > s+t \\ &= 2T_{\texttt{RRR+RR}}(n/2,s,t) + O(n(s+t)^{\omega-1}) \\ T_{\texttt{RRR+RR}}(n,s,t) &= T_{\texttt{RRRExpand}}(s+t,s) + O((s+t)^2 t^{\omega-2}) \quad \text{for } n \leq s+t \end{cases}$$

which solves in

$$\begin{aligned} T_{\texttt{RRR+RR}}(n,s,t) = &\quad O((s+t)^{\omega-1} n \log \frac{n}{s+t} + \frac{n}{s+t}(s+t)^2(s^{\omega-2} + t^{\omega-2}) \\ = &\quad O((s+t)^{\omega-1} n \log \frac{n}{s+t}). \end{aligned}$$

Algorithm 8 `RRRxTS`: multiplies a quasiseparable matrix in RRR representation with a tall and skinny matrix.

Require: A, an $n \times n$ s-quasiseparable matrix in RRR representation
Require: B, an $n \times t$ matrix
Ensure: $C \leftarrow AB$
1: **if** $n \leq s + t$ **then**
2: **return** $C \leftarrow \texttt{RRRExpand}(A) \times B$
3: **end if**
4: Split the matrices as $\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} \leftarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$.
5: $C_1 \leftarrow \texttt{RRRxTS}(A_{11}, B_1)$ ▷ $C_1 \leftarrow A_{11}B_1$
6: $C_2 \leftarrow \texttt{RRRxTS}(A_{22}, B_2)$ ▷ $C_2 \leftarrow A_{22}B_2$
7: $X \leftarrow R^A_{12} B_2$
8: $C_1 \leftarrow C_1 + L^A_{12} X$ ▷ $C_1 \leftarrow C_1 + A_{12}B_2$
9: $Y \leftarrow R^A_{21} B_1$
10: $C_2 \leftarrow C_2 + L^A_{21} Y$ ▷ $C_2 \leftarrow C_2 + A_{21}B_1$
11: **return** $C \leftarrow \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$

5.4. Quasiseparable times tall and skinny

Algorithm 8 multiplies an s-quasiseparable matrix of dimension $n \times n$ in RRR representation by a tall and skinny matrix: an $n \times t$ rectangular dense matrix with $t \leq n$.

Let $T_{\texttt{RRRxTS}}(n, s, t)$ denote its cost. The recurring relation

$$\begin{cases} T_{\texttt{RRRxTS}}(n,s,t) &= 2T_{\texttt{RRRxTS}}(n/2,s,t) + O(n \max(s,t) \min(s,t)^{\omega-2}) & \text{for } n > s+t \\ T_{\texttt{RRRxTS}}(n,s,t) &= T_{\texttt{RRRExpand}}(s+t,s) + O((s+t)^2 t^{\omega-2}) & \text{for } n \leq s+t \end{cases}$$

yields

$$\begin{aligned} T_{\texttt{RRRxTS}}(n,s,t) = & \quad O((s+t)^{\omega-1} n \log \frac{n}{s+t} + \frac{n}{s+t}(s+t)^2(s^{\omega-2} + t^{\omega-2}) \\ = & \quad O((s+t)^{\omega-1} n \log \frac{n}{s+t}). \end{aligned}$$

From this algorithm, follows Algorithm 9, computing the product of an s-quasiseparable matrix in RRR representation by a rank revealing factorization. Similarly as for Algorithm 5, R^X and R^B have full row rank, so has their product, which ensures that the factors L^D, R^D form a rank revealing factorization of the result. Its time complexity is

Algorithm 9 `RRRxRR`: multiplies a quasiseparable matrix in RRR representation with a rank revealing factorization.

Require: A, an $n \times n$ s-quasiseparable matrix in RRR representation
Require: B, an $n \times m$ matrix of rank $\leq t$ in an RR representation $L^B \times R^B$
Ensure: $D \leftarrow AB$ in a rank revealing factorization $L^D \times R^D$
1: $X \leftarrow \texttt{RRRxTS}(A, L^B)$ ▷ $X \leftarrow AL^B$
2: $(L^X, R^X) \leftarrow \texttt{RRF}(X)$ ▷ Computes the RR factorization $X = L^X \times R^X$
3: $L^D \leftarrow L^X$
4: $R^D \leftarrow R^X R^B$

$$T_{\texttt{RRRxRR}}(n,s,t) = T_{\texttt{RRRxTS}}(n,s,t) + T_{\texttt{RRF}}(n,t) + O(nt^{\omega-1}) = O((s+t)^{\omega-1} n \log \frac{n}{s+t})$$

5.5. Quasiseparable times quasiseparable

The product of an s-quasiseparable matrix by a t-quasiseparable matrix is an $(s+t)$-quasiseparable matrix (Eidelman and Gohberg, 1999). Algorithm 10, calling Algorithms 5, 6, 7 and 8, shows how to perform such a multiplication with the RRR representations. In steps 9 and 10, a $(s+t)$-quasiseparable

Algorithm 10 RRRxRRR.

Require: A, an $n \times n$ s-quasiseparable matrix in an RRR representation,
Require: B, an $n \times n$ t-quasiseparable matrix in an RRR representation,
Ensure: $C \leftarrow A \times B$ in an RRR representation.
1: **if** $n \leq s + t$ **then**
2: **return** $C \leftarrow$ RRRExpand(A) $\times$ RRRExpand(B)
3: **end if**
4: Split the matrices as $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \leftarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$.
5: $C_{11} \leftarrow$ RRRxRRR (A_{11}, B_{11}) ▷ $C_{11} \leftarrow A_{11}B_{11}$
6: $C_{22} \leftarrow$ RRRxRRR (A_{22}, B_{22}) ▷ $C_{22} \leftarrow A_{22}B_{22}$
7: $X \leftarrow$ RRxRR (A_{12}, B_{21}) ▷ $X \leftarrow A_{12}B_{21}$
8: $Y \leftarrow$ RRxRR (A_{21}, B_{12}) ▷ $Y \leftarrow A_{21}B_{12}$
9: $C_{11} \leftarrow$ RRR+RR (C_{11}, X) ▷ $C_{11} \leftarrow C_{11} + X$
10: $C_{22} \leftarrow$ RRR+RR (C_{22}, Y) ▷ $C_{22} \leftarrow C_{22} + Y$
11: $L^X \leftarrow$ RRRxTS(A_{11}, L^B_{12}); $R^X \leftarrow R^B_{12}$ ▷ $X \leftarrow A_{11}B_{12}$ in RR representation
12: $L^Y \leftarrow L^A_{12}$; $R^Y \leftarrow$ TSxRRR(R^A_{12}, B_{22}) ▷ $Y \leftarrow A_{12}B_{22}$ in RR representation revealing factorization
13: $C_{12} \leftarrow$ RR + RR(X, Y) ▷ $C_{12} \leftarrow X + Y$
14: $L^X \leftarrow$ RRRxTS(A_{11}, L^B_{21}); $R^X \leftarrow R^B_{21}$ ▷ $X \leftarrow A_{11}B_{21}$ in RR representation
15: $L^Y \leftarrow L^A_{21}$; $R^Y \leftarrow$ TSxRRR(R^A_{21}, B_{22}) ▷ $Y \leftarrow A_{21}B_{22}$ in RR representation
16: $C_{21} \leftarrow$ RR + RR(X, Y) ▷ $C_{21} \leftarrow X + Y$
17: **return** $C \leftarrow \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

matrix is added to a rank revealing factorization of rank $(s + t)$. It should in general result in an RRR representation of an $2(s + t)$-quasiseparable matrix. However, the matrix C is no more than $(s + t)$-quasiseparable, hence the rank revealing factorization of the result, will have rank only $s + t$. The reductions to RR representation, performed in step 2 of Algorithm 5 and steps 2 and 3 of Algorithm 6, ensure that this factorization will be reduced to this size.

Let $T_{\text{RRR}\times\text{RRR}}(n, s, t)$ denote the time complexity of this algorithm. If $n \leq s + t$, then $T_{\text{RRRxRRR}}(n, s, t) = T_{\text{RRRExpand}}(n, s) + T_{\text{RRRExpand}}(n, t) + O(n^\omega) = O((s + t)^\omega)$. Now consider the case $n > s + t$.

$$\begin{aligned} T_{\text{RRRxRRR}}(n, s, t) = \quad & 2T_{\text{RRRxRRR}}(n/2, s, t) + 2T_{\text{RRxRR}}(n/2, s, t) + 2T_{\text{RRRxTS}}(n/2, s, t) \\ & + 2T_{\text{RRR+RR}}(n/2, s + t, s + t) + 2T_{\text{RR+RR}}(n/2, s + t, s + t) \\ = \quad & 2T_{\text{RRRxRRR}}(n/2, s, t) + O((s + t)^{\omega-1} n \log \frac{n}{s + t}) \end{aligned}$$

Consequently, $T_{\text{RRRxRRR}}(n, s, t) = O((s + t)^{\omega-1} n \log^2 \frac{n}{s+t})$.

5.6. Computing the inverse in RRR representation

We consider the case, as in Eidelman and Gohberg (1999, § 6), where the matrix to be inverted has generic rank profile, i.e. all of its leading principal minors are non-vanishing. Under this assumption, Strassen's divide and conquer algorithm (Strassen, 1969) reduces the computation of the inverse to matrix multiplication. More precisely, the inverse is recursively computed using the following block 2×2 formula:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}D^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}D^{-1} \\ -D^{-1}A_{21}A_{11}^{-1} & D^{-1} \end{bmatrix},$$

where $D = A_{22} - A_{21}A_{11}^{-1}A_{12}$.

This formula leads to a recursive algorithm that we adapt to the case of quasiseparable matrices in RRR representation in Algorithm 11. The fact that the inverse matrix X is itself s-quasiseparable, implies that the matrix D is also s-quasiseparable and not $2s$-quasiseparable, as the generic upper bound would say. The compression happens in the RR+RR routine, at step 10. Hence all operations except the recursive calls take $O(s^{\omega-1} n \log \frac{n}{s})$. The overall complexity of Algorithm 11 is therefore $T_{\text{RRRInverse}}(n, s) = O(s^{\omega-1} n \log^2 \frac{n}{s})$.

Algorithm 11 `RRRinvert`: compute the inverse in RRR representation.

Require: A, an $n \times n$ s-quasiseparable strongly regular matrix in RRR representation,
Ensure: $X = A^{-1}$, s-quasiseparable in RRR representation.

1: **if** $n \leq s$ **then**
2: $Y \leftarrow \texttt{RRRExpand}(A)$
3: **return** $X \leftarrow \texttt{Invert}(Y)$
4: **end if**
5: Split the matrix as $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and $X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}$
6: $Y_{11} \leftarrow \texttt{RRRinvert}(A_{11})$ ▷ $Y_{11} = A_{11}^{-1}$
7: $Y_{12} \leftarrow \texttt{RRRxRR}(Y_{11}, A_{12})$ ▷ $Y_{12} \leftarrow A_{11}^{-1}A_{12}$
8: $Y_{21} \leftarrow \texttt{RRRxRR}(A_{21}, Y_{11})$ ▷ $Y_{21} \leftarrow A_{21}A_{11}^{-1}$
9: $Z \leftarrow -\texttt{RRxRR}(A_{21}Y_{12})$ ▷ $Z \leftarrow -A_{21}A_{11}^{-1}A_{12}$
10: $D \leftarrow \texttt{RR} + \texttt{RR}(A_{22}, Z)$ ▷ $D \leftarrow A_{22} - A_{21}A_{11}^{-1}A_{12}$
11: $X_{22} \leftarrow \texttt{RRRinvert}(D)$ ▷ $X_{22} = D^{-1}$
12: $X_{21} \leftarrow -\texttt{RRRxRR}(X_{22}, Y_{21})$ ▷ $X_{21} \leftarrow -D^{-1}A_{21}A_{11}^{-1}$
13: $W \leftarrow -\texttt{RRxRR}(Y_{12}, X_{21})$ ▷ $W \leftarrow A_{11}^{-1}A_{12}D^{-1}A_{21}A_{11}^{-1}$
14: $X_{12} \leftarrow -\texttt{RRRxRR}(Y_{12}, X_{22})$ ▷ $X_{12} \leftarrow -A_{11}^{-1}A_{12}D^{-1}$
15: $X_{11} \leftarrow \texttt{RRR} + \texttt{RR}(Y_{11}, W)$ ▷ $X_{11} \leftarrow A_{11}^{-1} + A_{11}^{-1}A_{12}D^{-1}A_{21}A_{11}^{-1}$
16: **return** $X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}$.

6. Computing with a compact Bruhat representation

6.1. Construction of the generator

We first propose in Algorithm 12 an evolution of Algorithm 2 to compute the factors of the Bruhat generator (without compression) for a left triangular matrix.

Theorem 24. *For any $n \times n$ matrix A with a left triangular part of quasiseparable order s, Algorithm 12 computes the Bruhat generator of the left triangular part of A in $O(s^{\omega-2}n^2)$ field operations.*

Proof. The correctness of $\mathcal{R}$ is proven in Theorem 10. We will prove by induction the correctness of $\mathcal{U}$, noting that the correctness of $\mathcal{L}$ works similarly.

Let $H = P_2L_2U_2Q_2$ and $I = P_3L_3U_3Q_3$ be PLUQ decompositions of H and I revealing their rank profile matrices. Assume that Algorithm LT-Bruhat is correct in the two recursive calls 15 and 16, that is

$$\mathcal{U}_2 = \text{Left}(P_2 \begin{bmatrix} U_2 \\ 0 \end{bmatrix} Q_2), \quad \mathcal{U}_3 = \text{Left}(P_3 \begin{bmatrix} U_3 \\ 0 \end{bmatrix} Q_3),$$
$$\mathcal{L}_2 = \text{Left}(P_2 \begin{bmatrix} L_2 & 0 \end{bmatrix} Q_2), \quad \mathcal{L}_3 = \text{Left}(P_3 \begin{bmatrix} L_3 & 0 \end{bmatrix} Q_3).$$

At step 7, we have

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & * \end{bmatrix} = \begin{bmatrix} P_1 & \\ & I_{\frac{n}{2}} \end{bmatrix} \left[\begin{array}{cc|c} L_1 & & \\ M_1 & I_{\frac{n}{2}-r_1} & \\ \hline E & 0 & I_{\frac{n}{2}} \end{array}\right] \left[\begin{array}{cc|c} U_1 & V_1 & D \\ & 0 & F \\ \hline & G & \end{array}\right] \begin{bmatrix} Q_1 & \\ & I_{\frac{n}{2}} \end{bmatrix}$$

As the first r_1 rows of P_1^TH are zeros, there exists $\bar{P}_2$ a permutation matrix and $\bar{L}_2$, a lower triangular matrix, such that $P_1^TP_2L_2 = \begin{bmatrix} 0_{r_1 \times \frac{n}{2}} \\ \bar{P}_2\bar{L}_2 \end{bmatrix}$. Similarly, there exist $\bar{Q}_3$, a permutation matrix and $\bar{U}_3$, an upper triangular matrix, such that $U_3Q_3Q_1^T = \begin{bmatrix} 0_{\frac{n}{2} \times r_1} & \bar{U}_3\bar{Q}_3 \end{bmatrix}$. Hence

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & * \end{bmatrix} = \begin{bmatrix} P_1 & \\ & P_3 \end{bmatrix} \left[\begin{array}{cc|c} L_1 & & \\ M_1 & \bar{P}_2\bar{L}_2 & \\ \hline P_3^TE & 0 & L_3 \end{array}\right] \left[\begin{array}{cc|c} U_1 & V_1 & DQ_2^T \\ & 0 & U_2 \\ \hline & \bar{U}_3\bar{Q}_3 & \end{array}\right] \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}$$

Algorithm 12 LT-Bruhat.

Require: A: an $n \times n$ matrix
Ensure: $(\mathcal{L}, \mathcal{R}, \mathcal{U})$: a Bruhat generator for the left triangular part of A

1: **if** $n = 1$ **then return** ([0], [0], [0])
2: Split $\mathsf{A} = \begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_2 \\ \mathsf{A}_3 & \end{bmatrix}$ where A_3 is $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$
3: Decompose $\mathsf{A}_1 = \mathsf{P}_1 \begin{bmatrix} \mathsf{L}_1 \\ \mathsf{M}_1 \end{bmatrix} \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \end{bmatrix} \mathsf{Q}_1$
4: $\mathsf{R}_1 \leftarrow \mathsf{P}_1 \begin{bmatrix} \mathsf{I}_{r_1} & \\ & 0 \end{bmatrix} \mathsf{Q}_1$ where $r_1 = \mathrm{rank}(\mathsf{A}_1)$.
5: $\begin{bmatrix} \mathsf{B}_1 \\ \mathsf{B}_2 \end{bmatrix} \leftarrow {\mathsf{P}_1}^T \mathsf{A}_2$
6: $\begin{bmatrix} \mathsf{C}_1 & \mathsf{C}_2 \end{bmatrix} \leftarrow \mathsf{A}_3 {\mathsf{Q}_1}^T$
7: Here $A = \left[\begin{array}{cc|c} \mathsf{L}_1 \backslash \mathsf{U}_1 & \mathsf{V}_1 & \mathsf{B}_1 \\ \mathsf{M}_1 & 0 & \mathsf{B}_2 \\ \hline \mathsf{C}_1 & \mathsf{C}_2 & \end{array}\right]$.
8: $\mathsf{D} \leftarrow {\mathsf{L}_1}^{-1} \mathsf{B}_1$
9: $\mathsf{E} \leftarrow \mathsf{C}_1 {\mathsf{U}_1}^{-1}$
10: $\mathsf{F} \leftarrow \mathsf{B}_2 - \mathsf{M}_1 \mathsf{D}$
11: $\mathsf{G} \leftarrow \mathsf{C}_2 - \mathsf{E}\mathsf{V}_1$
12: Here $\mathsf{A} = \left[\begin{array}{cc|c} \mathsf{L}_1 \backslash \mathsf{U}_1 & \mathsf{V}_1 & \mathsf{D} \\ \mathsf{M}_1 & 0 & \mathsf{F} \\ \hline \mathsf{E} & \mathsf{G} & \end{array}\right]$.
13: $\mathsf{H} \leftarrow \mathsf{P}_1 \begin{bmatrix} 0_{r_1 \times \frac{n}{2}} \\ \mathsf{F} \end{bmatrix}$
14: $\mathsf{I} \leftarrow \begin{bmatrix} 0_{r_1 \times \frac{n}{2}} & \mathsf{G} \end{bmatrix} \mathsf{Q}_1$
15: $(\mathcal{L}_2, \mathcal{R}_2, \mathcal{U}_2) \leftarrow \mathtt{LT-Bruhat}(\mathsf{H})$
16: $(\mathcal{L}_3, \mathcal{R}_3, \mathcal{U}_3) \leftarrow \mathtt{LT-Bruhat}(\mathsf{I})$
17: $\mathcal{L} \leftarrow \begin{bmatrix} \mathsf{P}_1 \begin{bmatrix} \mathsf{L}_1 & 0 \\ \mathsf{M}_1 & 0 \end{bmatrix} \mathsf{Q}_1 & 0 \\ \mathrm{Left}(\begin{bmatrix} \mathsf{E} & 0 \end{bmatrix} \mathsf{Q}_1) & 0 \end{bmatrix} + \begin{bmatrix} 0 & \mathcal{L}_2 \\ \mathcal{L}_3 & \end{bmatrix}$
18: $\mathcal{U} \leftarrow \begin{bmatrix} \mathsf{P}_1 \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \\ 0 & 0 \end{bmatrix} \mathsf{Q}_1 & \mathrm{Left}(\mathsf{P}_1 \begin{bmatrix} \mathsf{D} \\ 0 \end{bmatrix}) \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & \mathcal{U}_2 \\ \mathcal{U}_3 & \end{bmatrix}$
19: $\mathcal{R} \leftarrow \begin{bmatrix} \mathcal{R}_1 & \mathcal{R}_2 \\ \mathcal{R}_3 & \end{bmatrix}$
20: **return** $(\mathcal{L}, \mathcal{R}, \mathcal{U})$

Setting $\mathsf{N}_1 = \bar{\mathsf{P}}_2^T \mathsf{M}_1$ and $\mathsf{W}_1 = \mathsf{V}_1 \bar{\mathsf{Q}}_3^T$, we have

$$\begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_2 \\ \mathsf{A}_3 & * \end{bmatrix} = \begin{bmatrix} \mathsf{P}_1 \begin{bmatrix} \mathsf{I}_{r_1} & \\ & \bar{\mathsf{P}}_2 \end{bmatrix} & \\ & \mathsf{P}_3 \end{bmatrix} \left[\begin{array}{cc} \mathsf{L}_1 & \\ \mathsf{N}_1 & \bar{\mathsf{L}}_2 \\ \hline \mathsf{E} \quad 0 & \mathsf{L}_3 \end{array}\right] \left[\begin{array}{cc|c} \mathsf{U}_1 & \mathsf{W}_1 & \mathsf{D}\mathsf{Q}_2^T \\ & 0 & \mathsf{U}_2 \\ & \bar{\mathsf{U}}_3 & \end{array}\right] \begin{bmatrix} \begin{bmatrix} \mathsf{I}_{r_1} & \\ & \bar{\mathsf{Q}}_3 \end{bmatrix} \mathsf{Q}_1 & \\ & \mathsf{Q}_2 \end{bmatrix}.$$

A PLUQ of $\begin{bmatrix} \mathsf{A}_1 & \mathsf{A}_2 \\ \mathsf{A}_3 & \end{bmatrix}$ revealing its rank profile matrix is then obtained from this decomposition by a row block cyclic-shift on the second factor and a column block cyclic shift on the third factor as in Dumas et al. (2013, Algorithm 1).

Finally,

$$\mathsf{P} \begin{bmatrix} \mathsf{U} \\ 0 \end{bmatrix} \mathsf{Q} = \begin{bmatrix} \mathsf{P}_1 & \\ & \mathsf{I}_{\frac{n}{2}} \end{bmatrix} \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 & \mathsf{D} \\ & 0 & \bar{\mathsf{P}}_2 \mathsf{U}_2 \mathsf{Q}_2 \\ & \mathsf{P}_3 \bar{\mathsf{U}}_3 \bar{\mathsf{Q}}_3 & \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathsf{Q}_1 & \\ & \mathsf{I}_{\frac{n}{2}} \end{bmatrix}$$

$$= \begin{bmatrix} \mathsf{P}_1 \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \\ 0 & 0 \end{bmatrix} \mathsf{Q}_1 & \mathsf{P}_1 \begin{bmatrix} \mathsf{D} \\ 0 \end{bmatrix} \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} & \mathsf{P}_2 \begin{bmatrix} \mathsf{U}_2 \\ 0 \end{bmatrix} \mathsf{Q}_2 \\ \mathsf{P}_3 \begin{bmatrix} \mathsf{U}_3 \\ 0 \end{bmatrix} \mathsf{Q}_3 & \end{bmatrix}.$$

Hence $\text{Left}(\mathsf{PUQ}) = \begin{bmatrix} \mathsf{P}_1 \begin{bmatrix} \mathsf{U}_1 & \mathsf{V}_1 \\ 0 & 0 \end{bmatrix} \mathsf{Q}_1 & \text{Left}(\mathsf{P}_1 \begin{bmatrix} \mathsf{D} \\ 0 \end{bmatrix}) \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} & \mathcal{U}_2 \\ \mathcal{U}_3 & \end{bmatrix}$.

The complexity analysis is exactly that of Theorem 10. □

The computation of a compact Bruhat generator, as shown in Algorithm 13, is then directly obtained by combining Algorithm 12 with Algorithm 3.

Algorithm 13 Compact Bruhat generator.

Require: A: an $n \times n$ left triangular matrix of quasiseparable order s
Ensure: $(\mathsf{D}^{\mathsf{C_L}}, \mathsf{S}^{\mathsf{C_L}}, \mathsf{T}^{\mathsf{C_L}}), \mathsf{R_L}, (\mathsf{D}^{\mathsf{E_L}}, \mathsf{S}^{\mathsf{E_L}}, \mathsf{T}^{\mathsf{E_L}})$: a Compact Bruhat generator for $\mathsf{L} = \text{Left}(\mathsf{J}_n\mathsf{A})$
Ensure: $(\mathsf{D}^{\mathsf{C_U}}, \mathsf{S}^{\mathsf{C_U}}, \mathsf{T}^{\mathsf{C_U}}), \mathsf{R_U}, (\mathsf{D}^{\mathsf{E_U}}, \mathsf{S}^{\mathsf{E_U}}, \mathsf{T}^{\mathsf{E_U}})$: a Compact Bruhat generator for $\mathsf{U} = \text{Left}(\mathsf{A}\mathsf{J}_n)$
1: $\mathsf{L} \leftarrow \text{Left}(\mathsf{J}_n\mathsf{A})$
2: $\mathsf{U} \leftarrow \text{Left}(\mathsf{A}\mathsf{J}_n)$
3: $(\mathcal{L}_\mathsf{L}, \mathcal{R}_\mathsf{L}, \mathcal{U}_\mathsf{L}) \leftarrow \texttt{LT-Bruhat}(\mathsf{L})$
4: $(\mathcal{L}_\mathsf{U}, \mathcal{R}_\mathsf{U}, \mathcal{U}_\mathsf{U}) \leftarrow \texttt{LT-Bruhat}(\mathsf{U})$
5: $(\mathsf{D}^{\mathsf{C_L}}, \mathsf{S}^{\mathsf{C_L}}, \mathsf{T}^{\mathsf{C_L}}, \mathsf{P}^{\mathsf{C_L}}) \leftarrow \texttt{Compress-to-Block-Bidiagonal}(\mathcal{L}_\mathsf{L})$
6: $(\mathsf{D}^{\mathsf{E_L}}, \mathsf{S}^{\mathsf{E_L}}, \mathsf{T}^{\mathsf{E_L}}, \mathsf{P}^{\mathsf{E_L}}) \leftarrow (\texttt{Compress-to-Block-Bidiagonal}(\mathcal{U}_\mathsf{L}^T))^T$
7: $\mathsf{R_L} \leftarrow \begin{bmatrix} \mathsf{I}_r & 0 \end{bmatrix} \mathsf{P}^{\mathsf{C_L}} \mathcal{R}_\mathsf{L}^T \mathsf{P}^{\mathsf{E_L}} \begin{bmatrix} \mathsf{I}_r \\ 0 \end{bmatrix}$
8: $\mathsf{R_U} \leftarrow \begin{bmatrix} \mathsf{I}_r & 0 \end{bmatrix} \mathsf{P}^{\mathsf{C_U}} \mathcal{R}_\mathsf{U}^T \mathsf{P}^{\mathsf{E_U}} \begin{bmatrix} \mathsf{I}_r \\ 0 \end{bmatrix}$

6.2. Multiplication by a tall and skinny matrix

We consider the multiplication of an s-quasiseparable matrix in Compact Bruhat representation by an $n \times t$ dense rectangular matrix ($t \leq s$), and show that is can be performed in $O(st^{\omega-2}n) = O(s^{\omega-1}n)$ field operations.

The Compact Bruhat representation stores a representation of two left triangular matrices, corresponding to the upper and lower triangular parts of the matrix. Hence it suffices to show how to multiply an s-quasiseparable left triangular matrix in Compact Bruhat representation with a tall and skinny matrix.

Using the Definition 23, this means computing

$$\mathsf{C} = \text{Left}(\mathsf{C}^\mathsf{A}\mathsf{R}^\mathsf{A}\mathsf{E}^\mathsf{A})\mathsf{B}$$

where B is dense $n \times t$. Without the `Left` operator, the target complexity $O(s^{\omega-1}n)$ would be reached by first computing the product $\mathsf{E}^\mathsf{A}\mathsf{B}$ and then applying R^A and C^A on the left. However because of the `Left` operator, each row of the result matrix C involves a distinct partial sum of the product $\mathsf{E}^\mathsf{A}\mathsf{B}$:

$$\mathsf{C}_{i,*} = \mathsf{C}^\mathsf{A}_{i,*}\mathsf{R}^\mathsf{A} \left(\sum_{j=1}^{n-i} \mathsf{E}^\mathsf{A}_{*,j}\mathsf{B}_{j,*} \right).$$

We will therefore avoid computing the accumulation in this product, keeping point-wise products available in memory. In order to reach the target complexity, the products of dimension s will be computed with accumulation, keeping the terms of the unevaluated sum available at the level of size s blocks.

Cutting these matrices on a grid of size s, let $N = \lceil n/s \rceil$ and $\mathsf{C} = \begin{bmatrix} \mathsf{C}_1 & \dots & \mathsf{C}_N \end{bmatrix}^T$, $\mathsf{E}^\mathsf{A} = \begin{bmatrix} \mathsf{E}^\mathsf{A}_1 & \dots & \mathsf{E}^\mathsf{A}_N \end{bmatrix}$, $\mathsf{C}^\mathsf{A} = \begin{bmatrix} \mathsf{C}^\mathsf{A}_1 & \dots & \mathsf{C}^\mathsf{A}_N \end{bmatrix}^T$ and $\mathsf{B} = \begin{bmatrix} \mathsf{B}_1 & \dots & \mathsf{B}_N \end{bmatrix}^T$. We have

$$\mathsf{C}_i = \mathsf{C}^\mathsf{A}_i\mathsf{R}^\mathsf{A} \sum_{j=1}^{N-i} \mathsf{E}^\mathsf{A}_j\mathsf{B}_j + \text{Left}\left(\mathsf{C}^\mathsf{A}_i\mathsf{R}^\mathsf{A}\mathsf{E}^\mathsf{A}_{N-i+1}\right)\mathsf{B}_{N-i+1}.$$

Algorithm 14 `LeftCBxTS`.

Require: A, an $n \times n$ s-quasiseparable left triangular matrix: $\mathsf{A} = \text{Left}(\mathsf{C}^\mathsf{A}\mathsf{R}^\mathsf{A}\mathsf{E}^\mathsf{A})$
Require: B, an $n \times t$ matrix
Ensure: $\mathsf{C} \leftarrow \mathsf{AB}$, an $n \times t$ dense tall and skinny matrix
1: **for** $j = 1 \dots N-1$ **do**
2: $\quad \mathsf{X}_j \leftarrow \mathsf{E}^\mathsf{A}_j \mathsf{B}_j$ ▷ in a compact representation $\mathsf{X}_j = \mathsf{D}^\mathsf{X}_j + \mathsf{T}^\mathsf{E}\mathsf{S}^\mathsf{X}_j$
3: $\quad \mathsf{Y}_j \leftarrow \mathsf{R}^\mathsf{A}\mathsf{X}_j$
4: **end for**
5: compute all partial sums of these blocks: $\mathsf{Z}_i = \sum_{j=1}^{N-i} \mathsf{Y}_j$;
6: apply C^A_i to the left: $\mathsf{V}_i = \mathsf{C}^\mathsf{A}_i \mathsf{Z}_i$;
7: add the trailing term, $\mathsf{C}_i = \mathsf{V}_i + \text{Left}\left(\mathsf{C}^\mathsf{A}_i \mathsf{R}^\mathsf{A} \mathsf{E}^\mathsf{A}_{N-i+1}\right)\mathsf{B}_{N-i+1}$.
8: **return** $\mathsf{C} \leftarrow \begin{bmatrix} \mathsf{C}_1 \\ \vdots \\ \mathsf{C}_N \end{bmatrix}$

Each of these blocks C_i are then computed as shown in Algorithm 14.

In the compact Bruhat representation, the row echelon form E^A is stored in the form $\mathsf{E}^\mathsf{A} = \mathsf{D}^\mathsf{E} + \mathsf{T}^\mathsf{E}\mathsf{S}^\mathsf{E}$ where D and S are block diagonal with blocks of dimension $s \times k_j$ where $k_j \geq s$.

Step 2 reduces to computing $\mathsf{D}^\mathsf{X}_j = \mathsf{D}^\mathsf{E}_j \mathsf{B}_j$ and $\mathsf{S}^\mathsf{X}_j = \mathsf{S}^\mathsf{E}_j \mathsf{B}_j$ such that

$$\mathsf{X}_j = \mathsf{D}^\mathsf{X}_j + \mathsf{T}^\mathsf{E}\mathsf{S}^\mathsf{X}_j. \tag{7}$$

Each of these products requires $O(k_j s t^{\omega-2})$ field operations, hence Step 2 costs $O(nst^{\omega-2})$ field operations. After Step 2, the matrix X is stored in a compact representation, given by equation (7), requiring only $O(nt)$ space.

Step 3 does not involve any field operation as the multiplication on the left by T^E and the final sum act on matrices of non-overlapping support. The overall amount of data being copied is linear in the number of non-zero elements: $O(nt)$.

Step 5 can be achieved by computing the prefix sum of the Y_i's: $\mathsf{Z}_1 = \mathsf{Y}_1$ and $\mathsf{Z}_i = \mathsf{Z}_{i-1} + \mathsf{Y}_i$. Each step involves $O(st)$ additions (the number of non-zero elements in Y_i), hence Step 5 costs $O(nt)$ field operations.

Step 6 is a sequence of N products of an $s \times r$ matrix $\mathsf{C}^\mathsf{A}_i = \mathsf{D}^\mathsf{C}_i + \mathsf{S}^\mathsf{C}_i \mathsf{T}^\mathsf{C}$ by an $r \times r$ matrix Z_i. As both D^C_i and S^C_i have only s continuous non-zero columns, each of these product costs $O(s^2 t^{\omega-2})$ and the overall cost is $O(nst^{\omega-2})$.

Step 7 is achieved by computing the $s \times s$ factor $\text{Left}(\mathsf{C}^\mathsf{A}_i \mathsf{R}^\mathsf{A} \mathsf{E}^\mathsf{A}_{N-i+1})$ explicitly in $O(s^\omega)$, and then applying it to B_{N-i+1} in $O(s^2 t^{\omega-2})$.

Overall the cost of Algorithm 14 is $O(nts^{\omega-2})$ field operations.

Corollary 25. *An s-quasiseparable matrix in Compact Bruhat representation can be multiplied*

a. *by a vector in time $O(ns)$.*
b. *by a dense $n \times m$ matrix in time $O(s^{\omega-2}nm)$.*
c. *by a another s-quasiseparable matrix in time $O(s^{\omega-2}n^2)$.*

Proof.

a. Specializing this `LeftCBxTS` algorithm with $t = 1$ yields an algorithm for multiplying by vector in time $O(ns)$.
b. Splitting the dense matrix in $\lceil \frac{m}{s} \rceil$ slices and applying `LeftCBxTS` on each of them takes $O(s^{\omega-1} n \lceil \frac{m}{s} \rceil) = O(s^{\omega-2}nm)$.
c. Expanding one of the two matrices into a dense representation and multiplying it to the other one takes $O(s^{\omega-2}n^2)$. □

The last item in the corollary improves over the complexity of multiplying two dense matrices in $O(n^\omega)$. However, the result being itself a $2s$-quasiseparable matrix, it could be presented in a Compact Bruhat representation. Hence the target cost for this operation is far below: $O(s^{\omega-1}n)$ since both input and output have size $O(sn)$. Applying similar techniques as in Algorithm 14, we could only produce the output as two terms of the form Left(LR) where L and R^T are $n \times (\text{rank}(\mathsf{A}) + \text{rank}(\mathsf{B}))$ in time $O(s^{\omega-1}n)$, but we were unable to perform the compression to a Compact Bruhat representation within this target complexity for the moment.

References

Bini, D., Pan, V., 1994. Polynomial and Matrix Computations, vol.1: Fundamental Algorithms. Birkhäuser, Boston.

Boito, P., Eidelman, Y., Gemignani, L., 2016. Implicit QR for companion-like pencils. Math. Comput. 85 (300), 1753–1774. http://www.ams.org/mcom/2016-85-300/S0025-5718-2015-03020-8/.

Bostan, A., Jeannerod, C.-P., Schost, E., 2008. Solving structured linear systems with large displacement rank. Theor. Comput. Sci. 407 (1–3), 155–181. http://www.sciencedirect.com/science/article/pii/S0304397508003940.

Bruhat, F., 1956. Sur les représentations induites des groupes de Lie. Bull. Soc. Math. Fr. 84, 97–205. http://eudml.org/doc/86911.

Carrier, J., Greengard, L., Rokhlin, V., 1988. A fast adaptive multipole algorithm for particle simulations. SIAM J. Sci. Stat. Comput. 9 (4), 669–686. http://epubs.siam.org/doi/abs/10.1137/0909044.

Chan, T.F., 1987. Rank revealing QR factorizations. Linear Algebra Appl. 88, 67–82. http://www.sciencedirect.com/science/article/pii/0024379587901030.

Chandrasekaran, S., Dewilde, P., Gu, M., Pals, T., Sun, X., van der Veen, A., White, D., 2005. Some fast algorithms for sequentially semiseparable representations. SIAM J. Matrix Anal. Appl. 27 (2), 341–364. http://epubs.siam.org/doi/abs/10.1137/S0895479802405884.

Chandrasekaran, S., Gu, M., Pals, T., 2006. A fast ULV decomposition solver for hierarchically semiseparable representations. SIAM J. Matrix Anal. Appl. 28 (3), 603–622. http://epubs.siam.org/doi/abs/10.1137/S0895479803436652.

Chandrasekaran, S., Ipsen, I., 1994. On rank-revealing factorisations. SIAM J. Matrix Anal. Appl. 15 (2), 592–622. http://epubs.siam.org/doi/abs/10.1137/S0895479891223781.

Delvaux, S., Van Barel, M., 2007. A Givens-weight representation for rank structured matrices. SIAM J. Matrix Anal. Appl. 29 (4), 1147–1170. http://epubs.siam.org/doi/abs/10.1137/060654967.

Dumas, J.-G., Pernet, C., Sultan, Z., 2013. Simultaneous computation of the row and column rank profiles. In: Kauers, M. (Ed.), Proc. ISSAC'13. ACM Press, pp. 181–188.

Dumas, J.-G., Pernet, C., Sultan, Z., 2015. Computing the rank profile matrix. In: Proc ISSAC'15. ACM, New York, NY, USA, pp. 149–156. Distinguished paper award. http://doi.acm.org/10.1145/2755996.2756682.

Dumas, J.-G., Pernet, C., Sultan, Z., 2016. Fast computation of the rank profile matrix and the generalized Bruhat decomposition. J. Symb. Comput.

Eidelman, Y., Gohberg, I., 1999. On a new class of structured matrices. Integral Equ. Oper. Theory 34 (3), 293–324. http://link.springer.com/article/10.1007/BF01300581.

Eidelman, Y., Gohberg, I., 2005. On generators of quasiseparable finite block matrices. Calcolo 42 (3–4), 187–214. http://link.springer.com/article/10.1007/s10092-005-0102-4.

Eidelman, Y., Gohberg, I., Olshevsky, V., 2005. The QR iteration method for hermitian quasiseparable matrices of an arbitrary order. Linear Algebra Appl. 404, 305–324. http://www.sciencedirect.com/science/article/pii/S0024379505001369.

Gohberg, I., Kailath, T., Koltracht, I., 1985. Linear complexity algorithms for semiseparable matrices. Integral Equ. Oper. Theory 8 (6), 780–804. http://link.springer.com/article/10.1007/BF01213791.

Hwang, T.-M., Lin, W.-W., Yang, E.K., 1992. Rank revealing LU factorizations. Linear Algebra Appl. 175, 115–141. http://www.sciencedirect.com/science/article/pii/002437959290305T.

Jeannerod, C.-P., Pernet, C., Storjohann, A., 2013. Rank-profile revealing Gaussian elimination and the CUP matrix decomposition. J. Symb. Comput. 56, 46–68.

Kailath, T., Kung, S.-Y., Morf, M., 1979. Displacement ranks of matrices and linear equations. J. Math. Anal. Appl. 68 (2), 395–407. http://www.sciencedirect.com/science/article/pii/0022247X79901240.

Le Gall, F., 2014. Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. ISSAC '14. ACM, New York, NY, USA, pp. 296–303. http://doi.acm.org/10.1145/2608628.2608664.

Malaschonok, G.I., 2010. Fast generalized Bruhat decomposition. In: CASC'10. In: Lect. Notes Comput. Sci., vol. 6244. Springer-Verlag, Berlin, Heidelberg, pp. 194–202.

Manthey, W., Helmke, U., 2007. Bruhat canonical form for linear systems. Linear Algebra Appl. 425 (2–3), 261–282. Special issue in honor of Paul Fuhrmann.

Pan, C.-T., 2000. On the existence and computation of rank-revealing LU factorizations. Linear Algebra Appl. 316 (1–3), 199–222. http://www.sciencedirect.com/science/article/pii/S0024379500001208.

Pan, V., 1990. On computations with dense structured matrices. Math. Comput. 55 (191), 179–190.

Pernet, C., 2016. Computing with quasiseparable matrices. In: Proc. ISSAC'16. ACM, pp. 389–396. hal-01264131.

Sheng, Z., Dewilde, P., Chandrasekaran, S., 2007. Algorithms to solve hierarchically semi-separable systems. In: Alpay, D., Vinnikov, V. (Eds.), System Theory, the Schur Algorithm and Multidimensional Analysis. In: Oper. Theory, Adv. Appl., vol. 176. Birkhäuser, Basel, pp. 255–294. http://link.springer.com/chapter/10.1007/978-3-7643-8137-0_5.

Strassen, V., 1969. Gaussian elimination is not optimal. Numer. Math. 13, 354–356.

The LinBox Group, 2016. LinBox: Linear algebra over black-box matrices. v1.4.1 edition. http://linalg.org/.

Tyrtyshnikov, E., 1997. Matrix Bruhat decompositions with a remark on the QR (GR) algorithm. Linear Algebra Appl. 250, 61–68.

Vandebril, R., Barel, M.V., Golub, G., Mastronardi, N., 2005. A bibliography on semiseparable matrices. Calcolo 42 (3), 249–270. http://dx.doi.org/10.1007/s10092-005-0107-z.

Vandebril, R., Van Barel, M., Mastronardi, N., 2007. Matrix Computations and Semiseparable Matrices: Linear Systems, vol. 1. The Johns Hopkins University Press.

Xia, J., Chandrasekaran, S., Gu, M., Li, X.S., 2010. Fast algorithms for hierarchically semiseparable matrices. Numer. Linear Algebra Appl. 17 (6), 953–976. http://onlinelibrary.wiley.com/doi/10.1002/nla.691/abstract.

Certificates for Triangular Equivalence and Rank Profiles*

Jean-Guillaume Dumas David Lucas Clément Pernet
Université Grenoble Alpes, Laboratoire Jean Kuntzmann, CNRS, UMR 5224
700 avenue centrale, IMAG - CS 40700, 38058 Grenoble cedex 9, France
{firstname.lastname}@univ-grenoble-alpes.fr

ABSTRACT

In this paper, we give novel certificates for triangular equivalence and rank profiles. These certificates enable to verify the row or column rank profiles or the whole rank profile matrix faster than recomputing them, with a negligible overall overhead. We first provide quadratic time and space non-interactive certificates saving the logarithmic factors of previously known ones. Then we propose interactive certificates for the same problems whose Monte Carlo verification complexity requires a small constant number of matrix-vector multiplications, a linear space, and a linear number of extra field operations. As an application we also give an interactive protocol, certifying the determinant of dense matrices, faster than the best previously known one.

1. INTRODUCTION

Within the setting of verifiable computing, we propose in this paper *interactive certificates* with the taxonomy of [4]. Indeed, we consider a protocol where a *Prover* performs a computation and provides additional data structures or exchanges with a *Verifier* who will use these to check the validity of the result, faster than by just recomputing it. More precisely, in an interactive certificate, the Prover submits a *Commitment*, that is some result of a computation; the Verifier answers by a *Challenge*, usually some uniformly sampled random values; the Prover then answers with a *Response*, that the Verifier can use to convince himself of the validity of the commitment. Several *rounds* of challenge/response might be necessary for the Verifier to be fully convinced.

By Prover (resp. Verifier) *time*, we thus mean bounds on the number of arithmetic operations performed by the Prover (resp. Verifier) during the protocol, while by extra *space*, we mean bounds on the volume of data being exchanged, not counting the size of the input and output of the computation.

Such protocols are said to be *complete* if the probability that a true statement is rejected by the Verifier can be made arbitrarily small; and *sound* if the probability that a false statement is accepted by the Verifier can be made arbitrarily small. In practice it is sufficient that those probabilities are < 1, as the protocols can always be run several times. Some certificates will also be *perfectly complete*, that is a true statement is never rejected by the Verifier. All these certificates can be simulated non-interactively by Fiat-Shamir heuristic [10]: uniformly sampled random values produced by the Verifier are replaced by cryptographic hashes of the input and of previous messages in the protocol. Complexities are preserved.

We do not use generic approaches to verified computation (where protocols check circuits with polylogarithmic depth [12] or use amortized models and homomorphic encryption [2]). Rather, we use dedicated certificates as those designed for dense [11, 14] or sparse [4, 5] exact linear algebra. The obtained certificates are problem-specific, but try to reduce as much as possible the overhead for the Prover, while preserving a fast verification procedure.

We will consider an $m \times n$ matrix A of rank r over a field $\mathbb{F}$. The *row rank profile* of A is the lexicographically minimal sequence of r indices of independent rows of A. Matrix A has *generic row rank profile* if its row rank profile is $(1, \ldots, r)$. The *column rank profile* is defined similarly on the columns of A. Matrix A has generic rank profile if its r first leading principal minors are nonzero. The *rank profile matrix* of A, denoted by $\mathcal{R}_A$ is the unique $m \times n$ $\{0,1\}$-matrix with r nonzero entries, of which every leading sub-matrix has the same rank as the corresponding sub-matrix of A. It is possible to compute $\mathcal{R}_A$ with a deterministic algorithm in $O(mnr^{\omega-2})$ or with a Monte-Carlo probabilistic algorithm in $(r^\omega + m + n + \mu(A))^{1+o(1)}$ field operations [8], where $\mu(A)$ is the arithmetic cost to multiply A by a vector.

We first propose quadratic, space and verification time, non-interactive practical certificates for the row or column rank profile and for the rank profile matrix that are rank-sensitive. Previously known certificates have additional logarithmic factors to the quadratic complexities: replacing matrix multiplications by quadratic verifications in recursive algorithms yields at least one $\log(n)$ factor [14], graph-based approaches cumulate this and other logarithmic factors, at least from a compression by magical graphs and from a dichotomic search [16].

We then propose two linear space interactive certificates: one certifying that two non-singular matrices are triangular equivalent, i.e. there is a triangular change of basis from one to the other; the other one, certifying that a matrix has a generic rank profile. These certificates are then applied to certify the row or column rank profile, the Q (permutation) and D (diagonal) factors of a LDUP factorization, the determinant and the rank profile matrix. These certificates require, for the Verifier, between 1 and 3 applications of A to a vector and a linear amount of field operations. They are still elimination-based for the Prover, but do not require to communicate the obtained triangular decomposition. For the Determinant, this new certificates require the computation of a PLUQ decompo-

*This work is partly funded by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

ISSAC '17, July 25-28, 2017, Kaiserslautern, Germany

ACM ISBN 978-1-4503-5064-8/17/07…$15.00
http://dx.doi.org/10.1145/3087604.3087609

sition for the Prover, linear communication and Verifier time, with no restriction on the field size.

Table 1 compares linear quadratic volumes of communication, as well as sub-cubic (PLUQ, CHARPOLY) or quadratic matrix operations (one matrix-vector multiplication with a dense matrix is denoted `fgemv`). The results shows first that it is interesting to use linear space certificates even when they have quadratic Verification time. The table also presents a practical constant factor of about 5 between PLUQ and CHARPOLY computations. Computations use the FFLAS-FFPACK library (http://linbox-team.github.io/fflas-ffpack) on a single Intel Skylake core @3.4GHz, while we measured some communications between two workstations over an Ethernet Cat. 6, @1Gb/s network cable.

Dimension	$2k$	$10k$	$50k$
PLUQ	0.28s	17.99s	1448.16s
CHARPOLY	1.96s	100.37s	8047.56s
Linear comm.	0.50s	0.50s	0.50s
Quadratic comm.	1.50s	7.50s	222.68s
`fgemv`	0.0013s	0.038s	1.03s

Table 1: Communication of 64 bit words versus computation modulo 131071

A summary of our contributions is given in Table 3, to be compared with the state of the art in Table 2. We identify the symmetric group with the group of permutation matrices, and write $P \in \mathcal{S}_n$ to denote that a matrix P is a permutation matrix. There, $P[i]$ is the row index of the nonzero element of its i-th column; $\mathcal{D}_n(\mathbb{F})$ is the group of invertible diagonal matrices over the field $\mathbb{F}$ and $[A]_J^I$ is the (I, J)-minor of the matrix A (the determinant of the submatrix of A with row indices in I and column indices in J). Lastly, $x \xleftarrow{\$} S$ denotes that x is sampled uniformly at random from S.

2. NON INTERACTIVE AND QUADRATIC COMMUNICATION CERTIFICATES

In this section, we propose two certificates, first for the column (resp. row) rank profile, and, second, for the rank profile matrix. While the certificates have a quadratic space communication complexity, they have the advantage of being non-interactive.

2.1 Freivalds' certificate for matrix product

In this paper, we will use Freivalds' certificate [11] to verify matrix multiplication. Considering three matrices A, B and C in $\mathbb{F}^{n\times n}$, such that $A \times B = C$, a straightforward way of verifying the equality would be to perform the multiplication $A \times B$ and to compare its result coefficient by coefficient with C. While this method is deterministic, it has a time complexity of $O(n^\omega)$, which is the matrix multiplication complexity. As such, it cannot be a certificate, as there is no complexity difference between the computation and the verification.

Prover		Verifier
	$A, B \in \mathbb{F}^{n\times n}$	
$C = AB$	$\xrightarrow{C}$	$v \in \mathbb{F}^{n\times 1}$
		$A(Bv) - Cv \stackrel{?}{=} 0$

Protocol 1: Freivalds' certificate for matrix product

Freivalds' certificate proposes a probabilistic method to check this product in a time complexity of $\mu(A) + \mu(B) + \mu(C)$ using matrix/vector multiplication, as detailed in Figure 1.

2.2 Column rank profile certificate

We now propose a certificate for the column rank profile.

Prover		Verifier
	$A \in \mathbb{F}^{m\times n}$	
A $PLUQ$ decomposition of A s.t. UQ is in row echelon form	$\xrightarrow{P,L,U,Q}$	UQ row echelonized?
		$A \stackrel{?}{=} PLUQ$, by cert. 1
		Return $Q[1], \ldots, Q[r]$

Protocol 2: Column rank profile, non-interactive

LEMMA 1. *Let $A = PLUQ$ be the PLUQ decomposition of an $m \times n$ matrix A of rank r. If UQ is in row echelon form then $(Q[1], \ldots, Q[r])$ is the column rank profile of A.*

PROOF. Write $A = P \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} U_1 & U_2 \end{bmatrix} Q$, where L_1 and U_1 are $r \times r$ lower and upper triangular respectively. If UQ is in echelon form, then $R = \begin{bmatrix} I_r & U_1^{-1}U_2 \\ 0_{(m-r)\times n} \end{bmatrix}$ is in reduced echelon form. Now

$$\begin{bmatrix} U_1^{-1} & \\ & I_{m-r} \end{bmatrix} \begin{bmatrix} L_1 & \\ L_2 & I_{m-r} \end{bmatrix}^{-1} P^T A = \begin{bmatrix} U_1^{-1}UQ \\ 0_{(m-r)\times n} \end{bmatrix} = R$$

is left equivalent to A and is therefore the echelon form of A. Hence the sequence of column positions of the pivots in R, that is $(Q[1], \ldots, Q[r])$, is the column rank profile of A. □

Lemma 1 provides a criterion to verify a column rank profile from a PLUQ decomposition. Such decompositions can be computed in practice by several variants of Gaussian elimination, with no arithmetic overhead, as shown in [13] or [7, § 8]. Hence, we propose the certificate in Protocol 2.

THEOREM 1. *Let $A \in \mathbb{F}^{m\times n}$ with $r = \mathrm{rank}(A)$. Certificate 2, verifying the column rank profile of A is sound, perfectly complete, with a communication bounded by $O(r(m+n))$, a Prover computation bounded by $O(mnr^{\omega-2})$ and a Verifier computation cost bounded by $O(r(m+n)) + \mu(A)$.*

PROOF. If the Prover is honest, then, UQ will be in row echelon form and $A = PLUQ$, thus, by Lemma 1, the Verifier will be able to read the column rank profile of A from Q. If the Prover is dishonest, either $A \neq PLUQ$, which will be caught by the Prover with probabilty $p \geq 1 - \frac{1}{q}$ using Freivalds' certificate [11] or UQ is not in row echelon from, which will be caught every time by the Verifier.

The Prover sends P, L, U and Q to the Verifier, hence the communication cost of $O(r(m+n))$, as P and Q are permutation matrices and L, U, are respectively $m \times r$ and $r \times n$ matrices, with $r = rank(A)$. Using algorithms provided in [13], one can compute the expected $PLUQ$ decomposition in $O(mnr^{\omega-2})$. The Verifier has to check if $A = PLUQ$, and if UQ is in row echelon form, which can be done in $O(r(m+n))$. □

Note that this holds for the row rank profile of A: in that case, the Verifier has to check if PL is in column echelon form.

	Algorithm	Inter.	Prover		Communication	Probabilistic	$\#\mathbb{F}$
			Determ.	Time		Verifier Time	
RANK	[14] over [1]	No	No	$\widetilde{O}(r^\omega + \mu(A))$	$\widetilde{O}(r^2 + m + n)$	$\widetilde{O}(r^2 + \mu(A))$	≥ 2
	[4]	Yes	No	$O(n(\mu(A) + n))$	$O(m + n)$	$2\mu(A) + \widetilde{O}(m + n)$	$\widetilde{O}(min\{m, n\})$
	[9]	Yes	Yes	$O(mnr^{\omega-2})$	$O(m + r)$	$O(r + \mu(A) + m + n)$	≥ 2
CRP/RRP	[14] over [16]	No	No	$\widetilde{O}(r^\omega + m + n + \mu(A))$	$\widetilde{O}(r^2 + m + n)$	$\widetilde{O}(r^2 + m + n + \mu(A))$	$\widetilde{O}(min\{m, n\})$
	[14] over [13]	No	Yes	$O(mnr^{\omega-2})$	$\widetilde{O}(mn)$	$\widetilde{O}(mn)$	≥ 2
RPM	[14] over [8]	No	No	$\widetilde{O}(r^\omega + m + n + \mu(A))$	$\widetilde{O}(r^2 + m + n)$	$\widetilde{O}(r^2 + m + n + \mu(A))$	$\widetilde{O}(min\{m, n\})$
	[14] over [6]	No	Yes	$O(mnr^{\omega-2})$	$\widetilde{O}(mn)$	$\widetilde{O}(mn)$	≥ 2
DET	[11] & PLUQ	No	Yes	$O(n^\omega)$	$O(n^2)$	$O(n^2) + \mu(A)$	≥ 2
	[5] & CHARPOLY	Yes	No	$O(n\mu(A))$ or $O(n^\omega)$	$O(n)$	$\mu(A) + O(n)$	$\geq n^2$

Table 2: State of the art certificates for the rank, the row and column rank profiles, the rank profile matrix and the determinant

	Algorithm	Interactive	Prover		Communication	Probabilistic	$\#\mathbb{F}$
			Deterministic	Time		Verifier Time	
CRP/RRP	§ 2.2	No	Yes	$O(mnr^{\omega-2})$	$O(r(m + n))$	$O(r(m + n)) + \mu(A)$	≥ 2
	§ 4.2	Yes	Yes	$O(mnr^{\omega-2})$	$O(m + n)$	$2\mu(A) + O(m + n)$	≥ 2
RPM	§ 2.3	No	Yes	$O(mnr^{\omega-2})$	$O(r(m + n))$	$O(r(m + n)) + \mu(A)$	≥ 2
	§ 4.3	Yes	Yes	$O(mnr^{\omega-2})$	$O(m + n)$	$4\mu(A) + O(m + n)$	≥ 4
DET	§ 4.1 & PLUQ	Yes	Yes	$O(n^\omega)$	$O(n)$	$\mu(A) + O(n)$	≥ 2

Table 3: This paper's contributions

2.3 Rank profile matrix certificate

LEMMA 2. *A decomposition $A = PLUQ$ reveals the rank profile matrix, namely $\mathcal{R}_A = P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q$, if and only if $P \begin{bmatrix} L & 0 \end{bmatrix} P^T$ is lower triangular and $Q^T \begin{bmatrix} U \\ 0 \end{bmatrix} Q$ is upper triangular.*

PROOF. The *only if* case is proven in [8, Th. 21]. Now suppose that $P \begin{bmatrix} L & 0_{m\times(m-r)} \end{bmatrix} P^T$ is lower triangular. Then we must also have that $\overline{L} = P \begin{bmatrix} L & \\ & I_{m-r} \end{bmatrix} P^T$ is lower triangular and non-singular. Similarly suppose that $Q^T \begin{bmatrix} U \\ 0 \end{bmatrix} Q$ is upper triangular so that $\overline{U} = Q^T \begin{bmatrix} U & \\ 0 & I_{n-r} \end{bmatrix} Q$ is non-singular upper triangular. We have $A = \overline{L} P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q \overline{U}$. Hence the rank of any (i, j) leading submatrix of A is that of the (i, j) leading submatrix of $P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q$, thus proving that $\mathcal{R}_A = P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q$. □

We use this characterization to verify the computation of the rank profile matrix in the following protocol: Once the Verifier receives P, L, U and Q, he has to:

1. Check that $A = PLUQ$, using Freivalds' certificate [11]
2. Check that L is echelonized by P and U^T by Q^T.
3. If successful, compute the rank profile matrix of A as $\mathcal{R}_A = P \begin{bmatrix} I_r & \\ & 0_{(m-r)\times(n-r)} \end{bmatrix} Q$

Prover		Verifier
	$A \in \mathbb{F}^{m\times n}$	
	P,L,U,Q $\longrightarrow$	
a PLUQ decomp. of A revealing $\mathcal{R}_A$.		1. $A \stackrel{?}{=} PLUQ$ by Protoc. 2.1 2. Is PLP^T lower triangular? 3. Is Q^TUQ upper triangular?

Protocol 3: Rank profile matrix, non-interactive

THEOREM 2. *Certificate 3 verifies the rank profile matrix of A, it is sound and perfectly complete, with a communication cost bounded by $O(r(n + m))$, a Prover computation cost bounded by $O(mnr^{\omega-2})$ and a Verifier computation cost bounded by $O(r(m+n)) + \mu(A)$.*

PROOF. If the Prover is honest, then, the provided $PLUQ$ decomposition is indeed a factorization of A, which means Freivalds' certificate will pass. It also means this $PLUQ$ decomposition reveals the rank profile matrix. According to Lemma 2, PLP^T will be lower triangular and Q^TUQ upper triangular. Hence the verification will succeeds and $\mathcal{R}_A = P \begin{bmatrix} I_r & \\ & 0_{(m-r)\times(n-r)} \end{bmatrix} Q$ is indeed the rank profile matrix of A. If the Prover is dishonest, either $A \neq PLUQ$, which will be caught with probabilty $p \geq 1 - \frac{1}{q}$ by Freivalds' certificate or the $PLUQ$ decomposition does not reveal the rank profile matrix of A. In that case, Lemma 2 implies that either $P \begin{bmatrix} L & 0 \end{bmatrix} P^T$ is not lower triangular or $P \begin{bmatrix} U \\ 0 \end{bmatrix} Q$ is not upper triangular which the will be detected.

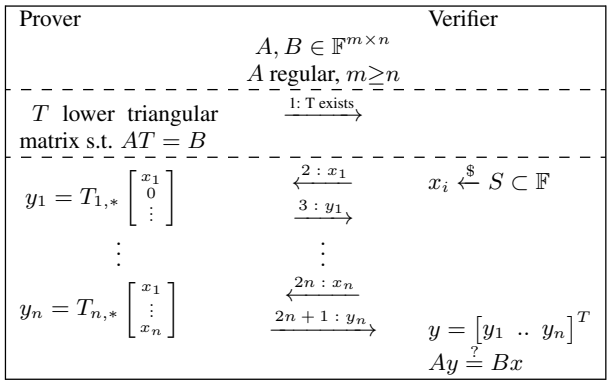
The Prover sends P, L, U and Q to the Verifier, hence the communication cost of $O((n + m)r)$. A rank profile matrix revealing $PLUQ$ decomposition can be computed in $O(mnr^{\omega-2})$ operations [6]. The Verifier has to check if $A = PLUQ$, which can be achieved in $O((m + n)r) + \mu(A)$ field operations. □

3. LINEAR COMMUNICATION CERTIFICATE TOOLBOX

3.1 Triangular one sided equivalence

Two matrices $A, B \in \mathbb{F}^{m\times n}$ are right (resp. left) equivalent if there exist an invertible $n \times n$ matrix T such that $AT = B$ (resp. $TA = B$). If in addition T is a lower triangular matrix, we say that A and B are lower triangular right (resp. left) equivalent. The upper triangular right (resp. left) equivalence is defined similarly. We propose a certification protocol that two matrices are left or right triangular equivalent. Here, A and B are input, known by the Verifier and the Prover. A simple certificate would be the matrix T itself, in which case the Verifier would check the product $AT = B$ using Freivalds' certificate. This certificate is non-interactive and

requires a quadratic amount of communication. In what follows, we present a certificate which allows to verify the one sided triangular equivalence without communicating T, requiring only $2n$ communications. It is essentially a Freivalds' certificate with a more constrained interaction pattern in the way the challenge vector and the response vector are communicated. This pattern imposes a triangular structure in the way the Provers' responses depend on the Verifier challenges which match with the structure of the problem.



Protocol 4: Lower triang. right equivalence of regular matrices

THEOREM 3. *Let $A, B \in \mathbb{F}^{m\times n}$, and assume A is regular. Certificate 4 proves that there exists a lower triangular matrix T such that $AT = B$. This certificate is sound, with probabilty larger than $1 - \frac{1}{|S|}$, perfectly complete, occupies $2n$ communication space, and can be computed in $O(mn^{\omega-1})$ field operations and verified in $\mu(A) + \mu(B)$ field operations.*

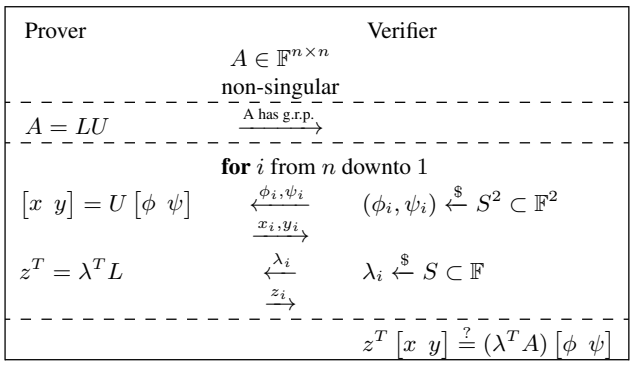
PROOF. If the Prover is honest, then $AT = B$ and she just computes $y = Tx$, so that $Ay = ATx = Bx$. If the Prover is dishonest, replace the random values $x_1, \ldots, x_n$ by algebraically independent variables $X_1, \ldots, X_n$. Since A is regular, there is a unique $n \times n$ matrix T (that is, $T = A^\dagger B$ with $A^\dagger$ the Moore-Penrose inverse of A) such that $AT = B$. For the same reason, there is a unique vector $\widehat{Y} = A^\dagger BX$ such that $A\widehat{Y} = BX$. The vector $\widehat{Y}$ is then formed by n degree-1 polynomials in $X_1, \ldots, X_n$. If T is not lower triangular, let i be the first row such that $T_{i,j} \neq 0$ for some $j > i$, and let j_m be the largest such j. Then $\widehat{Y}_i$ has degree 1 in X_{j_m}. Let Y be the vector output by the Prover. At step $2i + 1$, the value for X_{j_m} was still not released, hence Y_i is constant in X_{j_m}. As A is regular, the verification $AY = BX = A\widehat{Y}$ is equivalent to $Y - \widehat{Y} = 0$. The i-th component in this equation is $Y_i - \widehat{Y}_i = 0$, whose left hand-side contains a non zero monomial in X_j. There is therefore a probability lower than $1/|S|$ that the random choice for x_j makes this polynomial vanish.

This certificate requires to transmit x and y, which costs $2n$ in communication. The Verifier has to compute Ay and Bx, whose computational cost is $\mu(A) + \mu(B)$. The Prover has to compute T, this can be done by a PLUQ elimination on A followed by a triangular system solve, both in $O(mn^{\omega-1})$. Then $y = Tx$ requires only $O(n^2)$ operations. □

Note that the case where T is upper triangular works similarly: the Verifier needs to transmit x in reverse order, starting by x_n.

3.2 Generic rank profile-ness

The problem here is to verify whether a non-singular input matrix $A \in \mathbb{F}^{m\times n}$ has generic rank profile (to test non-singularity, one can apply beforehand the linear communication certificate in [4, Fig. 2], see also Protocol 8 thereafter). A matrix A has generic rank profile if and only if it has an LU decomposition $A = LU$, with L unit lower triangular and U non-singular upper triangular. The protocol picks random vectors ϕ, ψ, λ and asks the Prover to provide the vectors $z^T = \lambda^T L$, $x = U\phi$, $y = U\psi$ on the fly, while receiving the coefficients of the vectors ϕ, ψ, λ one at a time. These vectors satisfy the fundamental equations $z^T x = \lambda^T A\phi$ and $z^T y = \lambda^T A\psi$ that will be checked by the Verifier.



Protocol 5: Generic rank profile with linear communication

THEOREM 4. *Certificate 5 verifying that a non-singular matrix has generic rank profile is sound, with probability larger than $1 - \frac{1}{|S|}$, perfectly complete, communicates $3n$ field elements, and can be computed in $O(n^\omega)$ field operations for the Prover and $\mu(A) + 8n$ field operations for the Verifier.*

We will need the following Lemma, used in Dodgson determinant condensation rule.

LEMMA 3 (DESNANOT-JACOBI, OR DODGSON RULE [3]).

$$[A]^{\{1..n\}}_{\{1..n\}} [A]^{\{2..n-1\}}_{\{2..n-1\}} = \begin{vmatrix} [A]^{\{1..n-1\}}_{\{1..n-1\}} & [A]^{\{2..n\}}_{\{1..n-1\}} \\ [A]^{\{1..n-1\}}_{\{2..n\}} & [A]^{\{2..n\}}_{\{2..n\}} \end{vmatrix}.$$

Applying the same permutation, the cyclic shift of order 1 to the left, on the rows and columns of A, yields the following formula with no change of sign:

$$[A]^{\{1..n\}}_{\{1..n\}} [A]^{\{1..n-2\}}_{\{1..n-2\}} = \begin{vmatrix} [A]^{\{1..n-2,n\}}_{\{1..n-2,n\}} & [A]^{\{1..n-1\}}_{\{1..n-2,n\}} \\ [A]^{\{1..n-2,n\}}_{\{1..n-1\}} & [A]^{\{1..n-1\}}_{\{1..n-1\}} \end{vmatrix}. \quad (1)$$

PROOF OF THEOREM 4. The protocol is perfectly complete: if $A = LU$, then $z^T \begin{bmatrix} x & y \end{bmatrix} = \lambda^T LU \begin{bmatrix} \phi & \psi \end{bmatrix} = \lambda^T A \begin{bmatrix} \phi & \psi \end{bmatrix}$.

Now, for the soundness, replace every ϕ, ψ, λ chosen at random by the Verifier by vectors of algebraically independent variables Φ, Ψ, Λ. Similarly, the responses of the Prover z, x, y are now vectors of algebraically independent variables Z, X, Y. Under the assumption of the success of the Verifier test,

$$\begin{cases} Z^T X &= \Lambda^T A \Phi \\ Z^T Y &= \Lambda^T A \Psi \end{cases}, \quad (2)$$

and that A is non-singular, we will prove the following induction hypothesis (where $d_i = [A]^{\{1\ldots i\}}_{\{1\ldots i\}}$, $d_0 = 1$):

$$\mathrm{H}_i : \begin{cases} Z^T_{i\ldots n} X_{i\ldots n} &= \frac{1}{d_{i-1}} \sum_{i\leq j,k\leq n} \Lambda_k \, [A]^{\{1\ldots i-1,k\}}_{\{1\ldots i-1,j\}} \, \Phi_j \\ Z^T_{i\ldots n} Y_{i\ldots n} &= \frac{1}{d_{i-1}} \sum_{i\leq j,k\leq n} \Lambda_k \, [A]^{\{1\ldots i-1,k\}}_{\{1\ldots i-1,j\}} \, \Psi_j \\ d_j \neq 0 \; \forall j < i \end{cases}$$

For $i = 1$, note that $[A]_{\{1\ldots i-1,j\}}^{\{1\ldots i-1,k\}} = A_{k,j}$, hence the right hand-sides of the first two equations of H_1 can be written as:

$$\sum_{1\le j,k\le n} \Lambda_k A_{k,j} \Phi_j = \Lambda^T A \Phi = Z^T X$$
$$\sum_{1\le j,k\le n} \Lambda_k A_{k,j} \Psi_j = \Lambda^T A \Psi = Z^T Y$$

by (2). Finally $d_0 = 1$ is obviously nonzero.

Now suppose H_i is true for some $0 \le i < n$. Then

$$\begin{cases} Z_i X_i + Z_{i+1..n}^T X_{i+1..n} = \frac{1}{d_{i-1}} \Lambda_i \sum_{j=i}^n [A]_{\{1..i-1,j\}}^{\{1..i\}} \Phi_j \\ \qquad + \frac{1}{d_{i-1}} \sum_{j=i}^n \sum_{k=i+1}^n \Lambda_k [A]_{\{1..i-1,j\}}^{\{1..i-1,k\}} \Phi_j \\ Z_i Y_i + Z_{i+1..n}^T Y_{i+1..n} = \frac{1}{d_{i-1}} \Lambda_i \sum_{j=i}^n [A]_{\{1..i-1,j\}}^{\{1..i\}} \Psi_j \\ \qquad + \frac{1}{d_{i-1}} \sum_{j=i}^n \sum_{k=i+1}^n \Lambda_k [A]_{\{1..i-1,j\}}^{\{1..i-1,k\}} \Psi_j \end{cases} . \tag{3}$$

At the time of choosing the value for Λ_i, all variables are set, except Z_i. Hence for all value assigned to Λ_i, there is a value for Z_i that satisfies the above system of two linear equations in Z_i and Λ_i. Consequently this system is singular and the following two determinants vanish:

$$\begin{vmatrix} d_{i-1} X_i & \sum_{j=i}^n [A]_{\{1..i-1,j\}}^{\{1..i\}} \Phi_j \\ d_{i-1} Y_i & \sum_{j=i}^n [A]_{\{1..i-1,j\}}^{\{1..i\}} \Psi_j \end{vmatrix} = 0 \tag{4}$$

$$\begin{vmatrix} \sum_{j=i}^n [A]_{\{1..i-1,j\}}^{\{1..i\}} \Phi_j & d_{i-1} Z_{i+1..n}^T X_{i+1..n} - F_A(\Lambda, i, \Phi) \\ \sum_{j=i}^n [A]_{\{1..i-1,j\}}^{\{1..i\}} \Psi_j & d_{i-1} Z_{i+1..n}^T Y_{i+1..n} - F_A(\Lambda, i, \Psi) \end{vmatrix} = 0 \tag{5}$$

where $F_A(\Lambda, i, R) = \sum_{j=i}^n \sum_{k=i+1}^n \Lambda_k [A]_{\{1..i-1,j\}}^{\{1..i-1,k\}} R_j$, for $R = \Phi, \Psi$. Actually, Equation (5) has the form $\begin{vmatrix} d_i \Phi_i + b & a\Phi_i + e \\ d_i \Psi_i + c & a\Psi_i + f \end{vmatrix} = 0$ where $d_i = [A]_{\{1..i\}}^{\{1..i\}}$, $a = -\sum_{k=i+1}^n \Lambda_k [A]_{\{1..i\}}^{\{1..i-1,k\}}$ and b, c, e, f are constants with respect to the variables Φ_i, Ψ_i.

If $d_i = 0$, then, at least one $[A]_{\{1..i-1,j\}}^{\{1..i\}}$ for $j > i$ must be nonzero, otherwise A would be singular. Similarly, at least one $[A]_{\{1..i\}}^{\{1..i-1,k\}}$ for $k > i$ is nonzero, hence a is a nonzero polynomial in $\Lambda_{i+1}, \ldots, \Lambda_n$ and b, c are nonzero polynomials in Φ_j, Ψ_j for $j > i$, but constant in Φ_i and Ψ_i. This is a contradiction, as the first column of the determinant, $\begin{bmatrix} b \\ c \end{bmatrix}$ can not be colinear with the second one. Hence $d_i \neq 0$.

Therefore $\begin{bmatrix} e \\ f \end{bmatrix} = \frac{a}{d_i} \begin{bmatrix} b \\ c \end{bmatrix}$ which is

$$\begin{cases} d_{i-1} Z_{i+1..n}^T X_{i+1..n} = \frac{1}{d_i} \sum_{j,k=i+1}^n \Lambda_k \left(d_i [A]_{\{1..i-1,j\}}^{\{1..i-1,k\}} \right. \\ \qquad \left. - [A]_{\{1..i\}}^{\{1..i-1,k\}} [A]_{\{1..i-1,j\}}^{\{1..i\}} \right) \Phi_j \\ d_{i-1} Z_{i+1..n}^T Y_{i+1..n} = \frac{1}{d_i} \sum_{j,k=i+1}^n \Lambda_k \left(d_i [A]_{\{1..i-1,j\}}^{\{1..i-1,k\}} \right. \\ \qquad \left. - [A]_{\{1..i\}}^{\{1..i-1,k\}} [A]_{\{1..i-1,j\}}^{\{1..i\}} \right) \Psi_j \end{cases}$$

Applying variant (1) of Lemma 3 to $[A]_{\{1..i,j\}}^{\{1..i,k\}}$, yields

$$\begin{cases} d_{i-1} Z_{i+1..n}^T X_{i+1..n} = \frac{1}{d_i} \sum_{j,k=i+1}^n \Lambda_k d_{i-1} [A]_{\{1..i,j\}}^{\{1..i,k\}} \Phi_j \\ d_{i-1} Z_{i+1..n}^T Y_{i+1..n} = \frac{1}{d_i} \sum_{j,k=i+1}^n \Lambda_k d_{i-1} [A]_{\{1..i,j\}}^{\{1..i,k\}} \Psi_j \end{cases}$$

and H_{i+1} is verified.

We have proven that if H_i is true, then either H_{i+1} is also true or the system (3) has a single solution and the Verifier randomly chose precisely that λ_i. Therefore, suppose that A has not generic rank profile, it means that some $d_j = 0$ and H_j is false. But the Verifier checks that H_1 is true. If this is the case, then at least once, did the Verifier choose the value expected by the dishonest Prover. This happens with probability lower than $1/|S|$.

Finally, for the complexity, the Prover needs one Gaussian elimination to compute LU in time $O(n^\omega)$, then her extra work is just three triangular solve in $O(n^2)$. The extra communication is three vectors, ϕ, ψ, λ, and the Verifier's work is four dot-products and one multiplication by the initial matrix A. □

3.3 LDUP decomposition

With Protocol 5, when the matrix A does not have generic rank profile, any attempt to prove that it has generic rank profile will be detected w.h.p. (soundness). However when it is the case, the verification will accept many possible vectors x, y, z: any scaling of z_i by α_i and x_i, y_i by $1/\alpha_i$ would be equally accepted for any non zero constants α_i. This slack correspond to our lack of specification of the diagonals' shape in the used LU decomposition. Indeed, for any diagonal matrix with non zero elements, $LD \times D^{-1}U$ is also a valid LU decomposition and yields x, y and z scaled as above. Specifying these diagonals is not necessary to prove generic rank profileness, so we left it as is for this task.

However, for the determinant or the rank profile matrix certificates of Sections 4.1 and 4.3, we will need to ensure that this scaling is independent from the choice of the vectors ϕ, ψ, λ. Hence we propose an updated protocol, where L has to be unit diagonal, and the prover has to first commit the main diagonal D of U.

For an $n \times n$ triangular matrix T, its strictly triangular part is denoted $\widetilde{T} \in \mathbb{F}^{(n-1)\times(n-1)}$: for instance if T is upper triangular, then $\widetilde{t}_{i,j} = t_{i,j+1}$ for $j \ge i$ and 0 otherwise.

For U an invertible upper triangular matrix we have for its diagonal $(d_1, \ldots, d_n)$ and the associated diagonal matrix D, that $U_1 = D^{-1}U$ is unitary. Thus, for any $\mathbb{F}^n \ni \psi = [\psi_1, \widetilde{\psi}]^T$: $U\psi = DU_1\psi = D\left(\psi + \begin{bmatrix} \widetilde{U_1}\widetilde{\psi} \\ 0 \end{bmatrix}\right)$.

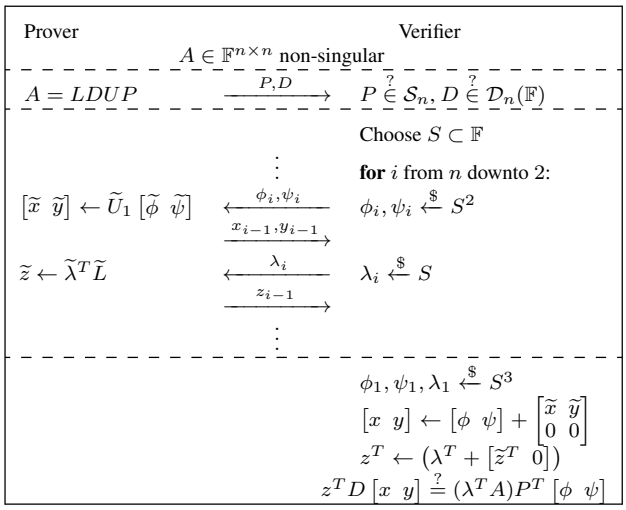
So the idea is that the Prover will commit D beforehand, and that within a generic rank profile certificate, the Verifier will only communicate $\widetilde{\phi}, \widetilde{\psi}$ and $\widetilde{\lambda}$ to obtain $\widetilde{z} = \widetilde{\lambda}^T \widetilde{L}$, $\widetilde{x} = \widetilde{U}_1 \widetilde{\phi}$ and $\widetilde{y} = \widetilde{U}_1 \widetilde{\psi}$. Then the Verifier will compute by herself the complete vectors. This ensures that L is unitary and that $U = DU_1$ with U_1 unitary.

Finally, if an invertible matrix does not have generic rank profile, we note that it is also possible to incorporate the permutations, by committing them in the beginning and reapplying them to the matrix during the checks. The full certificate is given in Figure 6.

THEOREM 5. *The Protocol of Figure 6, committing a permutation matrix P and a diagonal matrix D for an invertible matrix A, such that there exists unitary triangular matrices L and U with $A = LDUP$, is sound, with probability larger than $1 - \frac{1}{|S|}$, and perfectly complete. For an $n \times n$ matrix, it requires less than $8n$ extra communications and the computational cost for the Verifier is bounded by $\mu(A) + 12n + o(n)$.*

PROOF. If the Prover is honest, then $A = LUP = LDU_1P$, so that for any choice of λ and ψ we have: $\lambda^T A P^T \psi = \lambda^T L D U_1 \psi$, that is $\begin{bmatrix} \widetilde{\lambda}^T & \lambda_n \end{bmatrix} \left(I + \begin{bmatrix} 0 & 0 \\ \widetilde{L} & 0 \end{bmatrix}\right) D \left(\begin{bmatrix} 0 & \widetilde{U} \\ 0 & 0 \end{bmatrix} + I\right) \begin{bmatrix} \widetilde{\psi} \\ \psi_n \end{bmatrix} = z^T y$ and the same is true for λ and ϕ, so that the protocol is perfectly complete.

Now, the last part of the Protocol of Figure 6 is actually a verification that AP^T has generic rank profile, in other words that there exists lower and upper triangular matrices L^* and U^* such that $AP^TA = L^*U^*$. This verification is sound by Theorem 4. Next, the multiplication by the diagonal D is performed by the Verifier, so he is actually convinced that there exists lower and upper triangular matrices L^* and U_1^* such that $AP^T = L^*DU_1^*$. Finally, the



Protocol 6: LDUP decomposition (linear communication)

construction of the vectors with the form $a + \begin{bmatrix}\widetilde{b}\\ 0\end{bmatrix}$ is also done by the Verifier, so he in fact has a guaranty that L^* and U_1^* are unitary.

Overall, if the Prover is dishonest, the Verifier will catch him with the probability of Theorem 4.

Finally, for the complexity bounds, the extra communications are: one permutation matrix P, a diagonal matrix D and 6 vectors $\widetilde{\lambda}$, $\widetilde{\phi}$, $\widetilde{\psi}$ and $\widetilde{z}$, $\widetilde{x}$ and $\widetilde{y}$. That is n non-negative integers lower than n and $6(n-1)+n$ field elements. The arithmetic computations of the Verifier are one multiplication by a diagonal matrix, 3 vector sums, 4 dot-products and one matrix-vector multiplication by A (for $(\lambda^T A)$), that is $n + 3(n-1) + 4(2n-1)$. □

We, furthermore, have some guaranties on the actual values of x, y, z:

PROPOSITION 1. *Let S be a finite subset of $\mathbb{F}$ in Protocol 6, if $\begin{bmatrix}x & y\end{bmatrix} \neq U_1 \begin{bmatrix}\phi & \psi\end{bmatrix}$ then the verification will pass with probability at most $\frac{2}{|S|}$.*

PROOF. Equation (4) implies that, if the verification check passes, with (z, x, y), then the vector $\begin{bmatrix}x_i & y_i\end{bmatrix}^T$ must be co-linear with the right column of this determinant, that can be written in the form $\begin{bmatrix}d_i\phi_i + b & d_i\psi_i + c\end{bmatrix}^T$ with $d_i \neq 0$ and b and c depending only on $\phi_k, \psi_k, x_k, y_k, \lambda_k, z_k$ with $k > i$. Hence, any value $\widetilde{x}_i, \widetilde{y}_i$, supplied by the Prover, must satisfy

$$\begin{vmatrix}\phi_i + \widetilde{x}_i & d_i\phi_i + b\\ \psi_i + \widetilde{y}_i & d_i\psi_i + c\end{vmatrix} = 0, \tag{6}$$

when ϕ_i and ψ_i are still unknown. This condition is ensured for any ϕ_i and ψ_i if and only if $\begin{bmatrix}\widetilde{x}_i & \widetilde{y}_i\end{bmatrix} = \frac{1}{d_i}\begin{bmatrix}b & c\end{bmatrix}$. If the Prover is dishonest and if $\begin{bmatrix}x & y\end{bmatrix} \neq U_1 \begin{bmatrix}\phi & \psi\end{bmatrix}$ then at least one couple $(\widetilde{x}_i, \widetilde{y}_i)$ is incorrect. Then, either the Verifier has chosen a couple of values (ϕ_i, ψ_i) making the degree 1 determinant (6) vanish, this happens with probability at most $1/|S|$, or System (3) has a unique solution (z_i, λ_i). But if the latter is true and the final check succeeds then, as for Theorem 4, at least once the Prover chose to have $1/|S|$ chances that the Verifier picked the unique possibility for λ_j, $i \geq j \geq 1$. Overall, the Verification thus fails with probability at most $1 - \frac{2}{|S|}$. □

REMARK 1. *Correctness of the vector z can also be ensured with the same probability: for the singular System (3), with respect to the unknowns Λ_i and Z_i, to have rank at least one, it is sufficient that one of X_i or Y_i is non zero. The Verifier, knowing $\widetilde{x}_i$, can ensure this by restricting the set of choices for $\phi_i \in S\backslash\{-\widetilde{x}_i\}$. Thus if x_i and y_i are correct, the Prover will have to provide a correct associated z_i or increase the probability of being caught.*

4. LINEAR COMMUNICATION INTERACTIVE CERTIFICATES

In this section, we give linear space communication certificates for the determinant, the column/row rank profile of a matrix, and for the rank profile matrix.

4.1 Linear communication certificate for the determinant

Existing certificates for the determinant are either optimal for the Prover in the dense case, using the strategy of [14, Theorem 5] over a PLUQ decomposition, but quadratic in communication; or linear in communication, using [5, Theorem 14], but using a reduction to the characteristic polynomial. In the sparse case the determinant and the characteristic polynomial both reduce to the same minimal polynomial computations and therefore the latter certificate is currently optimal for the Prover. Now in the dense case, while the determinant and characteristic polynomial both reduce to matrix multiplication, the determinant, via a single PLUQ decomposition is more efficient in practice [15]. Therefore, we propose here an alternative in the dense case: use only one PLUQ decomposition for the Prover while keeping linear extra communications and $O(n) + \mu(A)$ operations for the Verifier. The idea is to extract the information of a PLDU decomposition without communicating it: one uses Protocol 6 for $A = PLDU$ with L and U unitary, but kept on the Prover side, and then the Verifier only has to compute $Det(A) = Det(P)Det(D)$, with $n-1$ additional field operations.

COROLLARY 1. *For an $n\times n$ matrix, there exists a sound and perfectly complete protocol for the determinant over a field using less than $8n$ extra communications and with computational cost for the Verifier bounded by $\mu(A) + 13n + o(n)$.*

As a comparison, the protocol of [5, Theorem 14] reduces to CHARPOLY instead of PLUQ for the Prover, requires $5n$ extra communications and $\mu(A) + 13n + o(n)$ operations for the Verifier as well. Also the new protocol requires $3n$ random field elements for any field, where that of [5, Theorem 14] requires 3 random elements but a field larger than n^2.

For instance, using the routines shown in Table 1, the determinant of an $50k\times 50k$ random dense matrix can be computed in about 24 minutes, where with the certificate of Figure 6, the overhead of the Prover is less than 5s and the Verifier time is about 1s.

4.2 Column or row rank profile certificate

In Figure 7 and 8, we first recall the two linear time and space certificates for an upper and a lower bound to the rank that constitute a rank certificate. We present here the variant sketched in [9, § 2] of the certificates of [4]. An upper bound r on the rank is certified by the capacity for the Prover to generate any vector sampled from the image of A by a linear combination of r column of A. A lower bound r is certified by the capacity for the Prover to recover the unique coefficients of a linear combination of r linearly independent columns of A.

THEOREM 6. *Let $A \in \mathbb{F}^{m\times n}$, and let S be a finite subset of $\mathbb{F}$. The interactive certificate 7 of an upper bound for the rank of A is*

Prover		Verifier
	$A \in \mathbb{F}^{m\times n}$	
r s.t. $\text{rank}(A) \le r$	$\xrightarrow{r}$	
		Choose $S \subset \mathbb{F}$
	$\xleftarrow{w}$	$v \xleftarrow{\$} S^n, w = Av$
$A\gamma = w$	$\xrightarrow{\gamma}$	$\lvert\gamma\rvert_H \stackrel{?}{=} r$
		$A\gamma \stackrel{?}{=} w$

Protocol 7: Upper bound on the rank of a matrix

sound, with probability larger than $1 - \frac{1}{|S|}$*, perfectly complete, occupies* $2n$ *communication space, can be computed in* $LINSYS(r)$ *and verified in* $2\mu(A) + n$ *time.*

Prover		Verifier
	$A \in \mathbb{F}^{m\times n}$	
$c_1, .., c_r$ indep. cols of A	$\xrightarrow{c_1,..,c_r}$	
		Choose $S \subset \mathbb{F}$
	$\xleftarrow{v}$	$\alpha = \begin{cases} \alpha_{c_j} \xleftarrow{\$} S^* \\ 0 \text{ otherwise} \end{cases}$
		$v = A\alpha$
Solve $A\beta = v$	$\xrightarrow{\beta}$	$\beta \stackrel{?}{=} \alpha$

Protocol 8: Lower bound on the rank of a matrix

THEOREM 7. *Let* $A \in \mathbb{F}^{m\times n}$*, and let* S *be a finite subset of* $\mathbb{F}$*. The interactive certificate 8 of a lower bound for the rank of* A *is sound, , with probability larger than* $1 - \frac{1}{|S|}$*, perfectly complete and occupies* $n + 2r$ *communication space, can be computed in* $LINSYS(r)$ *and verified in* $\mu(A) + r$ *operations.*

We now consider a column rank profile certificate: the Prover is given a matrix A, and answers the column rank profile of A, $\mathcal{J} = (c_1, \ldots, c_r)$. In order to certify this column rank profile, we need to certify two properties:

1. the columns given by $\mathcal{J}$ are linearly independent;
2. the columns given by $\mathcal{J}$ form the lexicographically smallest set of independent columns of A.

Property 1 is verified by Certificate 8, as it checks wether a set of columns are indeed linearly independent. Property 2 could be certified by successive applications of Certificate 7: at step i, checking that the rank of $A_{*,(0,\ldots,c_i-1)}$ is at most $i-1$ would certify that there is no column located between c_{i-1} and c_i in A which increases the rank of A. Hence, it would prove the minimality of $\mathcal{J}$. However, this method requires $O(nr)$ communication space.

Instead, we reduce these communication by seeding all challenges from a single n dimensional vector, and by compressing the responses with a random projection. The right triangular equivalence certificate plays here a central role, ensuring the lexicographic minimality of $\mathcal{S}$. More precisely, the Verifier chooses a vector $v \in \mathbb{F}^n$ uniformly at random and sends it to the Prover. Then, for each index $c_k \in \mathcal{S}$ the Prover computes the linear combination of the first $c_k - 1$ columns of A using the first $c_k - 1$ coefficients of v and has to prove that it can be generated from the $k-1$ columns $c_1, \ldots, c_{k-1}$. This means, find a vector $\gamma^{(k)}$ solution to the system:

$$\begin{bmatrix} A_{*,c_1} & A_{*,c_2} & \ldots & A_{*,c_{k-1}} \end{bmatrix} \gamma^{(k)} = A \begin{bmatrix} v_1 \\ \vdots \\ v_{c_{k-1}} \\ 0 \\ \vdots \end{bmatrix}.$$

Equivalently, find a strictly upper triangular matrix Γ such that:

$$\begin{bmatrix} A_{*,c_1} & A_{*,c_2} & \ldots & A_{*,c_{r-1}} \end{bmatrix} \Gamma = A \underbrace{\begin{bmatrix} v_1 & v_1 & \cdots & \cdots & v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{c_1-1} & \vdots & \vdots & \vdots & \vdots \\ 0 & v_{c_2-1} & \vdots & \vdots & \vdots \\ 0 & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & v_{c_r-1} & \vdots \\ 0 & 0 & 0 & 0 & v_n \end{bmatrix}}_{V}.$$

Note that $V = \text{Diag}(v_1, \ldots, v_n)W$ where $W = [\mathbb{1}_{i<c_{j+1}}]_{i,j}$ (with $c_{r+1} = n+1$ by convention) In order to avoid having to transmit the whole $r \times r$ upper triangular matrix Γ, the Verifier only checks a random projection x of it, using the triangular equivalence Certificate 4. We then propose the certificate in Figure 9.

Prover		Verifier
	$A \in \mathbb{F}^{m\times n}$	
$(c_1, .., c_r)$ CRP of A	$\xrightarrow{(c_1,..,c_r)}$	rank $A \stackrel{?}{\ge} r$ by Cert. 8
		Choose $S \subset \mathbb{F}$
	$\xleftarrow{v}$	$v \xleftarrow{\$} S^n$
$V = \text{Diag}(v_i)W$		$W = [\mathbb{1}_{i<c_{j+1}}]$
Γ upper tri. s.t.		$D \leftarrow \text{Diag}(v_i)$
$A_{*,\{c_1,..,c_r\}}\Gamma = AV$		
$y = \Gamma x$	$\xleftrightarrow{x \text{ (Cert. 4) } y}$	$x \xleftarrow{\$} S^r$
		$z \leftarrow D(Wx)$
		$z_{c_j} \leftarrow z_{c_j} - y_j, j = 1..r$
		$Az \stackrel{?}{=} 0$

Protocol 9: Certificate for the column rank profile

THEOREM 8. *For* $A \in \mathbb{F}^{m\times n}$ *and* $S \subset \mathbb{F}$*, certificate 9 is sound, with probability larger than* $1 - \frac{1}{|S|}$*, perfectly complete, with a Prover computational cost bounded by* $O(mnr^{\omega-2})$*, a communication space complexity bounded by* $2n + 4r$ *and a Verifier cost bounded by* $2\mu(A) + n + 3r$*.*

PROOF. If the Prover is honest, the protocol corresponds first to an application of Theorem 7 to certify that $\mathcal{J}$ is a set of independent columns. This certificate is perfectly complete. Second the protocol also uses challenges from Certificate 7, which is perfectly complete, together with Certificate 4, which is perfectly complete as well. The latter certificate is used on $A_{*,\mathcal{J}}$, a regular submatrix, as $\mathcal{J}$ is a set of independent columns of A. The final check then corresponds to $A(D(Wx)) - A_{*,\{c_1,..c_r\}}y \stackrel{?}{=} 0$ and, overall, Certificate 9 is perfectly complete.

If the Prover is dishonest, then either the set of columns in $\mathcal{J}$ are not linearly independent, which will be caught by the Verifier with probability at least $1 - \frac{1}{|S|}$, from Theorem 7, or $\mathcal{J}$ is not lexicographically minimal, or the rank of A is not r. If the rank is wrong, it will not be possible for the prover to find a suitable Γ. This will be caught by the verifier with probability $1 - \frac{1}{|S|}$, from Theorem 3. Finally, if $\mathcal{J}$ is not lexicographically minimal, there exists at least one column $c_k \notin \mathcal{J}, c_i < c_k < c_{i+1}$ for some fixed

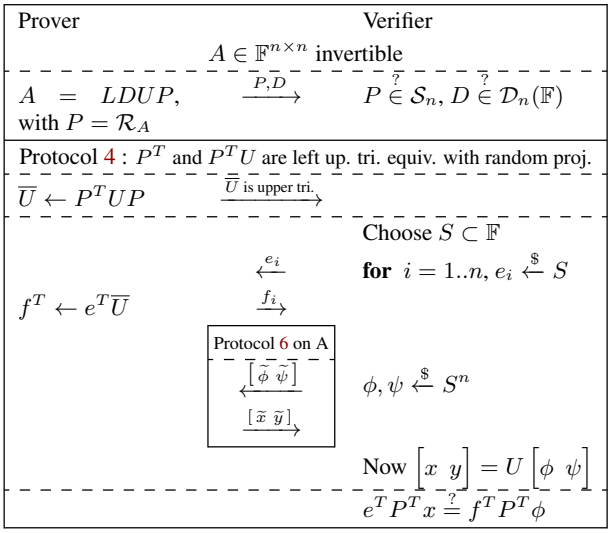
i such that $\{c_1, \ldots, c_i\} \cup \{c_k\}$ form a set of linearly independant columns of A. This means that $\text{rank}(A_{*,1,\ldots,c_{i+1}-1}) = i+1$, whereas it was expected to be i. Thus, the prover cannot reconstruct a suitable triangular Γ and this will be detected by the verifier also with probability $1 - \frac{1}{|S|}$, as shown in Theorem 3).

The Prover's time complexity is that of computing a $PLUQ$ decomposition of A. The transmission of v, x and y yields a communication space of $n + 2r$. Finally, in addition to Protocol 8, the Verifier computes Wx as a prefix sum with $r-1$ additions, multiplies it by D, then substracts y_i at the r correct positions and finally multiplies by A for a total cost bounded by $2\mu(A)+n+3r-1$. □

4.3 Rank profile matrix certificate

We propose an interactive certificate for the rank profile matrix based on [8, Algorithm 4]: first computing the row and column support of the rank profile matrix, using Certificate 9 twice for the row and column rank profiles, then computing the rank profile matrix of the invertible submatrix of A lying on this grid.

In the following we then only focus on a certificate for the rank profile matrix of an invertible matrix. It relies on an LUP decomposition that reveals the rank profile matrix. From Theorem 2, this is the case if and only if P^TUP is upper triangular. Protocol 10 thus gives an interactive certificate that combines Certificate 6 for a LDUP decomposition with a certificate that P^TUP is upper triangular. The latter is achieved by Certificate 4 showing that P^T and P^TU are left upper triangular equivalent, but since U is unknown to the Verifier, the verification is done on a random right projection with the vector ϕ used in Certificate 6.



Protocol 10: Rank profile matrix of an invertible matrix

THEOREM 9. *Protocol 10 is sound, with probability greater than* $1 - \frac{2}{|S|}$, *and perfectly complete. The Prover cost is* $O(n^\omega)$ *field operations, the communication space is bounded by* $10n$ *and the Verifier cost is bounded by* $\mu(A) + 16n$.

PROOF. If the Prover is dishonest and $\overline{U} = P^TUP$ is not upper triangular, then let (i, j) be the lexicographically minimal coordinates such that $i > j$ and $\overline{U}_{i,j} \neq 0$. Now either $\begin{bmatrix} x & y \end{bmatrix} \neq U \begin{bmatrix} \phi & \psi \end{bmatrix}$, and the verification will then fail to detect it with probability less than $\frac{2}{|S|}$, from Proposition 1. Or one can write $e^TP^Tx - f^TP^T\phi = (e^T\overline{U} - f^T)P\phi = 0$. If

$$e^TP^TUP - f^T = 0. \tag{7}$$

is not satisfied, then a random ϕ will fail to detect it with probability less than $\frac{1}{|S|}$, since $e, \overline{U}$ and f are set before choosing for ϕ. At the time of commiting f_j, the value of e_i is still unknown, hence f_j is constant in the symbolic variable E_i. Thus the j-th coordinate in (7) is a nonzero polynomial in E_j and therefore vanishes with probability $1/|S|$ when sampling the values of e uniformly. Hence, overall if P^TUP is not upper triangular, the verification will fail to detect it with probability at most $2/|S|$. □

Finally, the rank profile matrix of any matrix, even a singular one, can thus be verified with two applications of Certificate 9 (one for the row rank profile and one for the column rank profile, themselves calling Certificate 8 only once), followed by Certificate 10 on the $r \times r$ selection of lexicographically minimal independent rows and columns. Overall this is $4\mu(A) + 2n + 21r$ operations for the Verifier, and $3n + 16r$ communications.

5. REFERENCES

[1] H. Y. Cheung, T. C. Kwok, and L. C. Lau. Fast Matrix Rank Algorithms and Applications. *Journal of the ACM*, 60(5):31:1–31:25, Oct. 2013.

[2] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 253–270, 2015.

[3] C. L. Dodgson. Condensation of Determinants, Being a New and Brief Method for Computing their Arithmetical Values. *Proceedings of the Royal Society of London*, 15:150–155, 1866.

[4] J.-G. Dumas and E. Kaltofen. Essentially optimal interactive certificates in linear algebra. In K. Nabeshima, editor, *ISSAC'2014*, pages 146–153. ACM Press, New York, July 2014.

[5] J.-G. Dumas, E. Kaltofen, E. Thomé, and G. Villard. Linear time interactive certificates for the minimal polynomial and the determinant of a sparse matrix. In X.-S. Gao, editor, *ISSAC'2016*, pages 199–206. ACM Press, New York, July 2016.

[6] J.-G. Dumas, C. Pernet, and Z. Sultan. Simultaneous computation of the row and column rank profiles. In M. Kauers, editor, *ISSAC'2013*, pages 181–188. ACM Press, New York, June 2013.

[7] J.-G. Dumas, C. Pernet, and Z. Sultan. Computing the rank profile matrix. In K. Yokoyama, editor, *ISSAC'2015*, pages 149–156. ACM Press, New York, July 2015.

[8] J.-G. Dumas, C. Pernet, and Z. Sultan. Fast computation of the rank profile matrix and the generalized Bruhat decomposition. *Journal of Symbolic Computation*, 2016. in press.

[9] W. Eberly. A new interactive certificate for matrix rank. Technical Report 2015-1078-11, University of Calgary, June 2015.

[10] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1987, 11–15 Aug. 1986.

[11] R. Freivalds. Fast probabilistic algorithms. *Mathematical Foundations of Computer Science, LNCS*, 74:57–69, Sept. 1979.

[12] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In C. Dwork, editor, *STOC'2008*, pages 113–122. ACM Press, May 2008.

[13] C.-P. Jeannerod, C. Pernet, and A. Storjohann. Rank-profile revealing gaussian elimination and the CUP matrix decomposition. *Journal of Symbolic Computation*, 56:pages 46–68, 2013.

[14] E. L. Kaltofen, M. Nehring, and B. D. Saunders. Quadratic-time certificates in linear algebra. In A. Leykin, editor, *ISSAC'2011*, pages 171–176. ACM Press, New York, June 2011.

[15] C. Pernet and A. Storjohann. Faster algorithms for the characteristic polynomial. In C. W. Brown, editor, *ISSAC'2007*, pages 307–314. ACM Press, New York, July 29 – August 1 2007.

[16] A. Storjohann and S. Yang. A Relaxed Algorithm for Online Matrix Inversion. In K. Yokoyama, editor, *ISSAC'2015*, pages 339–346. ACM Press, New York, July 2015.

Disclaimer: this report, together with its annexes and the reports for the earlier deliverables, is self contained for auditing and reviewing purposes. Hyperlinks to external resources are meant as a convenience for casual readers wishing to follow our progress; such links have been checked for correctness at the time of submission of the deliverable, but there is no guarantee implied that they will remain valid.