Computational linear algebra over finite fields

Jean-Guillaume Dumas, Clément Pernet

Computational linear algebra over finite fields

Jean-Guillaume Dumas* Clément Pernet†

April 17, 2012

*Université de Grenoble; Laboratoire Jean Kuntzmann, (umr CNRS 5224, Grenoble INP, INRIA, UJF, UPMF); Jean-Guillaume.Dumas@imag.fr; 51, rue des Mathématiques, BP 53X, F-38041 Grenoble, France.

†INRIA, Université de Grenoble; Laboratoire LIG (umr CNRS 5217, Grenoble INP, INRIA, UJF, UPMF); Clement.Pernet@imag.fr; ENSIMAG Antenne de Montbonnot, 51, avenue Jean Kuntzmann, F-38330 Montbonnot Saint-Martin, France.

We present here algorithms for efficient computation of linear algebra problems over finite fields. Implementations[1] of the proposed algorithms are available through the MAGMA, MAPLE (within the `LinearAlgebra[Modular]` subpackage) and SAGE systems; some parts can also be found within the C/C++ libraries NTL, FLINT, IML, M4RI and the special purpose LINBOX template library for exact, high-performance linear algebra computation with dense, sparse, and structured matrices over the integers and over finite fields [16].

1 Dense matrix multiplication

Definition 1 *For $A \in \mathbb{F}_q^{m\times k}$ and $B \in \mathbb{F}_q^{k\times n}$ with elements $A_{i,j}$ and $B_{i,j}$, the matrix $C = A \times B$ has $C_{i,j} = \sum_{l=1}^{k} A_{i,l}B_{l,j}$. We denote by $\mathtt{MM}(m,k,n)$ a time complexity bound on the number of field operations necessary to compute C.*

Classical triple loop implementation of matrix multiplication makes $\mathtt{MM}(m,k,n) \leq 2mkn$. The best published estimates to date gives $\mathtt{MM}(n,n,n) \leq \mathcal{O}(n^\omega)$ with $\omega \approx 2.3755$ [13], though improvements to 2.3737 and 2.3727 are now claimed [50, 53]. For very rectangular matrices one also have astonishing results like $\mathtt{MM}(n,n,n^\alpha) \leq \mathcal{O}(()\, n^{2+\epsilon})$ for a constant $\alpha > 0.294$ and any $\epsilon > 0$ [12]. Nowadays practical implementations mostly use Strassen-Winograd's algorithm, see section 1.4, with an intermediate complexity and $\omega \approx 2.8074$.

1.1 Tiny finite fields

The practical efficiency of matrix multiplication depends highly on the representation of field elements. We thus present three kinds of compact representations for elements of a finite field with very small cardinality: bitpacking (for $\mathbb{F}_2$), bit-slicing (for say $\mathbb{F}_3, \mathbb{F}_5, \mathbb{F}_7, \mathbb{F}_{2^3}$, or $\mathbb{F}_{3^2}$) and Kronecker substitution. These representations are designed to allow efficient linear algebra operations, including matrix multiplication.

Definition 2 *[5] Bitslicing consists in representing an n-dimensional vector of k-bit sized coefficients using k binary vectors of dimension n. In particular, one can use boolean word instruction to perform arithmetic on 64 dimensional vectors.*

- *Over $\mathbb{F}_3$, the binary representation $0 \equiv [0,0], 1 \equiv [1,0], -1 \equiv [11]$ allows to add and subtract two elements in 6 boolean operations:*

$$\begin{array}{ll} \mathit{Add}([x_0,x_1],[y_0,y_1]): & s \leftarrow x_0 \oplus y_1, t \leftarrow x_1 \oplus y_0 \\ & \mathit{Return}(s \wedge t, (s \oplus x_1) \vee (t \oplus y_1)) \\ \mathit{Sub}([x_0,x_1],[y_0,y_1]): & t \leftarrow x_0 \oplus y_0 \\ & \mathit{Return}(t \vee (x_1 \oplus y_1), (t \oplus y_1) \wedge (y_0 \oplus x_1)) \end{array}$$

[1] `http://magma.maths.usyd.edu.au`, `http://www.maplesoft.com`, `http://sagemath.org`, `http://www.shoup.net/ntl`, `http://www.flintlib.org`, `http://www.cs.uwaterloo.ca/~astorjoh/iml.html`, `http://m4ri.sagemath.org`, `http://linalg.org`

Algorithm 1.1 [Greasing]

Over $\mathbb{F}_2$, the method of *the four Russians* [2], also called *Greasing* can be used as follows:

- A 64 bit machine word can be used to represent a row vector of dimension 64.
- Matrix multiplication of a $m \times k$ matrix A by a $k \times n$ matrix B can be done by first storing all 2^k k-dimensional linear combinations of rows of B in a table. Then the i-th row of the product is copied from the row of the table indexed by the i-th row of A.
- By ordering indices of the table according to a binary Gray Code, each row of the table can be deduced from the previous one, using only one row addition. This brings the bit operation count to build the table from $k2^k n$ to $2^k n$.
- Choosing $k = \log_2 n$ in the above method implies $\mathtt{MM}(n) = \mathcal{O}\left(n^3/\log n\right)$ over $\mathbb{F}_2$.

- *Over $\mathbb{F}_5$ (resp. $\mathbb{F}_7$), a redundant representation $x = x_0 + 2x_1 + 4x_2 \equiv [x_0, x_{1,2}]$ allows to add two elements in 20 (resp. 17) boolean operations, negate in 3 (resp. 6) boolean operations and double in 0 (resp. 5) boolean operations.*

	$\mathbb{F}_3$	$\mathbb{F}_5$	F_7
Addition	6	20	17
Negation	1	5	3
Double		5	0

Table 1: boolean operation counts for basic arithmetic using bit slicing

Definition 3 *Bitpacking consists in representing a vector of field elements as an integer fitting in a single machine word using a 2^k-adic representation:*

$$(x_0, \ldots, x_{n-1}) \in \mathbb{F}_q^n \equiv X = x_0 + 2^k x_1 + \cdots + (2^k)^{n-1} x_{n-1} \in \mathbb{Z}_{2^{64}}$$

Elements of extension fields are viewed as polynomials and stored as the evaluation of this polynomial at the characteristic of the field. The latter evaluation is called Kronecker substitution.

We first need a way to simultaneously reduce coefficients modulo the characteristic, see [14].

Once we can pack and simultaneously reduce coefficients of finite field in a single machine word, the obtained parallelism can be used for matrix multiplication. Depending on the respective sizes of the matrix in the multiplication one can pack only the left operand or only the right one or both [15]. We give here only a generic

Algorithm 1.2 [REDQ: Q-adic REDuction]

Input: Three integers p, q and $\tilde{r} = \sum_{i=0}^{d} \widetilde{\mu}_i q^i \in \mathbb{Z}$.
Output: $\rho \in \mathbb{Z}$, with $\rho = \sum_{i=0}^{d} \mu_i q^i$ where $\mu_i = \widetilde{\mu}_i \bmod p$.

REDQ COMPRESSION

1: $s = \left\lfloor \frac{\tilde{r}}{p} \right\rfloor$;
2: **for** $i = 0$ to d **do**
3: $\quad u_i = \left\lfloor \frac{\tilde{r}}{q^i} \right\rfloor - p \left\lfloor \frac{s}{q^i} \right\rfloor$;
4: **end for**

REDQ CORRECTION {only when $p \nmid q$, otherwise $\mu_i = u_i$ is correct}

5: $\mu_d = u_d$;
6: **for** $i = 0$ to $d - 1$ **do**
7: $\quad \mu_i = u_i - q u_{i+1} \bmod p$;
8: **end for**
9: Return $\rho = \sum_{i=0}^{d} \mu_i q^i$;

Algorithm 1.3 [Right packed matrix multiplication]

Input: A prime p and $A_c \in \mathbb{F}_p^{m \times k}$ and $B_c \in \mathbb{F}_p^{k \times n}$, stored with several field elements per machine word.
Output: $C_c = A_c \times B_c \in \mathbb{F}_p^{m \times n}$

1: $A = \text{Uncompress}(A_c)$; {extract the coefficients}
2: $C_c = A \times B_c$; {Using e.g., algorithm 1.5}
3: Return REDQ(C_c);

algorithm for packed matrices, which use multiplication of a right packed matrix by a non packed left matrix.

Then, over extensions, fast floating point operations can be used on the Kronecker substitution of the elements. Indeed, it is very often desirable to use floating point arithmetic, *exactly*. For instance floating point routines can more easily use large hardware registers, they can more easily optimize the memory hierarchy usage [30, 57] and portable implementations are more widely available. We present next the dot product and the matrix multiplication is then straightforward [17, 14, 15].

Algorithm 1.4 [Compressed Dot product over extension fields]

Input: A field $\mathbb{F}_{p^k}$ with elements represented as exponents of a generator of the field;
Input: two vectors v_1 and v_2 of elements of $\mathbb{F}_{p^k}$;
Input: a sufficiently large integer q.
Output: $R \in \mathbb{F}_{p^k}$, with $R = v_1^T \cdot v_2$.
{Tabulated conversion: uses tables from exponent to floating point evaluation}
1: Set $\widetilde{v_1}$ and $\widetilde{v_2}$ to the floating point Kronecker substitution of the elements of v_1 and v_2.
2: Compute $\tilde{r} = \widetilde{v_1}^T \cdot \widetilde{v_2}$; {The floating point computation}
3: $r = REDQ\_COMPRESSION(\tilde{r}, p, q)$; {Computing a radix decomposition}
{Variant of REDQ_CORRECTION: $\mu_i = \widetilde{\mu_i} \bmod p$ for $\tilde{r} = \sum_{i=0}^{2k-2} \widetilde{\mu_i} q^i$}
4: Set $L = representation(\sum_{i=0}^{k-2} \mu_i X^i)$;
5: Set $H = representation(X^{k-1} \times \sum_{i=k-1}^{2k-2} \mu_i X^{i-k+1})$;
6: Return $R = H + L \in \mathbb{F}_{p^k}$; {Reduction in the field}

1.2 Word size prime fields

Over word-size prime fields one can also use the reduction to floating point routines of algorithm 1.4. The main point is to be able to perform efficiently the matrix multiplication of blocks of the initial matrices without modular reduction. Thus delaying the reduction as much as possible, depending on the algorithm and internal representations, in order to amortize its cost. We present next such a delaying with the classical matrix multiplication algorithm and a centered representation [18].

1.3 Large finite fields

If the field is too large for the strategy 1.5 over machine words, then two main approaches would have to be considered:

- Use extended arithmetic, either arbitrary of fixed precision, if the characteristic is large, and a polynomial representation for extension fields. The difficulty here is to preserve an optimized memory management and to have an almost linear time extended precision polynomial arithmetic.

Algorithm 1.5 [fgemm: Finite Field GEneric Matrix Multiplication]

Input: An odd prime p of size smaller than the floating point mantissa β and F_p elements stored by values between $\frac{1-p}{2}$ and $\frac{p-1}{2}$
Input: $A \in \mathbb{F}_p^{m \times k}$ and $B \in \mathbb{F}_p^{k \times n}$
Output: $C = A \times B \in \mathbb{F}_p^{m \times n}$

1: **if** $n(p-1)^2 < 2^{\beta+1}$ **then**
2: Convert A and B to floating point matrices A_f and B_f;
3: Use floating point routines to compute $C_f = A_f \times B_f$;
4: $C = C_f \mod p$;
5: **else**
6: Cut A and B into smaller blocks;
7: Call the algorithm recursively for the block multiplications;
8: Perform the block additions modulo p;
9: **end if**

- Use a residue number system and an evaluation/interpolation scheme: one can use algorithm 1.5 for each prime in the RNS and each evaluation point. For $\mathbb{F}_{p^k}$, the number of needed primes is roughly $2\log_{2^\beta}(p)$ and the number of evaluations points is $2k-1$.

1.4 Large matrices: subcubic time complexity

With matrices of large dimension, sub-cubic time complexity algorithms, such as Strassen-Winograd's [59] can be used to decrease the number of operations. Algorithm 1.4 describes how to compute one recursive level of the algorithm, using seven recursive calls and 15 block additions.

Algorithm 1.6 [Strassen-Winograd]

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}; B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}; C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix};$$

$$\begin{array}{llll} S_1 \leftarrow A_{21} + A_{22}; & T_1 \leftarrow B_{12} - B_{11}; & P_1 \leftarrow A_{11} \times B_{11}; & P_2 \leftarrow A_{12} \times B_{21}; \\ S_2 \leftarrow S_1 - A_{11}; & T_2 \leftarrow B_{22} - T_1; & P_3 \leftarrow S_4 \times B_{22}; & P_4 \leftarrow A_{22} \times T_4; \\ S_3 \leftarrow A_{11} - A_{21}; & T_3 \leftarrow B_{22} - B_{12}; & P_5 \leftarrow S_1 \times T_1; & P_6 \leftarrow S_2 \times T_2; \\ S_4 \leftarrow A_{12} - S_2; & T_4 \leftarrow T_2 - B_{21}; & P_7 \leftarrow S_3 \times T_3; & \end{array}$$

$$\begin{array}{llll} C_{11} \leftarrow P_1 + P_2; & U_2 \leftarrow P_1 + P_6; & U_3 \leftarrow U_2 + P_7; & U_4 \leftarrow U_2 + P_5; \\ C_{12} \leftarrow U_4 + P_3; & C_{21} \leftarrow U_3 - P_4; & C_{22} \leftarrow U_3 + P_5; & \end{array}$$

In practice, one uses a threshold in the matrix dimension to switch to a base case algorithm, that can be any of the one previously described. Following section 1.2, one can again delay the modular reductions, but the intermediate computations of Strassen-Winograd's algorithm impose a tighter bound:

Theorem 4 *[18] Let $A \in \mathbb{Z}^{m\times k}$, $B \in \mathbb{Z}^{k\times n}$ $C \in \mathbb{Z}^{m\times n}$ and $\beta \in \mathbb{Z}$ with $a_{i,j}, b_{i,j}, c_{i,j}, \beta \in \{0 \dots p-1\}$. Then every intermediate value z involved in the computation of $A \times B + \beta C$ with l ($l \geq 1$) recursive levels of algorithm 1.4 satisfy:*

$$|z| \leq \left(\frac{1+3^l}{2}\right)^2 \left\lfloor \frac{k}{2^l} \right\rfloor (p-1)^2$$

Moreover, this bound is tight.

For instance, on a single Xeon 2.8GHz core with gcc-4.6.3, Strassen-Winograd's variant implemented with LinBox-1.2.1 and GotoBLAS2-1.13 can be 37% faster for the multiplication of $10\,000 \times 10\,000$ matrices over $\mathbb{F}_{2^{19}-1}$, in less than $1'49"$.

2 Dense Gaussian elimination and echelon forms

In this section, we present algorithms computing the determinant and inverse of square matrices; the rank, rank profile, nullspace, and system solving for arbitrary shape and rank matrices. All these problems are solved a la Gaussian elimination, but recursively in order to effectively incorporate matrix multiplication. The latter is denoted generically `gemm` and, depending on the underlying field, can be implemented using any of the techniques of sections 1.1, 1.2 or 1.3.

A special care is given to the asymptotic time complexities: the exponent is reduced to that of matrix multiplication using block recursive algorithms, and the constants are also carefully compared. Meanwhile, this approach is also effective for implementations: grouping arithmetic operations into matrix-matrix products allow to better optimize cache accesses.

2.1 Building blocks

Algorithms 2.1, 2.2, 2.3 and 2.4 show how to reduce the computation of triangular matrix systems, triangular matrix multiplications, and triangular matrix inversions to matrix-matrix multiplication. Note that they do not require any temporary storage other than the input and output arguments.

2.2 PLE decomposition

Dense Gaussian elimination over finite fields can be reduced to matrix multiplication, using the usual techniques for the LU decomposition of numerical linear algebra [7]. However, in applications over a finite field, the input matrix often has non-generic rank profile and special care needs to be taken about linear dependencies and rank deficiencies. The PLE decomposition is thus a generalization of the PLU decomposition for matrices with any rank profile.

Definition 5 *A matrix is in row-echelon form if all its zero rows occupy the last row positions and the leading coefficient of any non-zero row except the first one is strictly to the right of the leading coefficient of the previous row. Moreover, it is said*

Algorithm 2.1 [`trsm`: Triangular System Solve with Matrix right hand side]

Input: $A \in \mathbb{F}_q^{m \times m}$ non-singular upper triangular, $B \in \mathbb{F}_q^{m \times n}$

Output: $X \in \mathbb{F}_q^{m \times n}$ s.t. $AX = B$

1: **if** m=1 **then return** $X = A_{1,1}^{-1} \times B$ **end if**
2: $X_2 =$`trsm`(A_3, B_2);
3: $B_1 = B_1 - A_2 X_2$; {using gemm, e.g., via alg. 1.5}
4: $X_1 =$`trsm`(A_1, B_1);
5: **return** $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$;

Using the conformal block decomposition:
$\begin{bmatrix} A_1 & A_2 \\ & A_3 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_1 \end{bmatrix}$

Algorithm 2.2 [`trmm`: Triangular Matrix Multiplication]

Input: $A \in \mathbb{F}_q^{m \times m}$ upper triangular, $B \in \mathbb{F}_q^{m \times n}$

Output: $C \in \mathbb{F}_q^{m \times n}$ s.t. $AB = C$

1: **if** m=1 **then return** $C = A_{1,1} \times B$ **end if**
2: $C_1 =$`trmm`(A_1, B_1);
3: $C_1 = C_1 + A_2 B_2$; {using gemm}
4: $C_2 =$`trmm`(A_3, B_2);
5: **return** $C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$;

Using the conformal block decomposition:
$\begin{bmatrix} A_1 & A_2 \\ & A_3 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$

Algorithm 2.3 [`trtri`: Triangular Matrix Inversion]

Input: $A \in \mathbb{F}_q^{n \times n}$ upper triangular and non-singular

Output: $C = A^{-1}$

1: **if** m=1 **then return** $C = A_{1,1}^{-1}$ **end if**
2: $C_1 = A_1^{-1}$; {using trtri recursively}
3: $C_3 = A_3^{-1}$; {using trtri recursively}
4: $C_2 = A_2 C_3$; {using trmm }
5: $C_2 = -C_1 C_2$; {using trmm }
6: **return** $C = \begin{bmatrix} C_1 & C_2 \\ & C_3 \end{bmatrix}$;

Using the conformal block decomposition:
$\begin{bmatrix} A_1 & A_2 \\ & A_3 \end{bmatrix}, \begin{bmatrix} C_1 & C_2 \\ & C_3 \end{bmatrix}$

Algorithm 2.4 [`trtrm`: Upper-Lower Triangular Matrix Multiplication]

Input: $L \in \mathbb{F}_q^{n\times n}$ lower triangular
Input: $U \in \mathbb{F}_q^{n\times n}$ upper triangular
Output: $A = UL$

1: **if** m=1 **then return** $A = U_{1,1}L_{1,1}$ **end if**
2: $A_1 = U_1L_1$; {using trtrm recursively}
3: $A_1 = A_1 + U_2L_2$; {using gemm}
4: $A_2 = U_2L_3$; {using trmm }
5: $A_3 = U_3L_2$; {using trmm }
6: $A_4 = U_3L_3$; {using trtrm recursively}
7: **return** $A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$;

Using the conformal block decomposition:
$\begin{bmatrix} L_1 & \\ L_2 & L_3 \end{bmatrix}, \begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix}, \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$

to be in reduced row-echelon form, if all coefficients above a leading coefficient are zeros.

Definition 6 *For any matrix $A \in F_q^{m\times n}$ of rank r, there is a PLE decomposition $A = PLE$ where P is a permutation matrix, L is a $m \times r$ lower triangular matrix and E is a $r \times n$ matrix in row-echelon form, with unit leading coefficients.*

Algorithm 2.5 shows how to compute such a decomposition by a block recursive algorithm, thus reducing the complexity to that of matrix multiplication.

Algorithm 2.5 [PLE decomposition]

Input: $A \in \mathbb{F}_q^{m\times n}$
Output: (P, L, E) a PLE decomposition of A

1: **if** $n = 1$ **then**
2: **if** $A = 0_{m\times 1}$ **then return** (I_m, I_0, A); **end if**
3: Let j be the column index of the first non zero entry of A and $P = T_{1,j}$ the transposition between indices 1 and j;
4: **return** $(P, PA, [1])$;
5: **else**
6: $(P_1, L_1, E_1) = \texttt{PLE}(A_1)$; {recursively}
7: $A_2 = P_1A_2$;
8: $A_3 = L_{1,1}^{-1}A_3$; {using trsm}
9: $A_4 = A_4 - L_{1,2}A_3$; {using gemm}
10: $(P_2, L_2, E_2) = \texttt{PLE}(A_4)$; {recursively}
11: **return** $(P_1 \begin{bmatrix} I_{r_1} & \\ & P_2 \end{bmatrix}, \begin{bmatrix} L_{1,1} & \\ P_2L_{1,2} & L_2 \end{bmatrix}, \begin{bmatrix} E_1 & A_3 \\ & E_2 \end{bmatrix})$;
12: **end if**

Split A columnwise in halves: $A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$
Split $A_2 = \begin{bmatrix} A_3 \\ A_4 \end{bmatrix}$, $L_1 = \begin{bmatrix} L_{1,1} \\ L_{1,2} \end{bmatrix}$ where A_3 and $L_{1,1}$ have r_1 rows.

2.3 Echelon forms

The row-echelon and reduced row-echelon forms can be obtained from the PLE decomposition, using additional operations: `trsm`, `trtri` and `trtrm`, as shown in algorithm 2.6 and 2.7.

Algorithm 2.6 [`RowEchelon`]

Input: $A \in \mathbb{F}_q^{m \times n}$
Output: (X, E) such that $XA = E$, X is non-singular and E is in row-echelon form

1: $(P, L, E) = \texttt{PLE}(A)$;
2: $X_1 = L_1^{-1}$; {using trtri}
3: $X_2 = -L_2 X_1$; {using trmm}
4: **return** $\left(X = \begin{bmatrix} X_1 & \\ X_2 & I_{m-r} \end{bmatrix} P^T, E \right)$;

Split $L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$, $L_1 : r \times r$.

Algorithm 2.7 [`ReducedRowEchelon`]

Input: $A \in \mathbb{F}_q^{m \times n}$
Output: (Y, R) such that $YA = R$, Y is non-singular and R is in reduced row-echelon form

1: $(X, E) = \texttt{RowEchelon}(A)$;
2: Let Q be the permutation matrix that brings the leading row coefficients of E to the diagonal;
3: Set $EQ = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$; {where U_1 is $r \times r$ upper triangular}
4: $Y_1 = U_1^{-1}$; {using trtri}
5: $Y_1 = Y_1 X_1$; {using trtrm}
6: $R = \begin{bmatrix} I_r & U_1^{-1} U_2 \end{bmatrix} Q^T$; {using trsm}
7: **return** $\left(Y = \begin{bmatrix} Y_1 & \\ U_2 & I_{n-r} \end{bmatrix} P^T, R \right)$;

Figure 1 shows the various steps between the classical Gaussian elimination (LU decomposition), the computation of the echelon form and of the reduced echelon form, together with the various problems that each of them solve. Table 2 shows the leading constant K_ω in the asymptotic time complexity of these algorithms, assuming that two $n \times n$ matrices can be multiplied in $C_\omega n^\omega + o(n^\omega)$.

Remark 7 *Note that, if the rank r is very small compared to the dimensions $m \times n$ of the matrix, a system $Ax = b$ can be solved in time bounded by $\mathcal{O}\left((m+n)r^2\right)$ [45, Theorem 1].*

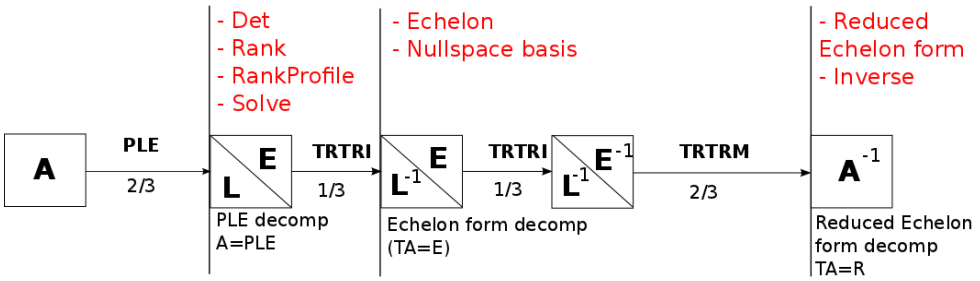

Figure 1: Reductions from PLE decomposition to Reduced echelon form

Algorithm	Constant K_ω	K_3	$K_{\log_2 7}$
`gemm`	C_ω	2	6
`trsm`	$\frac{C_\omega}{2^{\omega-1}-2}$	1	4
`trtri`	$\frac{C_\omega}{(2^{\omega-1}-2)(2^{\omega-1}-1)}$	$\frac{1}{3} \approx 0.33$	$\frac{8}{5} = 1.6$
`trtrm`, PLE	$\frac{C_\omega}{2^{\omega-1}-2} - \frac{C_\omega}{2^\omega-2}$	$\frac{2}{3} \approx 0.66$	$\frac{14}{5} = 2.8$
`Echelon`	$\frac{C_\omega}{2^{\omega-2}-1} - \frac{3C_\omega}{2^\omega-2}$	1	$\frac{22}{5} \approx 4.4$
`RedEchelon`	$\frac{C_\omega(2^{\omega-1}+2)}{(2^{\omega-1}-2)(2^{\omega-1}-1)}$	2	$\frac{44}{5} = 8.8$

Table 2: Complexity of elimination algorithms

3 Minimal and characteristic polynomial of a dense matrix

Definition 8 *1. A* Las-Vegas algorithm *is a randomized algorithm which is always correct. Its expected running is time is always finite.*

2. A Monte-Carlo algorithm *is a randomized algorithm which is correct with a certain probability. Its running time is deterministic.*

The computation of the minimal and characteristic polynomials is closely related to that of the Frobenius normal form.

Definition 9 *Any matrix $A \in \mathbb{F}_q^{n\times n}$ is similar to a unique block diagonal matrix $F = P^{-1}AP = diag(C_{f_1}, \ldots, C_{f_t})$ where the blocks C_{f_i} are companion matrices of the polynomials f_i, which satisfy $f_{i+1}|f_i$. The f_i are the* invariant factors *of A and F is the* Frobenius normal form *of A.*

Most algorithms computing the minimal and characteristic polynomial or the Frobenius normal form rely on Krylov basis computations.

Definition 10 *1. The Krylov matrix of order d for a vector v w.r.t a matrix A is the matrix $K_{A,v,d} = \begin{bmatrix} v & Av & \ldots & A^{d-1}v \end{bmatrix} \in \mathbb{F}_q^{n\times d}$.*

2. *The minimal polynomial* $P_{min}^{A,v}$ *of* A *and* v *is the least degree monic polynomial* P *such that* $P(A)v = 0$.

Theorem 11 1. $AK_{A,v,d} = K_{A,v,d}C_{P_{min}^{A,v}}$, *where* $d = \deg(P_{min}^{A,v})$.

2. *For lineraly independent vectors* $(v_1, \ldots, v_k)$, *if* $K = \begin{bmatrix} K_{A,v_1,d_1} & \ldots & K_{A,v_k,d_k} \end{bmatrix}$ *is non singular. Then* $AK = K \begin{bmatrix} C_{P_{min}^{A,v_1}} & B_{1,2} & \ldots & B_{1,k} \\ B_{2,1} & C_{P_{min}^{A,v_1}} & \ldots & B_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ B_{k,1} & B_{k,2} & & C_{P_{min}^{A,v_k}} \end{bmatrix}$, *where the blocks* $B_{i,j}$ *are zero except on the last column.*

3. *For linearly independent vectors* $(v_1, \ldots, v_k)$, *let* $(d_1, \ldots d_k)$ *be the lexicographically largest sequence of degrees such that* $K = \begin{bmatrix} K_{A,v_1,d_1} & \ldots & K_{A,v_k,d_k} \end{bmatrix}$ *is non-singular. Then*

$$K^{-1}AK = \begin{bmatrix} C_{P_{min}^{A,v_1}} & B_{1,2} & \ldots & B_{1,k} \\ & C_{P_{min}^{A,v_1}} & \ldots & B_{2,k} \\ & & \ddots & \vdots \\ & & & C_{P_{min}^{A,v_k}} \end{bmatrix} = H \tag{1}$$

Remark 12 1. *Some choice of vectors* $v_1, \ldots, v_k$ *lead to a matrix* H *block diagonal: this is the Frobenius normal form [27]*

2. *The matrix obtained at equation (1) is called a Hessenberg form. It suffices to compute the characteristic polynomial from its diagonal blocks.*

Theorem 13 *The Frobenius normal form can be computed:*

1. *by a deterministic algorithm [49] in* $6n^3 + \mathcal{O}\left(n^2 \log^2 n\right)$ *field operations,(only* $(2 + \frac{2}{3})n^3 + \mathcal{O}\left(n^2\right)$ *for the characteristic polynomial [19])*

2. *by a deterministic algorithm [48] in* $\mathcal{O}\left(n^\omega \log n \log\log n\right)$, *together with a transformation matrix, (only* $\mathcal{O}\left(n^\omega \log n\right)$ *for the characteristic polynomial [40])*

3. *by a Las-Vegas algorithm [22] in* $\mathcal{O}\left(n^\omega \log n\right)$ *field operations for any field, together with a transformation matrix*

4. *by a Las-Vegas algorithm [47] in* $\mathcal{O}\left(n^\omega\right)$ *for* $q > 2n^2$, *without transformation matrix.*

The minimal and characteristic polynomials, obtained as the first invariant factor and the product of all invariant factors, can be computed with the same complexities.

Remark 14 *These algorithms are all based Krylov bases. Algorithm (1.) iteratively compute the Krylov iterates one after the other. Their cubic time complexity with a small leading constant makes them comparable to Gaussian elimination. A fast*

exponentiation scheme by Keller-Gehrig [40] achieves a sub-cubic time complexity for the characteristic polynomial, off by a logarithmic factor of n from the matrix multiplication. The choice for the appropriate vectors that will generate the Frobenius normal form can be done either probabilistically (Las-Vegas) or deterministically with an $\log\log n$ factor. Algorithm (4.) uses a different iteration where the size of the Krylov increases according to an arithmetic progression rather than geometric (as all others) and the transformation matrix is not computed. This allows it to match to the complexity of matrix multiplication. This reduction is practical and is implemented as in LinBox.

Remark 15 *These probabilistic algorithms depend on the ability to sample uniformly from a large set of coefficients from the field. Over small fields, it is always possible to embed the problem into an extension field, in order to make the random sampling set sufficiently large. In the worst case, this could add a $\mathcal{O}(\log(n))$ factor to the arithmetic cost and prevent most of the bit-packing techniques. Instead, the effort of [22] is to handle cleanly the small finite field case.*

4 Blackbox iterative methods

We consider now the case where the input matrix is sparse, i.e., has many zero elements, or has a structure which enables fast matrix-vector products. Gaussian elimination would fill-in the sparse matrix or modify the interesting structure. Therefore one can use iterative methods instead which only use matrix-vector iterations (*blackbox methods* [38]). There are two major differences with numerical iterative routines: over finite fields there exists isotropic vectors and there is no notion of convergence, hence the iteration must proceed until exactness of the result [42]. Probabilistic early termination can nonetheless be applied when the degree of the minimal polynomial is smaller than the dimension of the matrix [34, 20, 23]. More generally the probabilistic nature of the algorithms presented in this section is subtle: e.g., the computation of the minimal polynomial is Monte-Carlo, but that of system solving, using the minimal polynomial, is Las-Vegas (by checking consistency of the produced solution with the system). Making some of the Monte-Carlo solutions Las-Vegas is a key open-problem in this area.

4.1 Minimal Polynomial and the Wiedemann algorithm

The first iterative algorithm and its analysis are due to D. Wiedemann [58]. The algorithm computes the minimal polynomial in the Monte-Carlo probabilistic fashion.

Definition 16 *For a linearly recurring sequence $S = (S_i)$, its minimal polynomial is denoted by Π_S.*

- *The minimal polynomial of a matrix is denoted $\Pi_A = \Pi_{(A^i)}$.*
- *For a matrix A and a vector b, we note $\Pi_{A,b} = \Pi_{(A^i \cdot b)}$.*

- *With another vector u, we note $\Pi_{u,A,b} = \Pi_{(u^T \cdot A^i \cdot b)}$.*

Algorithm 4.1 [Wiedemann minimal polynomial]

Input: $A \in \mathbb{F}_q^{n \times n}$, $u, b \in \mathbb{F}_q^n$.
Output: $\Pi_{u,A,b}$.
1: Compute $S = (u^T A^i b)$ for $i \leq 2n$;
2: Use the Berlekamp-Massey algorithm to compute the minimal polynomial of the scalar sequence S;

Definition 17 *We extend Euler's totient function by $\Phi_{q,k}(f) = \prod(1-q^{-kd_i})$, where d_i are the degrees of the distinct monic irreducible factors of the polynomial f.*

Theorem 18 *For vectors $u_1, \ldots, u_j$ selected uniformly at random, the probability that $\mathrm{lcm}(\Pi_{u_j,A,b}) = \Pi_{A,b}$ is at least $\Phi_{q,k}(\Pi_{A,b})$.*

Theorem 19 *For vectors $b_1, \ldots, b_k$ selected uniformly at random, the probability that $\mathrm{lcm}(\Pi_{A,b_i}) = \Pi_A$ is at least $\Phi_{q,k}(\Pi_A)$.*

4.2 Rank, Determinant and Characteristic Polynomial

It is possible to compute the rank, determinant, and characteristic polynomial of a matrix from its minimal polynomial. All these reductions require to precondition the matrix so that the minimal polynomial of the obtained matrix will reveal the information sought, while keeping a low cost for the matrix-vector product [25, 37, 20, 52, 55, 56, 9].

Theorem 20 *[25] Let S be a finite subset of a field $\mathbb{F}$ that does not include 0. Let $A \in \mathbb{F}^{m \times n}$ having rank r. Let $D_1 \in S^{n \times n}$ and $D_2 \in S^{m \times m}$ be two random diagonal matrices then $degree(minpoly(D_1 \times A^t \times D_2 \times A \times D_1)) = r$, with probability at least $1 - \frac{11.n^2 - n}{2|S|}$.*

Theorem 21 *[52] Let S be a finite subset of a field $\mathbb{F}$ that does not include 0. Let $U \in S^{n \times n}$ be a unit upper bi-diagonal matrix where the second diagonal elements $u_1, \ldots, u_{n-1}$ are randomly selected in S. For $A \in \mathbb{F}^{n \times n}$, the term of degree 0 of the minimal polynomial of UA is the determinant of A with probability at least $1 - \frac{n^2 - n}{2|S|}$.*

Remark 22 *If A is known to be non-singular the algorithm can be repeated with different matrices U until the obtained minimal polynomial is of degree n. Then it is the characteristic polynomial of UA and the determinant is certified. Alternatively if the matrix is singular then X divides the minimal polynomial. As Wiedemann's algorithm always returns a factor of the true minimal polynomial, and U is invertible, the algorithm can be repeated on UA until either the obtained polynomial is of degree n or it is divisible by X. Overall the determinant has a Las-Vegas blackbox solution.*

Theorem 23 *[55, 56] Let S be a finite subset of a field $\mathbb{F}$ that does not include 0 and $A \in \mathbb{F}^{n\times n}$ with $s_1, \ldots, s_t$ as invariant factors. Let $U \in S^{n\times k}$ and $V \in S^{k\times n}$ be randomly chosen rank k matrices in $\mathbb{F}$. Then $\gcd(\Pi_A, \Pi_{A+UV}) = s_{k+1}$ with probability at least $1 - \frac{nk+n+1}{|S|}$.*

Remark 24 *Using the divisibility of the invariant factors and the fact that their product is of degree n, one can see that the number of degree changes between successive invariant factors is of order $\mathcal{O}(\sqrt{n})$ [55]. Thus by a binary search over successive applications of theorem 23 one can recover all of the invariant factors and thus the characteristic polynomial of the matrix in a Monte-Carlo fashion.*

4.3 System solving and the Lanczos algorithm

For the solution of a linear system $Ax = b$, one could compute the minimal polynomial $\Pi_{A,b}$ and then derive a solution of the system as a linear combination of the $A^i b$. The following Lanczos approach is more efficient for system solving as it avoids recomputing (or storing) the latter vectors [25, 28].

Algorithm 4.2 [Lanczos system solving]

Input: $A \in \mathbb{F}^{m\times n}$, $b \in \mathbb{F}^m$.
Output: $x \in \mathbb{F}^n$ such that $Ax = b$ or *failure*.

1: Let $\tilde{A} = D_1 A^T D_2 A D_1$ and $\tilde{b} = D_1 A^T D_2 b + \tilde{A}v$ with D_1 and D_2 random diagonal matrices and v a random vector;
2: $w_0 = \tilde{b}$; $v_1 = \tilde{A}w_0$; $t_0 = v_1^T w_0$; $\gamma = \tilde{b}^t w_0 t_0^{-1}$; $x_0 = \gamma w_0$;
3: **repeat**
4: $\quad \alpha = v_{i+1}^T v_{i+1} t_i^{-1}$; $\beta = v_{i+1}^T v_i t_{i-1}^{-1}$; $w_{i+1} = v_{i+1} - \alpha w_i - \beta w_{i-1}$;
5: $\quad v_{i+2} = \tilde{A}w_{i+1}$; $t_{i+1} = w_{i+1}^T v_{i+2}$;
6: $\quad \gamma = \tilde{b}^t w_{i+1} t_{i+1}^{-1}$; $x_{i+1} = x_i + \gamma w_{i+1}$;
7: **until** $w_{i+1} = 0$ or $t_{i+1} = 0$;
8: Return $x = D_1(x_{i+1} - v)$;

The probability of success of algorithm 4.2 follows also theorem 20.

Remark 25 *Over small fields, if the rank of the matrix is known, the diagonal matrices of line 1 can be replaced by sparse preconditioners with $\mathcal{O}(n\log(n))$ nonzero coefficients to avoid the need of field extensions[9, corollary 7.3].*

Remark 26 *If the system with A and b is known to have a solution then the algorithm can be turned Las-Vegas by checking that the output x indeed satisfies $Ax = b$. In general, we do not know if this algorithm returns failure because of bad random choices or because the system is inconsistent. However, Giesbrecht, Lobo and Saunders have shown that when the system is inconsistent, it is possible to produce a certificate vector u such that $u^T A = 0$ together with $u^T b \neq 0$ within the same complexity [28, Theorem 2.4]. Overall, system solving can be performed by blackbox algorithms in a Las-Vegas fashion.*

5 Sparse and structured methods

Another approach to sparse linear system is to use Gaussian elimination with pivoting, taking into account the zero coefficients. This algorithm modifies the structure of the matrix and might suffer from fill-in. Consequently the available memory is usually the bottleneck. From a triangularization one can naturally derive the rank, determinant, system solving and nullspace. Comparisons with the blackbox approaches above can be found e.g., in [20].

5.1 Reordering

Algorithm 5.1 [Gaussian elimination with linear pivoting]

Input: a matrix $A \in \mathbb{F}^{m \times n}$;

Output: An upper triangular matrix U such that there exists a unitary lower-triangular matrix L and permutations matrices P and Q over $\mathbb{F}$, with $A = P \cdot L \cdot U \cdot Q$;

1: **for all** elimination steps **do**
2: Choose as pivot row the sparsest remaining row;
3: In this row choose the non zero pivot with lowest number of non zero elements in its column;
4: Eliminate using this pivot;
5: **end for**

Remark 27 *Yannakakis showed that finding the minimal fill-in (or equivalently the best pivots) during Gaussian elimination is an NP-complete task [60]. In numerical algorithms, heuristics have been developed and comprise minimal degree ordering, cost functions or nested dissection (see e.g., [61, 1, 31]). These heuristics for reducing fill-in in the numerical setting, often assume symmetric and invertible matrices, and do not take into account that new zeros may be produced by elimination operations ($a_{ij} = a_{ij} + \delta_i * a_{kj}$), as is the case with matrices over finite fields. [20] thus proposed the heuristic 5.1 to take those new zeros into account, using a local optimization of a cost function at each elimination step.*

5.2 Structured matrices and displacement rank

Originating from the seminal paper [33] most of the algorithms dealing with structured matrices use the displacement rank approach [46].

Definition 28 *For $A \in \mathbb{F}^{m \times m}$ and $B \in \mathbb{F}^{n \times n}$, the* Sylvester (resp. Stein) linear displacement operator $\bigtriangledown_{A,B}$ (resp. $\bigtriangleup_{A,B}$) *satisfy for $M \in \mathbb{F}^{m \times n}$:*

$$\bigtriangledown_{A,B}(M) = AM - MB$$
$$\bigtriangleup_{A,B}(M) = M - AMB$$

A pair of matrices $(Y, Z) \in \mathbb{F}^{m \times \alpha} \times \mathbb{F}^{n \times \alpha}$ is a A, B-Sylvester-generator of length α *(resp. Stein) for M if $\bigtriangledown_{A,B}(M) = YZ^T$ (resp. $\bigtriangleup_{A,B}(M) = YZ^T$).*

The main idea behind algorithms for structured matrices is to use such generators as a compact data structure, in cases where the displacement has low rank.

Usual choices of matrices A and B are diagonal matrices and cyclic down shift matrices:

Definition 29 *$\mathbb{D}_x, x \in \mathbb{F}^n$ is the diagonal matrix whose (i,i) entry is x_i.*
$\mathbb{Z}_{n,\varphi}, \varphi \in \mathbb{F}$ is the $n \times n$ unit circulant matrix having φ at position $(1,n)$, ones in the subdiagonal $(i+1,i)$ and zeros elsewhere.

operator matrices A	operator matrices B	class of structured matrices M	rank of $\nabla_{A,B}(M)$	number of flops for computing $M \cdot v$
$\mathbb{Z}_{n,1}$	$\mathbb{Z}_{n,0}$	Toeplitz and its inverse	≤ 2	$\mathcal{O}\left((m+n)\log(m+n)\right)$
$\mathbb{Z}_{n,1}$	$\mathbb{Z}_{n,0}^T$	Hankel and its inverse	≤ 2	$\mathcal{O}\left((m+n)\log(m+n)\right)$
$\mathbb{Z}_{n,0} + \mathbb{Z}_{n,0}^T$	$\mathbb{Z}_{n,0} + \mathbb{Z}_{n,0}^T$	Toeplitz+Hankel	≤ 4	$\mathcal{O}\left((m+n)\log(m+n)\right)$
$\mathbb{D}_x$	$\mathbb{Z}_{n,0}$	Vandermonde	≤ 1	$\mathcal{O}\left((m+n)\log^2(m+n)\right)$
$\mathbb{Z}_{n,0}$	$\mathbb{D}_x$	inverse of Vandermonde	≤ 1	$\mathcal{O}\left((m+n)\log^2(m+n)\right)$
$\mathbb{Z}_{n,0}^T$	$\mathbb{D}_x$	transposed of Vandermonde	≤ 1	$\mathcal{O}\left((m+n)\log^2(m+n)\right)$
$\mathbb{D}_x$	$\mathbb{D}_y$	Cauchy and its inverse	≤ 1	$\mathcal{O}\left((m+n)\log^2(m+n)\right)$

Table 3: Complexity of the matrix-vector product for some structured matrices

As computing matrix vector products with such structured matrices have close algorithmic correlation to computations with polynomials and rational functions, these matrices can be multiplied by vectors fast, in nearly linear time as shown on table 3. Therefore the algorithms of section 4 can naturally be applied to structured matrices, to yield almost $\mathcal{O}\left(n^2\right)$ time linear algebra.

Now, if the displacement rank is small there exists algorithms quasi linear in n, the dimension of the matrices, which over finite fields are essentially variations or extensions of the Morf/Bitmead-Anderson divide-and-conquer [44, 4] or Cardinal's [8] approaches. The method is based on dividing the original problem repeatedly into two subproblems with one leading principal submatrix and the related Schur complement. This leads to $\mathcal{O}\left(\alpha^2 n^{1+o(1)}\right)$ system solvers, which complexity bound have recently been reduced to $\mathcal{O}\left(\alpha^{\omega-1} n^{1+o(1)}\right)$ [6, 32]. We few exceptions, all algorithms thus need matrices in generic rank profile. Over finite fields this can be achieved using Kaltofen and Saunders unit upper triangular Toeplitz preconditioners [37] and by controlling the displacement rank growth and non-singularity issues [35].

6 Hybrid methods

6.1 Hybrid sparse-dense methods

Overall, as long as the matrix fits into memory, Gaussian elimination methods are usually faster than iterative methods, over finite fields [20]. There are then heuristics trying to take the best of both strategies. Among those we briefly mention the most widely used:

- Perform the Gaussian elimination with reordering 5.1 until the matrix is almost filled-up. If the remaining non-eliminated part would fit as a dense matrix, switch to the dense methods of section 2.
- Maintain two sets of rows (or columns), sparse and dense. Favor elimination on the sparse set. This is particularly adapted to index calculus [41].
- Perform a preliminary reordering in order to cut the matrix into four quadrants, the upper left one being triangular. This, together with the above strategies has proven effective on matrices which are already quasi-triangular, e.g., Gröbner bases computations in finite fields [26].
- If the rank is very small compared to the dimension of the matrix, one can use left and right highly rectangular projections to manipulate smaller structures [43].
- The arithmetic cost and thus timing predictions are easier on iterative methods than on elimination methods. On the other hand the number of non-zero elements at a given point of the elimination is usually increasing during an elimination, thus providing a lower bound on the remaining time to triangularize. Thus a heuristic is to perform one matrix-vector product with the original matrix and then eliminate using Gaussian elimination. If at one point the lower bound for elimination time surpasses to predicted iterative one or if the the algorithm runs out of memory, stop the elimination and switch to the iterative methods [21].

6.2 Block-iterative methods

Iterative methods based on one-dimensional projections, such as Wiedmann and Lanczos algorithm can be generalized with block projections. Via efficient preconditioning [9] these extensions to the scalar iterative methods can present enhanced properties:

- Usage of dense sub-blocks, after multiplications of blocks of vectors with the sparse matrix or the blackboxes, allows for a better locality and optimization of memory accesses, via the application of the methods of section 1.
- Applying the matrix to several vectors simultaneously introduces more parallelism [10, 11, 34].
- Also, their probability of success augments with the size of the considered blocks, especially over small fields [36, 54].

Definition 30 *Let $X \in \mathbb{F}_q^{k\times n}$, $Y \in \mathbb{F}_q^{n\times k}$ and $H_i = XA^iY$ for $i = 0 \dots n/k$. The matrix minimal polynomial of the sequence H_i is the matrix polynomial $F_{X,A,Y} \in \mathbb{F}_q[X]^{k\times k}$ of least degree, with its leading degree matrix column-reduced, that annihilates the sequence (H_i).*

Theorem 31 *The degree d matrix minimal polynomial of a block sequence $(H_i) \in (F_q^{k\times k})^{\mathbb{Z}}$ can be computed in $\mathcal{O}\left(k^3 d^2\right)$ using block versions of Hermite-Pade approximation and extended Euclidean algorithm [3] or Berlkamp-Massey algorithm [11, 36, 54]. Further improvement by [3, 51, 29, 39] bring this complexity down to $\mathcal{O}\left(k^\omega d\right)^{1+o(1)}$, using a matrix extended Euclidean algorithm.*

Algorithm 6.1 [Nullspace vector]

Input: $A \in \mathbb{F}_q^{n\times n}$
Output: $\omega \in \mathbb{F}_q^n$ a vector in the nullspace of A
Pick $X \in \mathbb{F}_q^{k\times n}, Y \in \mathbb{F}_q^{n\times k}$ uniformly at random;
Compute the sequence $H_i = XA^iY$;
Compute $F_{X,A,Y}$ the matrix minimal polynomial;
Let $f = f_r x^r + \cdots + f_d x^d$ be a column of $F_{X,A,Y}$;
Return $\omega = Yf_r + AYf_{r+1} + \cdots + A^{d-r}Yf_d$;

Remark 32 *These block-Krylov techniques are used to achieve the best known time complexities for several computations with black-box matrices over a finite field or the ring of integers: computing the determinant, the characteristic polynomial [39] and the solution of a linear system of equations [24].*

7 Acknowledgment

We thank an anonymous referee for numerous helpful suggestions that considerably improved the paper.

8 References

[1] Amestoy, Patrick R., Timothy A. Davis, and Iain S. Duff: *Algorithm 837: AMD, an approximate minimum degree ordering algorithm.* ACM Trans. Math. Software, 30(3):381–388, 2004, ISSN 0098-3500.

[2] Arlazarov, V. L., E. A. Dinic, M. A. Kronrod, and I. A. Faradžev: *The economical construction of the transitive closure of an oriented graph.* Dokl. Akad. Nauk SSSR, 194:487–488, 1970, ISSN 0002-3264.

[3] Beckermann, Bernhard and George Labahn: *Fraction-free computation of matrix rational interpolants and matrix GCDs.* SIAM J. Matrix Anal. Appl., 22(1):114–144 (electronic), 2000, ISSN 0895-4798.

[4] Bitmead, Robert R. and Brian D. O. Anderson: *Asymptotically fast solution of Toeplitz and related systems of linear equations.* Linear Algebra Appl., 34:103–116, 1980, ISSN 0024-3795.

[5] Boothby, Thomas J. and Robert W. Bradshaw: *Bitslicing and the method of four russians over larger finite fields*, January 2009. arXiv:0901.1413v1 [cs.MS].

[6] Bostan, Alin, Claude Pierre Jeannerod, and Éric Schost: *Solving structured linear systems with large displacement rank*. Theoret. Comput. Sci., 407(1-3):155–181, 2008, ISSN 0304-3975.

[7] Bunch, James R. and John E. Hopcroft: *Triangular factorization and inversion by fast matrix multiplication*. Math. Comp., 28:231–236, 1974, ISSN 0025-5718.

[8] Cardinal, Jean Paul: *On a property of Cauchy-like matrices*. C. R. Acad. Sci. Paris Sér. I Math., 328(11):1089–1093, 1999, ISSN 0764-4442.

[9] Chen, Li, Wayne Eberly, Erich Kaltofen, B. David Saunders, William J. Turner, and Gilles Villard: *Efficient matrix preconditioners for black box linear algebra*. Linear Algebra Appl., 343/344:119–146, 2002, ISSN 0024-3795. Special issue on structured and infinite systems of linear equations.

[10] Coppersmith, Don: *Solving linear equations over* GF(2)*: block Lanczos algorithm*. Linear Algebra Appl., 192:33–60, 1993, ISSN 0024-3795. Computational linear algebra in algebraic and related problems (Essen, 1992).

[11] Coppersmith, Don: *Solving homogeneous linear equations over* GF(2) *via block Wiedemann algorithm*. Math. Comp., 62(205):333–350, 1994, ISSN 0025-5718.

[12] Coppersmith, Don: *Rectangular matrix multiplication revisited*. J. Complexity, 13(1):42–49, 1997, ISSN 0885-064X. http://dx.doi.org/10.1006/jcom.1997.0438.

[13] Coppersmith, Don and Shmuel Winograd: *Matrix multiplication via arithmetic progressions*. J. Symbolic Comput., 9(3):251–280, 1990, ISSN 0747-7171.

[14] Dumas, Jean Guillaume: *Q-adic transform revisited*. In *Proceedings of the 2008 International Symposium on Symbolic and Algebraic Computation*, pages 63–69, New York, 2008. ACM. http://hal.archives-ouvertes.fr/hal-00173894.

[15] Dumas, Jean Guillaume, Laurent Fousse, and Bruno Salvy: *Simultaneous modular reduction and Kronecker substitution for small finite fields*. Journal of Symbolic Computation, 46(7):823 – 840, 2011, ISSN 0747-7171. http://hal.archives-ouvertes.fr/hal-00315772, Special Issue in Honour of Keith Geddes on his 60th Birthday.

[16] Dumas, Jean Guillaume, Thierry Gautier, Mark Giesbrecht, Pascal Giorgi, Bradford Hovinen, Erich Kaltofen, B. David Saunders, Will J. Turner, and Gilles Villard: *LinBox: A generic library for exact linear algebra*. In Cohen, Arjeh M., Xiao Shan Gao, and Nobuki Takayama (editors): *ICMS'2002, Proceedings of the 2002 International Congress of Mathematical Software, Beijing, China*, pages 40–50. World Scientific Pub., August 2002. http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Publications/icms.pdf.

[17] Dumas, Jean Guillaume, Thierry Gautier, and Clément Pernet: *Finite field linear algebra subroutines.* In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 63–74, New York, 2002. ACM. http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Publications/Field_blas.pdf.

[18] Dumas, Jean Guillaume, Pascal Giorgi, and Clément Pernet: *Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages.* ACM Trans. Math. Software, 35(3):Art. 19, 35, 2008, ISSN 0098-3500. http://hal.archives-ouvertes.fr/hal-00018223.

[19] Dumas, Jean Guillaume, Clément Pernet, and Zhendong Wan: *Efficient computation of the characteristic polynomial.* In *ISSAC'05*, pages 140–147 (electronic). ACM, New York, 2005. http://dx.doi.org/10.1145/1073884.1073905.

[20] Dumas, Jean Guillaume and Gilles Villard: *Computing the rank of sparse matrices over finite fields.* In Ganzha, Victor G., Ernst W. Mayr, and Evgenii V. Vorozhtsov (editors): *CASC 2002, Proceedings of the fifth International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*, pages 47–62. Technische Universität München, Germany, September 2002. http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Publications/sparseeliminationCASC2002.pdf.

[21] Duran, Ahmet, B. David Saunders, and Zhendong Wan: *Hybrid algorithms for rank of sparse matrices.* In Mathias, Roy and Hugo Woerdeman (editors): *SIAM Conference on Applied Linear Algebra, Williamsburg, VA, USA*, July 2003.

[22] Eberly, Wayne: *Black box Frobenius decompositions over small fields (extended abstract).* In *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (St. Andrews)*, pages 106–113 (electronic), New York, 2000. ACM. http://dx.doi.org/10.1145/345542.345596.

[23] Eberly, Wayne: *Early termination over small fields.* In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, ISSAC '03, pages 80–87, New York, NY, USA, 2003. ACM, ISBN 1-58113-641-2.

[24] Eberly, Wayne, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann, and Gilles Villard: *Faster inversion and other black box matrix computations using efficient block projections.* In *ISSAC 2007*, pages 143–150. ACM, New York, 2007.

[25] Eberly, Wayne and Erich Kaltofen: *On randomized Lanczos algorithms.* In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (Kihei, HI)*, pages 176–183 (electronic), New York, 1997. ACM.

[26] Faugère, Jean Charles and Sylvain Lachartre: *Parallel gaussian elimination for gröbner bases computations in finite fields.* In Maza, Marc Moreno and Jean Louis Roch (editors): *PASCO 2010, Proceedings of the 4th International Workshop on Parallel Symbolic Computation, Grenoble, France*, pages 89–97. ACM, July 2010.

[27] Gantmacher, F. R.: *The theory of matrices. Vol. 1.* AMS Chelsea Publishing, Providence, RI, 1998, ISBN 0-8218-1376-5. Translated from the Russian by K. A. Hirsch, Reprint of the 1959 translation.

[28] Giesbrecht, M., A. Lobo, and B. D. Saunders: *Certifying inconsistency of sparse linear systems.* In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation (Rostock)*, pages 113–119, New York, 1998. ACM. http://dx.doi.org/10.1145/281508.281591.

[29] Giorgi, Pascal, Claude Pierre Jeannerod, and Gilles Villard: *On the complexity of polynomial matrix computations.* In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 135–142 (electronic), New York, 2003. ACM.

[30] Goto, Kazushige and Robert van de Geijn: *High-performance implementation of the level-3 BLAS.* ACM Trans. Math. Software, 35(1):Art. 4, 14, 2009, ISSN 0098-3500.

[31] Hendrickson, Bruce and Edward Rothberg: *Improving the run time and quality of nested dissection ordering.* SIAM J. Sci. Comput., 20(2):468–489 (electronic), 1998, ISSN 1064-8275.

[32] Jeannerod, Claude Pierre and Christophe Mouilleron: *Computing specified generators of structured matrix inverses.* In Koepf, Wolfram (editor): *Symbolic and Algebraic Computation, International Symposium, ISSAC 2010, Munich, Germany, July 25-28, 2010, Proceedings*, pages 281–288. ACM, 2010.

[33] Kailath, T., S. Y. Kung, and M. Morf: *Displacement ranks of a matrix.* Bull. Amer. Math. Soc. (N.S.), 1(5):769–773, 1979, ISSN 0273-0979.

[34] Kaltofen, E. and A. Lobo: *Distributed matrix-free solution of large sparse linear systems over finite fields.* Algorithmica, 24(3-4):331–348, 1999, ISSN 0178-4617. http://www.math.ncsu.edu/~kaltofen/bibliography/99/KaLo99.pdf.

[35] Kaltofen, Erich: *Asymptotically fast solution of Toeplitz-like singular linear systems.* In *Proceedings of the international symposium on Symbolic and algebraic computation*, ISSAC '94, pages 297–304, New York, NY, USA, 1994. ACM. http://www.math.ncsu.edu/~kaltofen/bibliography/94/Ka94_issac.pdf.

[36] Kaltofen, Erich: *Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems.* Math. Comp., 64(210):777–806, 1995, ISSN 0025-5718.

[37] Kaltofen, Erich and B. David Saunders: *On Wiedemann's method of solving sparse linear systems.* In *Applied algebra, algebraic algorithms and error-correcting codes (New Orleans, LA, 1991)*, volume 539 of *Lecture Notes in Comput. Sci.*, pages 29–38. Springer, Berlin, 1991.

[38] Kaltofen, Erich and Barry M. Trager: *Computing with polynomials given by black boxes for their evaluations: greatest common divisors, factorization, separation of numerators and denominators.* J. Symbolic Comput., 9(3):301–320, 1990, ISSN 0747-7171. http://www.math.ncsu.edu/~kaltofen/bibliography/90/KaTr90.pdf.

[39] Kaltofen, Erich and Gilles Villard: *On the complexity of computing determinants.* Comput. Complexity, 13(3-4):91–130, 2004, ISSN 1016-3328.

[40] Keller-Gehrig, Walter: *Fast algorithms for the characteristic polynomial.* Theoret. Comput. Sci., 36(2-3):309–317, 1985, ISSN 0304-3975. http://dx.doi.org/10.1016/0304-3975(85)90049-0.

[41] LaMacchia, Brian A. and Andrew M. Odlyzko: *Solving large sparse linear systems over finite fields.* Lecture Notes in Computer Science, 537:109–133, 1991. http://www.dtc.umn.edu/~odlyzko/doc/arch/sparse.linear.eqs.pdf.

[42] Lambert, Rob: *Computational aspects of discrete logarithms.* PhD thesis, University of Waterloo, Ontario, Canada, 1996. http://www.cacr.math.uwaterloo.ca/techreports/2000/lambert-thesis.ps.

[43] May, John P., David Saunders, and Zhendong Wan: *Efficient matrix rank computation with application to the study of strongly regular graphs.* In *ISSAC 2007*, pages 277–284. ACM, New York, 2007.

[44] Morf, M.: *Doubling algorithms for teoplitz and related equations.* In *Proc. 1980 Int'l Conf. Acoustics Speech and Signal Processing*, pages 954–959, Denver, Colo., April 1980.

[45] Mulders, Thom and Arne Storjohann: *Rational solutions of singular linear systems.* In *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (St. Andrews)*, pages 242–249 (electronic), New York, 2000. ACM. http://dx.doi.org/10.1145/345542.345644.

[46] Pan, Victor Y.: *Structured matrices and polynomials.* Birkhäuser Boston Inc., Boston, MA, 2001, ISBN 0-8176-4240-4. Unified superfast algorithms.

[47] Pernet, Clément and Arne Storjohann: *Faster algorithms for the characteristic polynomial.* In *ISSAC 2007*, pages 307–314. ACM, New York, 2007.

[48] Storjohann, Arne: *Deterministic computation of the Frobenius form (extended abstract).* In *42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001)*, pages 368–377. IEEE Computer Soc., Los Alamitos, CA, 2001.

[49] Storjohann, Arne and Gilles Villard: *Algorithms for similarity transforms.* Technical report, Rhine Workshop on Computer Algebra, May 2000. Extended abstract.

[50] Stothers, Andrew J.: *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, 2010. http://www.maths.ed.ac.uk/pg/thesis/stothers.pdf.

[51] Thomé, Emmanuel: *Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm*. In *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, pages 323–331 (electronic), New York, 2001. ACM.

[52] Turner, William Jonathan: *Black box linear algebra with the linbox library*. PhD thesis, North Carolina State University, 2002, ISBN 0-493-97081-9.

[53] Vassilevska Williams, Virginia: *Breaking the Coppersmith-Winograd barrier*. http://www.cs.berkeley.edu/~virgi/matrixmult.pdf, 2011.

[54] Villard, Gilles: *Further analysis of Coppersmith's block Wiedemann algorithm for the solution of sparse linear systems (extended abstract)*. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, ISSAC '97, pages 32–39, New York, NY, USA, 1997. ACM, ISBN 0-89791-875-4.

[55] Villard, Gilles: *Computing the Frobenius normal form of a sparse matrix*. In *Computer algebra in scientific computing (Samarkand, 2000)*, pages 395–407. Springer, Berlin, 2000.

[56] Villard, Gilles: *Algorithmique en algèbre linéaire exacte*. Mémoire d'habilitation, Université Claude Bernard Lyon 1, 2003.

[57] Whaley, R. Clint, Antoine Petitet, and Jack J. Dongarra: *Automated empirical optimizations of software and the ATLAS project*. Parallel Computing, 27(1–2):3–35, January 2001. http://www.netlib.org/utk/people/JackDongarra/PAPERS/atlas_pub.pdf.

[58] Wiedemann, Douglas H.: *Solving sparse linear equations over finite fields*. IEEE Trans. Inform. Theory, 32(1):54–62, 1986, ISSN 0018-9448.

[59] Winograd, S.: *On multiplication of* 2×2 *matrices*. Linear Algebra and Appl., 4:381–388, 1971.

[60] Yannakakis, Mihalis: *Computing the minimum fill-in is NP-complete*. SIAM J. Algebraic Discrete Methods, 2(1):77–79, 1981, ISSN 0196-5212.

[61] Zlatev, Zahari: *Computational methods for general sparse matrices*, volume 65 of *Mathematics and its Applications*. Kluwer Academic Publishers Group, Dordrecht, 1991, ISBN 0-7923-1154-X.