Efficient Computation of the Characteristic Polynomial

Jean-Guillaume Dumas
Université Joseph Fourier,
LMC-IMAG B.P. 53,
38041 Grenoble Cedex 9,
France.
jgdumas@imag.fr

Clément Pernet
Université Joseph Fourier,
LMC-IMAG B.P. 53,
38041 Grenoble Cedex 9,
France.
pernet@imag.fr

Zhendong Wan
Department of Computer and
Information Science
University of Delaware,
Newark, DE 19716, USA.
wan@cis.udel.edu

ABSTRACT

We deal with the computation of the characteristic polynomial of dense matrices over word size finite fields and over the integers. We first present two algorithms for finite fields: one is based on Krylov iterates and Gaussian elimination. We compare it to an improvement of the second algorithm of Keller-Gehrig. Then we show that a generalization of Keller-Gehrig's third algorithm could improve both complexity and computational time. We use these results as a basis for the computation of the characteristic polynomial of integer matrices. We first use early termination and Chinese remaindering for dense matrices. Then a probabilistic approach, based on integer minimal polynomial and Hensel factorization, is particularly well suited to sparse and/or structured matrices.

Categories and Subject Descriptors

G.4 [**Mathematics and Computing**]: Mathematical Software—*Algorithm Design and Analysis*; I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation

General Terms

Algorithms, Experimentation

Keywords

Characteristic polynomial, minimal polynomial, Keller-Gehrig, probabilistic algorithm, finite field, integer, Magma

1. INTRODUCTION

Computing the characteristic polynomial of an integer matrix is a classical mathematical problem. It is closely related to the computation of the Frobenius normal form which can be used to test two matrices for similarity. Although the Frobenius normal form contains more information on the matrix than the characteristic polynomial, most algorithms to compute it are based on computations of characteristic polynomial (see for example [21, §9.7]). Using classical matrix multiplication, the algebraic time complexity for the computation of the characteristic polynomial is optimal : several algorithms require only $\mathcal{O}(n^3)$ algebraic operations (to our knowledge the oldest one is due to Danilevski [11, §24]). Now considering that the determinant can be deduced from the characteristic polynomial, and that its computation is proven to be as hard as matrix multiplication [2] the optimality is then straightforward. But with fast matrix arithmetic ($\mathcal{O}(n^\omega)$ with $2 \leq \omega < 3$), the best asymptotic time complexity is $\mathcal{O}(n^\omega \log n)$, given by Keller-Gehrig's branching algorithm [15]. Now the third algorithm of Keller-Gehrig has a $\mathcal{O}(n^\omega)$ algebraic time complexity but only works for generic matrices. In this paper we focus on the practicability of such algorithms applied on matrices over a finite field. Therefore we used the techniques developped in [4, 5], for efficient basic linear algebra operations over a finite field. We propose a new $\mathcal{O}(n^3)$ algorithm designed to take benefit of the block matrix operations; improve Keller-Gehrig's branching algorithm and compare these two algorithms. Then we focus on Keller-Gehrig's third algorithm and prove that its generalization is not only of theoretical interest but is also promising in practice.

As an application, we show that these results directly lead to an efficient computation of the characteristic polynomial of integer matrices using chinese remaindering and an early termination criterion adapted from [6]. This basic application outperforms the best existing softwares on many cases. Now better algorithms exist for the integer case, and can be more efficients with sparse or structured matrices. Therefore, we also propose a probabilistic algorithm using a black-box computation of the minimal polynomial and our finite field algorithm. This can be viewed as a simplified version of the algorithm described in [22] and [14, §7.2]. Its efficiency in practice is also very promising.

In the following we will denote by $A_{i_1 \dots i_2, j_1 \dots j_2}$ the submatrix of A located between rows i_1 and i_2 and columns j_1 and j_2 and by $A_{k,1\dots n}$ the kth row vector of A.

2. KRYLOV'S APPROACH

Among the different techniques to compute the characteristic polynomial over a field, many of them rely on the Krylov approach. A description of them can be found in [11]. They are based on the following fact: the minimal linear dependance relation between the Krylov iterates of a vector v (i.e. the sequence $(A^i v)_i$) gives the minimal polynomial $P_{A,v}^{min}$ of this sequence, and a divisor of the minimal polyno-

ISSAC'05, July 24–27, 2005, Beijing, China.

mial of A. Moreover, if X is formed by the the first independent column vectors of this sequence and $C_{P^{min}_{A,v}}$ is the companion matrix associated to $P^{min}_{A,v}$, we have $AX = XC_{P^{min}_{A,v}}$.

2.1 Minimal polynomial

We show here an algorithm to compute the minimal polynomial of the sequence of the Krylov iterates of a vector v and a matrix A. This is the monic polynomial $P^{min}_{A,v}$ of least degree such that $P(A).v = 0$. We firstly presented it in [19, 18], however, we recall it here for clarity and accessibility purposes. A similar algorithm was also simultaneously published in [16, Algorithm 2.14], but it does not take advantage of fast matrix multiplication as we do here. The idea is to compute the $n \times n$ matrix $K_{A,v}$ (called the Krylov matrix), whose ith column is the vector $A^i u$, and to perform an elimination on it. More precisely, one computes the LSP [12] factorization of $K^t_{A,v}$. Let k be the degree of $P^{min}_{A,v}$. So the first k columns of $K_{A,v}$ are linearly independent, and the $n-k$ following ones are a linear combination of the first k ones. Therefore S is triangular with its last $n-k$ rows equals to 0. Thus, the LSP factorization of $K^t_{A,v}$ can be viewed as in figure 1.

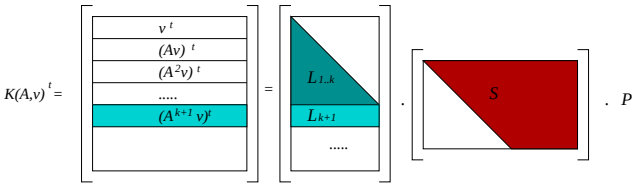


Figure 1: LSP factorization of the Krylov matrix

Now the trick is to notice that the vector $m = L_{k+1}L^{-1}_{1\ldots k}$ contains the opposites of the coefficients of $P^{min}_{A,v}$. Indeed, let us define $X = K^T_{A,v}$ therefore $X_{k+1,1\ldots n} = (A^k v)^T = \sum_{i=0}^{k-1} m_i (A^i v)^T = m \cdot X_{1\ldots k,1\ldots n}$ where $P^{min}_{A,v}(X) = X^k - m_k X^{k-1} - \cdots - m_1 X - m_0$. Thus $L_{k+1}SP = m \cdot L_{1\ldots k}SP$ and finally $m = L_{k+1}.L^{-1}_{1\ldots k}$. Algorithm 2.1 is then straightforward. The dominant operation in this algorithm is the computation of K, in $\log_2 n$ matrix multiplications, i.e. in $\mathcal{O}(n^\omega \log n)$ algebraic operations. The LSP factorization requires $\mathcal{O}(n^\omega)$ operations and the triangular system resolution, $\mathcal{O}(n^2)$. So the algebraic time complexity of this algorithm is $\mathcal{O}(n^\omega \log n)$. Now with classical matrix multiplications ($\omega = 3$), one should prefer to compute the Krylov matrix K by k successive matrix vector products. The complexity is then $\mathcal{O}(n^3)$. One can also merge the construction of the Krylov matrix and its LSP factorization so as to avoid the computation of the last $n-k$ Krylov iterates with an early termination. This reduces the time complexity to $\mathcal{O}(n^\omega \log k)$ for fast matrix arithmetic, and $\mathcal{O}(n^2 k)$ for classical matrix arithmetic. Note that if the finite field is reasonably large, then choosing v randomly makes the algorithm Monte-Carlo for the computation of the minimal polynomial of A.

Algorithm 2.1 `MinPoly` : Minimal Polynomial of A and v

Require: A a $n \times n$ matrix and v a vector over a field
Ensure: $P^{A,v}_{\min}(X)$ minimal polynomial of $(A^i v)_i$
1: $K_{1\ldots n,1} = v$
2: **for** $i = 1$ to $\log_2(n)$ **do**
3: $\quad K_{1\ldots n,2^i\ldots 2^{i+1}-1} = A^{2^{i-1}} K_{1\ldots n,1\ldots 2^i-1}$
4: **end for**
5: $(L, S, P) = \mathtt{LSP}(K^T), k = \mathrm{rank}(K)$
6: $m = L_{k+1}.L^{-1}_{1\ldots k}$
7: return $P^{A,v}_{\min}(X) = X^k - \sum_{i=0}^{k-1} m_i X^i$

2.2 LU-Krylov algorithm

From algorithm 2.1, we then derive the computation of the characteristic polynomial since it produces the k first independent Krylov iterates of v. They form a basis of an invariant subspace under the action of A (the first invariant subspace of A if $P^{min}_{A,v} = P^{min}_{A}$). The idea is to use the elimination performed on this basis to compute a basis of its supplementary subspace. Then a recursive call on a $n-k \times n-k$ matrix provides the remaining factors. The algorithm follows, where k, P, and S are as in algorithm 2.1.

Algorithm 2.2 `LUK` : LU-Krylov algorithm

Require: A a $n \times n$ matrix over a field
Ensure: P^A_{char} the characteristic polynomial of A
1: Pick a random vector v
2: $P^{A,v}_{\min} = \mathrm{MinPoly}(A, v)$ of degree k
$\quad \{X = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} [S_1|S_2]P \text{ is computed}\}$
3: **if** $(k = n)$ **then**
4: $\quad$ return $P^A_{\mathrm{char}} = P^{A,v}_{\min}$
5: **else**
6: $\quad A' = PA^TP^T = \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix}$ where A'_{11} is $k \times k$.
7: $\quad P^{A'_{22}-A'_{21}S_1^{-1}S_2}_{\mathrm{char}}(X) = \mathtt{LUK}(A'_{22} - A'_{21}S_1^{-1}S_2)$
8: $\quad$ return $P^A_{\mathrm{char}} = P^{A,v}_{\min} \times P^{A'_{22}-A'_{21}S_1^{-1}S_2}_{\mathrm{char}}$
9: **end if**

THEOREM 2.1. *Algorithm LU-Krylov computes the characteristic polynomial of a $n \times n$ matrix A in $\mathcal{O}(n^3)$ field operations.*

PROOF. The first k rows of X ($X_{1\ldots k,1\ldots n}$) form a basis of the invariant subspace generated by v. Moreover we have $X_{1..k}A^T = C^T_{P^{A,v}_{\min}} X_{1..k}$ Indeed $\forall i < k$ we have $X_i A^T = \left(A^{i-1}v\right)^T A^T = \left(A^i v\right)^T = X_{i+1}$ and $X_k A^T = \left(A^{k-1}v\right)^T A^T = \left(A^k v\right)^T = \sum_{i=0}^{k-1} m_i \left(A^i v\right)^T$ The idea is now to complete this basis into a basis of the whole space. Viewed as a matrix, this basis form the $n \times n$ invertible matrix $\overline{X}$:

$$\overline{X} = \underbrace{\begin{bmatrix} L_1 & 0 \\ 0 & I_{n-k} \end{bmatrix}}_{\overline{L}} \underbrace{\begin{bmatrix} S_1 & S_2 \\ 0 & I_{n-k} \end{bmatrix}}_{\overline{S}} P = \begin{bmatrix} X_{1\ldots k,1\ldots n} \\ \begin{bmatrix} 0 & I_{n-k} \end{bmatrix} P \end{bmatrix}$$

Let us compute

$$\begin{aligned} \overline{X}A^T\overline{X}^{-1} &= \left[\begin{array}{c|c} C^T & 0 \\ \hline \begin{bmatrix} 0 & I_{n-k} \end{bmatrix} PA^TP^T\overline{S}^{-1}\overline{L}^{-1} & \end{array}\right] \\ &= \left[\begin{array}{c|c} C^T & 0 \\ \hline \begin{bmatrix} A'_{21} & A'_{22} \end{bmatrix} \overline{S}^{-1}\overline{L}^{-1} & \end{array}\right] = \left[\begin{array}{c|c} C^T & 0 \\ \hline Y & X_2 \end{array}\right] \end{aligned}$$

with $X_2 = A'_{22} - A'_{21}S_1^{-1}S_2$. By a similarity transformation, we thus have reduced A to a block triangular matrix.

Then the characteristic polynomial of A is the product of the characteristic polynomial of these two diagonal blocks: $P^A_{\text{char}} = P^{A,v}_{\min} \times P^{A'_{22}-A'_{21}S_1^{-1}S_2}_{\text{char}}$. Now for the time complexity, we will denote by $T_{\texttt{LUK}}(n)$ the number of field operations for this algorithm applied on a $n \times n$ matrix, by $T_{\texttt{minpoly}}(n,k)$ the cost of the algorithm 2.1 applied on a $n \times n$ matrix having a degree k minimal polynomial, by $T_{\texttt{LSP}}(m,n)$ the cost of the LSP factorization of a $m \times n$ matrix, by $T_{\texttt{trsm}}(m,n)$ the cost of the simultaneous resolution of m triangular systems of dimension n, and by $T_{\texttt{MM}}(m,k,n)$ the cost of the multiplication of a $m \times k$ matrix by a $k \times n$ matrix. The values of $T_{\texttt{LSP}}$ and $T_{\texttt{trsm}}$ can be found in [5]. Then, using classical matrix arithmetic, we have: $T_{\texttt{LUK}}(n) = T_{\texttt{minpoly}}(n,k) + T_{\texttt{LSP}}(k,n) + T_{\texttt{trsm}}(n-k,k) + T_{\texttt{mm}}(n-k,k,n-k) + T_{\texttt{LUK}}(n-l) = \mathcal{O}(n^2k + k^2n + k^2(n-k) + k(n-k)^2) + T_{\texttt{LUK}}(n-k) = \mathcal{O}(\sum_i n^2k_i + k_i^2 n) = \mathcal{O}(n^3)$, The latter being true since $\sum_i k_i = n$ and $\sum_i k_i^2 \leq n^2$. □

We have thus derived a deterministic algorithm from a probabilistic one. When algorithm 2.1 fails, it still returns a factor of the true minimal polynomial and the next recursive calls then compute the forgotten factors. Note also that when using fast matrix arithmetic, it is no longer possible to sum the $\log(k_i)$ into $\log(n)$ nor the $k_i^{\omega-2}n^2$ into n^ω. Therefore the best known algebraic time complexity, $\mathcal{O}(n^\omega \log n)$, can not be reached by such an algorithm. We thus focus on the second algorithm of Keller-Gehrig achieving this best known time complexity.

2.3 Improving Keller-Gehrig branching algorithm

In [15], Keller-Gehrig presents a so called branching algorithm, computing the characteristic polynomial of a $n \times n$ matrix over a field K in the best known time complexity : $\mathcal{O}(n^\omega \log n)$ field operations. The idea is to compute the Krylov iterates of a several vectors at the same time, so as to replace several matrix vector products by a fast matrix multiplication. More precisely, the algorithm computes a sequence of $n \times n$ matrices $(V_i)_i$ whose columns are the Krylov iterates of vectors of the canonical basis. V_0 is the identity matrix (every vector of the canonical basis is present). At the i-th iteration, the algorithm computes the next 2^i Krylov iterates of the remaining vectors. Then a Gaussian elimination determines the linear dependencies between them so as to form V_{i+1} by picking the n linearly independent vectors. The algorithm ends when no more iterate can be added ($V_{i+1} = V_i$). Then the matrix $V_i^{-1}AV_i$ is block upper triangular with companion blocks on the diagonal. The polynomials of these blocks are the minimal polynomials of each of the sequence of Krylov iterates, and their product is the characteristic polynomial of the input matrix. The removal of the linear dependencies is performed by a step-form elimination algorithm defined by Keller-Gehrig. Its formulation is rather sophisticated, and we propose to replace it by the column reduced form algorithm (algorithm 2.3) using the more standard LQUP factorization [12]. More precisely, the step form elimination of Keller-Gehrig, the LQUP factorization of Ibarra & Al. and the echelon form elimination (see e.g. [21]) are equivalent and can be used to determine the linear dependencies in a set of vectors. Our second improvement is to apply the idea of algorithm 2.1 to compute the polynomials associated to each companion block, instead of computing $V^{-1}AV$. The permutation Q of the

Algorithm 2.3 `ColReducedForm`

Require: A a $m \times n$ matrix of rank r
Ensure: r linearly independent columns of A
1: $(L,Q,U,P,r) = \text{LQUP}(A^T)$ $(r = rank(A))$
2: **return** $([I_r 0](Q^T A^T))^T$

`LQUP` factorization indicates the positions of the linearly independant blocks of iterates in W. To each of these blocks, one can associate a block column in L. Now applying the triangular system resolution of algorithm 2.1 to this block column will compute the coefficients of the first linear dependency between these iterates. Since the Krylov iterates are already computed, and the last call to `ColReducedForm` performed the elimination on them, there only remains to solve triangular systems. We thus get the coefficients of each polynomial, for a total cost of $\mathcal{O}(n^2)$. Algorithm 2.4 shows

Algorithm 2.4 `KGB`: Keller-Gehrig Branching algorithm

Require: A a $n \times n$ matrix over a field
Ensure: $P^A_{\text{char}}(X)$ the characteristic polynomial of A
1: $i = 0$
2: $V_0 = I_n = (V_{0,1}, V_{0,2}, \ldots, V_{0,n})$
3: $B = A$
4: **while** $(\exists k, V_k$ has 2^i columns) **do**
5: **for all** j **do**
6: **if** ($V_{i,j}$ has strictly less than 2^i columns) **then**
7: $W_j = V_{i,j}$
8: **else**
9: $W_j = [V_{i,j} | BV_{i,j}]$
10: **end if**
11: **end for**
12: $W = (W_j)_j$
13: V_{i+1} = ColReducedForm(W) (remember L and Q from `LQUP`)
$\{V_{i+1,j}$ are the remaining vectors of W_j in $V_{i+1}\}$
14: $B = B \times B$
15: $i = i + 1$
16: **end while**
17: **for all** j **do**
18: Let s,t be the indexes of the first and last column of linearly independent iterates of the vector e_j in W (given by Q)
19: $m = L_{t+1,s\ldots t} . L^{-1}_{s\ldots t, s\ldots t}$
20: $P_j(X) = X^{t-s} - \sum_{i=0}^{t-s-1} m_i X^i$
21: **end for**
22: **return** $\Pi_j P_j$

these modifications. The operations in the `while` loop have a $\mathcal{O}(n^\omega)$ algebraic time complexity. This loop is executed at most $\log n$ times and the overall algebraic time complexity is therefore $\mathcal{O}(n^\omega \log n)$. More precisely it is $\mathcal{O}(n^\omega \log k_{\max})$ where $k_{\max}$ is the degree of the largest invariant factor.

2.4 Experimental comparisons

We implemented these two algorithms, using a finite field representation on double size machine floating point numbers : `modular<double>` (see [5]), and the efficient routines for finite field linear algebra `FFLAS-FFPACK` presented in [5, 4]. We also only considered classic matrix arithmetic. We ran them on a series of matrices of order 300 whose Frobenius normal forms had different number of diagonal com-

panion blocks. Figure 2 shows the computational time on a Pentium IV 2.4Ghz with 512Mb of RAM. It appears that

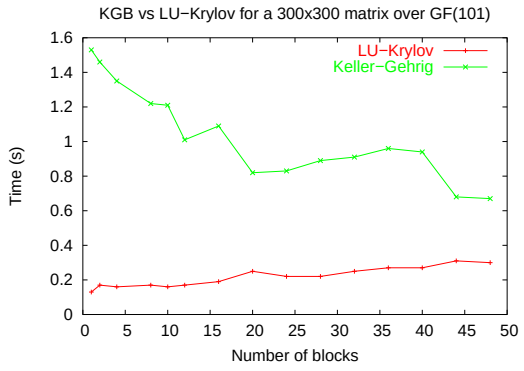


Figure 2: LU-Krylov vs. KGB

`LUK` is faster than `KGB` on every matrix. This is due to the extra log$n$ factor in the time complexity of the latter. One can note that the computational time of `KGB` is decreasing with the number of blocks. This is due to the fact that the $\log(n)$ is in fact $\log(k_{\max}$ where $k_{\max}$ is the size of the largest block. This factor is decreasing when the number of blocks increases. Conversely, `LUK` computational time is almost constant. It slightly increases, due to the increasing number of rectangular matrix operations: their computation are less efficient than square matrix operations, due to BLAS optimizations of memory accesses.

3. TOWARD AN OPTIMAL ALGORITHM

As mentioned in the introduction, the best known algebraic time complexity for the computation of the characteristic polynomial is not optimal in the sense that it is not $\mathcal{O}(n^\omega)$ but $\mathcal{O}(n^\omega \log n)$. However, Keller-Gehrig gives a third algorithm (let us name it `KG3`), having this time complexity but only working on generic matrices. In the following, we will use Keller-Gehrig's definition of a generic matrix : each of its coefficients can be considered as an independent indeterminate.

To get rid of the extra $\log(n)$ factor, it is no longer based on a Krylov approach. The algorithm is inspired by a $\mathcal{O}(n^3)$ algorithm by Danilevski (described in [11]), improved into a block algorithm. The genericity assumption ensures the existence of a series of similarity transformations changing the input matrix into a companion matrix.

3.1 Comparing the constants

The optimal "big-O" complexity often hides a large constant in the exact expression of the time complexity. This makes these algorithms impracticable since the improvement induced is only significant for huge matrices. However, we show in the following lemma that the constant of `KG3` is very close to the one of `LUK`.

LEMMA 3.1. *The computation of the characteristic polynomial of a $n \times n$ generic matrix using `KG3` algorithm requires $K_\omega n^\omega + o(n^\omega)$ algebraic operations, where*

$$K_\omega = C_\omega \left[-\frac{2^{\omega-2}}{2(2^{\omega-2}-1)(2^{\omega-1}-1)(2^\omega-1)} - \frac{1}{2^\omega-1} + \frac{1}{(2^{\omega-2}-1)(2^{\omega-1}-1)} - \frac{3}{2^{\omega-1}-1} + \frac{2}{2^{\omega-2}-1} + \frac{1}{(2^{\omega-2}-1)(2^\omega-1)} + \frac{2^{\omega-2}}{2(2^{\omega-2}-1)(2^{\omega-1}-1)^2} \right]$$

and C_ω is the constant in the algebraic time complexity of the matrix multiplication.

The proof and a description of the algorithm are given in appendix A. In particular, with classical matrix arithmetic ($\omega = 3, C_\omega = 2$), we have on the one hand $K_\omega = 176/63 \approx 2.794$. On the other hand, the algorithm 2.2 called on a generic matrix simply computes the n Krylov vectors $A^i v$ ($2n^3$ operations), computes the LUP factorization of these vectors ($2/3n^3$ operations) and the coefficients of the polynomial by the resolution of a triangular system ($\mathcal{O}(n^2)$). Therefore, the constant for this algorithm is $2+2/3 \approx 2.667$. These two algorithms have thus a similar algebraic complexity, LU-Krylov being slightly faster than Keller-Gehrig's third algorithm. We now compare them in practice.

3.2 Experimental comparison

We claim that the study of precise algebraic time complexity of these algorithms is worthwhile in practice. Indeed these estimates directly correspond to the computational time of these algorithms applied over finite fields. Therefore

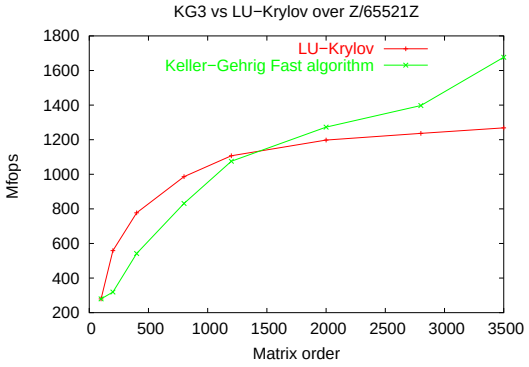


Figure 3: LUK vs. KG3: speed comparison

we ran these algorithms on a word size prime finite field with modular arithmetic. Again we used `modular<double>` and `FFLAS-FFPACK`. These routines can use fast matrix arithmetic, we, however, only used classical matrix multiplication so as to compare two $\mathcal{O}(n^3)$ algorithms having similar constants (2.67 for `LUK` and 2.794 for `KG3`). We used random dense matrices over the finite field $\mathbb{Z}_{65521}$, as generic matrices. We report the computational speed in Mfops (Millions of field operations per second) for the two algorithms on figure 3. It appears that LU-Krylov is faster than `KG3` for small matrices (better algebraic time complexity), but for matrices of order larger than 1500, `KG3` is faster. Indeed, the $\mathcal{O}(n^3)$ operations are differently performed: LU-Krylov computes the Krylov basis by n matrix-vector products, whereas `KG3` only uses matrix multiplications. Now, as the order of the matrices increases, the BLAS routines

provide better efficiency for matrix multiplications than for matrix vector products. Once again, algorithms exclusively based on matrix multiplications are preferable: from the complexity point of view, they make it possible to achieve $\mathcal{O}(n^\omega)$ time complexity. In practice, they promise the best efficiency thanks to the BLAS better memory management.

4. OVER THE INTEGERS

There exist several algorithms to compute the characteristic polynomial of an integer matrix. A first idea is to perform the algebraic operations over the ring of integers, using exact divisions [1] or by avoiding divisions [3, 13, 14]. We focus here on field approaches. Concerning the bit complexity of this computation, a first approach, using Chinese remaindering gives $\mathcal{O}^\sim(n^{\omega+1}\log\|A\|)$ bit operations ($\mathcal{O}^\sim$ is the "soft-O" notation, hiding logarithmic and polylogarithmic factors in n and $\|A\|$). Baby-step Giant-step techniques applied by Kaltofen [13] improves this complexity to $\mathcal{O}^\sim(n^{3.5}\log\|A\|)$ (using classical matrix arithmetic). Lastly, the recent improvement of [14, §7.2], combining Coppersmith's block-Wiedemann techniques set the best known exponent for this computation to 2.697263 using fast matrix arithmetic.

Our goal here is not to give an exhaustive comparison of these methods, but to show that a straightforward application of our finite field algorithm `LU-Krylov` is already very efficient and can outperform the best existing softwares. A first dense deterministic algorithm, using Chinese remaindering is given in section 4.1. Then we propose in section 4.2 a probabilistic algorithm that can be adapted for dense or for sparse and structured matrices. It combines the early termination technique of [6, §3.3], and a recent alternative to chinese remaindering in [22], also developed in [14, §7.2]. Lastly we compare implementations of these algorithms in practice.

4.1 Dense deterministic : Chinese remainder

The first naive way of computing the characteristic polynomial is to use Hadamard's bound [8, Theorem 16.6] to show that any integer coefficient of the characteristic polynomial has the order of n bits:

LEMMA 4.1. *Let $A \in \mathbb{Z}^{n\times n}$, with $n > 4$, whose coefficients are bounded in absolute value by $B > 1$. The coefficients of the characteristic polynomial of A are denoted by c_j. Then $\forall j\ |c_j| \leq max_{i=0..\frac{-1+\sqrt{1+4en}}{2e}} \binom{n}{i}\sqrt{(n-i)B^2}^{(n-i)}$ and $\log_2(|c_j|) \leq \frac{n}{2}\left(\log_2(n) + \log_2(B^2) + 0.21163175\right)$*

PROOF. c_i, the i-th coefficient of the characteristic polynomial, is an alternate sum of all the $(n-i)\times(n-i)$ diagonal minors of A. It is therefore bounded by $H(n,i) = \binom{n}{i}\sqrt{(n-i)B^2}^{(n-i)}$. First note, that from the symmetry of the binomial coefficients we only need to explore the $\lfloor n/2\rfloor$ first ones, since $\sqrt{(n-i)B^2}^{(n-i)} > \sqrt{iB^2}^{i}$ for $i < \lfloor n/2\rfloor$. Now, the lemma claims that actually the maximal value must occur within the $\mathcal{O}(\sqrt{n})$ first ones. The lemma is true for $j = 0$ by Hadamard's bound. And for $j = 1$, we have $\log_2(H(n,i)) < \frac{n}{2}(\log_2(n) + \log_2(B^2) + 0.21163175)$ as soon as $n > 4$, since the difference is decreasing in n. Then from Stirling's formula ($n! = (1+\epsilon(n))\sqrt{2\pi n}\frac{n^n}{e^n}$), we have $\forall i \geq 2\ \binom{n}{i} < \frac{1+\epsilon(n)}{\sqrt{2\pi}}\sqrt{\frac{n}{i(n-i)}}\left(\frac{n}{i}\right)^i\left(\frac{n}{n-i}\right)^{n-i}$. Now first $\frac{1}{12n} < \epsilon(n) < \frac{1}{12n+1}$. Therefore for $n > 4$, $\log_2\left(\frac{1+\epsilon(n)}{\sqrt{2\pi}}\right) \leq -1.296$. Then $\frac{n}{i(n-i)}$ is decreasing in i for $i < \lfloor n/2\rfloor$ so that its maximum is $\frac{n}{2(n-2)}$.

Consider now $K(n,i) = \left(\frac{n}{i}\right)^i\left(\frac{n}{n-i}\right)^{n-i}\sqrt{(n-i)B^2}^{(n-i)}$. We have $\log_2(K(n,i)) = \frac{n-i}{2}\log_2(B^2) + \frac{n}{2}\log_2(n) + \frac{n}{2}T(n,i)$, where $T(n,i) = \log_2(\frac{n}{n-i}) + \frac{i}{n}\log_2(\frac{n-i}{i^2})$. Well $T(n,i)$ is maximal for $i = \frac{-1+\sqrt{1+4en}}{2e}$ as announced in the lemma. We end with the fact that $T(n,i) - \frac{2}{n}1.296 + \frac{1}{n}\log_2(\frac{n}{2(n-2)})$ is maximal over $\mathbb{Z}$ for $n = 15$ where it is lower than 0.208935. The latter is lower than 0.21163175. □

Well, $\begin{bmatrix} 1 & 1 & 1 & 1 & 1\\ 1 & 1 & -1 & -1 & -1\\ 1 & -1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1 & -1\\ 1 & -1 & -1 & -1 & 1 \end{bmatrix}$ has $X^5 - 5X^4 + 40X^2 - 80X + 48$ for characteristic polynomial and $80 = \binom{5}{1}\sqrt{4}^4$ is greater than Hadamard's bound 55.9, and less than our bound 80.66661.

Note that the complexity of finding the maximal value is only $\mathcal{O}(\sqrt{n})$, since $\binom{n}{i} = \binom{n}{i-1}\frac{n-i+1}{i}$. Note also that the numerical bound improves the one used in [9, lemma 2.1] since $0.21163175 < 2 + \log_2(e) \approx 3.4427$. Now, using fast integer arithmetic and the fast Chinese remaindering algorithm [8, Theorem 10.25], one gets the overall complexity for the dense integer characteristic polynomial via Chinese remaindering of $\mathcal{O}(n^4(log(n) + log(B)))$. In the following, we focus on probabilistic methods that can improve drastically the computation time for both dense and structured matrices.

4.2 Probabilistic improvements

The idea here, is to limit the chinese remaindering to the computation of the minimal polynomial. It is then factored by Hensel lifting algorithm. One modular characteristic polynomial is then computed (resuming algorithm `LUK` from one modular minimal polynomial). Lastly the multiplicities of each factor are recovered from this last polynomial.

4.2.1 Early termination

To reduce the number of homomorphic computations of the minimal polynomial, we use the early termination of [6, §3.3], to stop the remaindering :

LEMMA 4.2. *[6] Let $v \in \mathbb{Z}$ be a coefficient of the characteristic polynomial, and U be a given upper bound on $|v|$. Let P be a set of primes and let $\{p_1 \dots p_k, p^*\}$ be a random subset of P. Let l be a lower bound such that $p^* > l$ and let $M = \prod_{i=1}^k p_i$. Let $v_k = v \mod M$, $v^* = v \mod p^*$ and $v_k^* = v_k \mod p^*$ as above. Suppose now that $v_k^* = v^*$. Then $v = v_k$ with probability at least $1 - \frac{\log_l(\frac{U-v_k}{M})}{|P|}$.*

The proof is that of [6, lemma 3.1]. The probabilistic algorithm is then straightforward: after each modular computation of a minimal polynomial, the algorithm stops if every coefficient is unchanged. It is of the Monte-Carlo type: always fast with a controlled probability of success. This probability is bounded by the probability of lemma 4.2. In practice it is much higher, since the n coefficients are checked. But since they are not independent, we are not able to produce a tighter bound. Note that one could have just applied this early termination to the remaindering of the algorithm of section 4.1. But the improvement here, is that the number of homomorphic computations is reduced if the minimal polynomial has a small degree. The recovery of the rest of the integer characteristic polynomial is then

cheap (an integer factorization and one modular computation of the whole characteristic polynomial as shown next).

4.2.2 Multiplicities search and sparse/structured

For dense matrices, the modular computation can be done by algorithm 2.1. The integer coefficients are recovered by chinese remaindering. The idea is to compute first the integer minimal polynomial and to factor it. Then only the multiplicities of the factors remain to be found. To recover them, we perform a modular computation of the characteristic polynomial and compute the multiplicities that match with this modular polynomial. By structured or sparse matrices we mean matrices for which the matrix-vector product can be performed with less than n^2 arithmetic operations. In this case one can use the specialized methods of [6, §3] (denoted by `IMP`). Note that one modular computation of the characteristic polynomial will still be dense. Thus it is not a pure sparse or structured algorithm (memory storage can become the limiting factor), but since the dominant operation is usually the computation of the minimal polynomial, this approach still make sense.

Algorithm 4.1 `CIA` : Characteristic polynomial over Integers Algorithm

Require: $A \in \mathbb{Z}^{n\times n}$, even as a blackbox, ϵ.
Ensure: The characteristic polynomial of A with a probability of $1-\epsilon$.
1: $\eta = 1 - \sqrt{1-\epsilon}$
2: $P^A_{\min} = \texttt{IMP}(A, \eta)$ via [6, §3] or `CRA(MinPoly)`(A, η)
3: Factor $P^A_{\min}$ over the integers, e.g. by Hensel's lifting.
4: $B = 2^{\frac{n}{2}\left(\log_2(n)+\log_2(||A||^2)+0.21163175\right)}$
5: Choose a random prime p in a set of $\frac{1}{\eta}\log_2(\sqrt{n+1}\ 2^{n+1}B+1)$ primes.
6: Compute P_p the characteristic polynomial of A mod p via `LUK`.
7: **for all** f_i irreducible factor of $P^A_{\min}$ **do**
8: Compute $\overline{f}_i \equiv f_i \mod p$.
9: Find α_i the multiplicity of $\overline{f}_i$ within P_p.
10: **if** $\alpha_i == 0$ **then** Return "FAIL". **end if**
11: **end for**
12: Compute $P^A_{\text{char}} = \prod f_i^{\alpha_i} = X^n - \sum_{i=0}^{n-1} a_i X^i$.
13: **if** $(\sum \alpha_i \text{degree}(f_i) \neq n)$ or $(Trace(A) \neq a_{n-1})$ **then**
14: Return "FAIL".
15: **end if**
16: Return P^A_{char}.

THEOREM 4.3. *Algorithm 4.1 is correct. It is probabilistic of the Monte-Carlo type.*

PROOF. Let $P^{\min}$ be the integer minimal polynomial of A and $\tilde{P}^{\min}$ the result of the call to `IMP`. With a probability of $\sqrt{1-\epsilon}$, $P^{\min} = \tilde{P}^{\min}$. Then the only problem that can occur is that an irreducible factor of $P^{\min}$ divides another factor when taken modulo p, or equivalently, that p divides the resultant of these polynomials. Now from [8, Algorithm 6.38] and lemma 4.1 an upper bound on the size of this resultant is $\log_2(\sqrt{n+1}\ 2^{n+1}B+1)$. Therefore, the probability of choosing a bad prime is less than η. Thus the result will be correct with a probability greater than $1-\epsilon$ □

This algorithm is also able to detect many erroneous results and return "FAIL" instead. In that case, one could perform the computation again. However, since we don't have any certification of success, it can not be of the type "Las-Vegas". The first case is when $P^{\min} = \tilde{P}^{\min}$ and a factor of $P^{\min}$ divides another factor modulo p. In such a case, the exponent of this factor will appear twice in the reconstructed characteristic polynomial. The overall degree being greater than n, `FAIL` will be returned. Now, if $P^{\min} \neq \tilde{P}^{\min}$, the tests $\alpha_i > 0$ will detect it unless $\tilde{P}^{\min}$ is a divisor of $P^{\min}$, say $P^{\min} = \tilde{P}^{\min}Q$. In that case, on the one hand, if Q does not divide $\tilde{P}^{\min}$ modulo p, the total degree will be lower than n and `FAIL` will be returned. On the other hand, a wrong characteristic polynomial will be reconstructed, but the trace test will detect most cases. The overall complexity is not better than e.g. [22, 14] but its practical timings shown in section 4.3 prove its efficiency.

4.3 Experimental results

We now compare implementations of the previously described algorithms. `ILUK-det` from section 4.1 is dense deterministic. We declined the probabilistic algorithm 4.1 into a dense version, `CIA-dense`, (using algorithm 2.1) and a sparse version, `CIA-prob`, (using algorithm [6, §3]). We used the same programming environment as in section 2.4. The polynomial factorization is computed with NTL[1] via Hensel's factorization. The choice of moduli is there linked to the constraints of the matrix multiplication of `FFLAS`. Indeed, the wrapping of numerical BLAS is only valid if $n(p-1)^2 < 2^{53}$ (the result can be stored in the 53 bits of the `double` mantissa). Therefore, we chose to sample the primes between 2^m and 2^{m+1} (where $m = \lfloor 25.5 - \frac{1}{2}\log_2(n) \rfloor$). This set was always sufficient in practice. Even with 5000×5000 matrices, $m = 19$ and there are 38658 primes between 2^{19} and 2^{20}. Now if the coefficients of the matrix are between -1000 and 1000, the upper bound on the coefficients of the minimal polynomial is $\log_{2^m}(U) \approx 4267.3$. Therefore, the probability of finding a bad prime is lower than $4267.3/38658 \approx 0.1104$. Then performing a couple a additional modular computations to check the result will improve this probability. In this example, only 18 more computations (compared to the 4268 required for the deterministic computation) are enough to ensure a probability of error lower than 2^{-55}. At this rate, there is e.g. more chances that cosmic radiations perturbed the output! [2] In the following, the probabibilistic algorithms will always ensure a probability of failure less than 2^{-55}. Table 1 focuses on dense matrices with coefficients chosen uniformly between 0 and 10. Their minimal polynomial equals their characteristic polynomial. `ILUK-det` and `CIA-dense` are compared `Maple-v9` and `Magma-2.11`. We ran these tests on an Athlon 2200 (1.8 Ghz) with 2Gb of RAM, running Linux-2.4.[3] The implementation of Berkowitz algorithm used by `Maple` has prohibitive computational timings. `Magma` uses a p_adic algorithm that combines a probabilistic computation and a proof of correctness to make it deterministic. For these matrices, this proof is free, since the minimal polynomial equals the characteristic polynomial [20]. Comparing deterministic algorithms, `ILUK-det` is slightly faster than `magma` on most cases. For matrices of order over 800, `magma` tries to allocate

[1] `www.shoup.net/ntl`
[2] Indeed, cosmic rays only can be responsible for 10^5 software errors in 10^9 chip-hours at sea level[17] . At 1GHz, this makes 1 error every 2^{55} computations.
[3] We are grateful to the Medicis computing center hosted by the CNRS STIX lab : `medicis.polytechnique.fr/medicis`.

n	Maple	Magma	ILUK-det	CIA
100	163s	0.34s	0.23s	0.20s
200	3355s	4.45s	3.95s	3.25s
		11.1Mb	3.5Mb	3.5Mb
400	74970s	69.8s	91.4s	71.74s
		56Mb	10.1Mb	10.1Mb
800		1546s	1409s	1110s
		403Mb	36.3Mb	36.3Mb
1200		8851s	7565s	5999s
		1368Mb	81Mb	81Mb
1500		MT	21010s	16705s
			136Mb	136Mb
3000		MT	349494s	
			521Mb	

Table 1: Characteristic polynomial of dense integer matrices (comp. time and mem. allocation)

more than 2Gb of RAM to store the corresponding integer Krylov space, and the computation crashes (MT stands for Memory Thrashing). The memory usage of our implementations is much smaller than that of `magma`, and makes it possible to handle larger matrices. Concerning the probabilistic algorithms, `CIA` improves the computational time of the deterministic one of roughly 25% and is faster than `magma-prob`. In table 2, we denote by d the degree of the integer minimal polynomial and by ω the average number of nonzero elements per row within the sparse matrix. We

Matrix	A	$U^{-1}AU$	A^TA	B	$U^{-1}BU$	B^TB
n	300	300	300	600	600	600
d	75	75	21	424	424	8
ω	1.9	300	2.95	4	600	13
Magma-prob	1.14	7.11	**0.23**	6.4	184.7	6.04
Magma-det	1.31	10.55	0.24	6.4	185	6.07
ILUK-det	1.1	93.3	64.87	68.4	2305	155.3
CIA-sparse	**0.32**	4.32	0.81	**4.4**	352.6	**2.15**
CIA-dense	1.22	**1.3**	0.87	38.9	**42.6**	2.57
IMP-sparse	0.03	4.03	0.02	1.62	349	0.08
IMP-dense	0.93	1.0	0.08	36.2	39.9	0.5
Fact	0.04	0.04	0.01	0.6	0.58	0.01
Mul	0.25	0.25	0.78	2.17	2.08	2.06

Table 2: `CIA` on sparse or structured matrices

show the computational times of algorithm 4.1 (`CIA`), decomposed into the time for the integer minimal polynomial computation (`IMP-sparse` for [6, §3] and `IMP-dense` for 2.1 and chinese remaindering), the factorization of this polynomial (`Fact`) and the computation of the multiplicities (`Mul`). They are compared to the timings of the algorithm of section 4.1 and that of `magma` with and without the proof of correctness. We used two sparse matrices A and B of order 300 and 600, having a minimal polynomial of degree respectively 75 and 424. A is the almost empty matrix `Frob08blocks` and is in Frobenius normal form with 8 companion blocks and B is the matrix `ch5-5.b3.600x600.sms` presented in [6]. Again `ILUK-det` is faster than `magma` but still expensive. On the two matrices A and B, `CIA-sparse` is the fastest thanks to its usage of sparsity. Then, we made these matrices dense with an integral similarity transformation. The lack of sparsity slows down both `magma` and `CIA-sparse`, whereas `CIA-prob` maintains similar timings and outperforms every other algorithms. Lastly, we used symmetric matrices with small minimal polynomial (A^TA and B^TB). `CIA-sparse` is still rather efficient (the best on B^TB), but `magma` appears to be extremely fast on A^TA. We report in table 3 on

Matrix	n	ω	magma-sparse	CIA-sparse	CIA-dense
TF12	552	7.6	10.12s	6.8s	61.77s
Tref500	500	16.9	112s	65.14s	372.6s
mk9b3	1260	3	48.4s	31.25s	433s

Table 3: `CIA` on other sparse matrices

some comparisons using other sparse matrices available at `www-lmc.imag.fr/lmc-mosaic/Jean-Guillaume.Dumas`. There, `ILUK-det` is rather expensive, although it is slightly faster than `magma`. The probabilistic approach of algorithm `CIA` specialized for dense or sparse matrices appears to reach the best computation times in almost every cases.

5. CONCLUSION

We presented a new algorithm for the computation of the characteristic polynomial over a finite field, and proved its efficiency in practice. We also considered Keller-Gehrig's third algorithm and showed that its generalization would be not only interesting in theory but produce a practicable algorithm. There, improvements of the dense computation over a finite field could be achieved and, at least, some heuristics could be built when a row-reduced form elimination can ensure generic rank profile.

We applied our algorithm for the computation of the integer characteristic polynomial in two ways: a simple deterministic use of Chinese remaindering for dense matrix computations, and a probabilistic one, using integer minimal polynomial computation. The latter can be specialized for dense or sparse and structured matrices. This last algorithm outperforms existing softwares for this task. Moreover we showed that the recent improvements of [22, 14] should be highly practicable since the successful `CIA` algorithm is inspired by their ideas. It remains to show how much they improve the simple approach of `CIA`.

Lastly, concerning the sparse computations, the search for multiplicities could be done by a pure blackbox algorithm, and would make it possible to handle problems of much higher size. These techniques have to be studied and compared to the blackbox algorithms of [23] and of [7], where preconditionners are introduced which use can be prohibitive in practice.

Acknowledgement

We are indebted to the two anonymous referee for their worthy suggestions and especially to Allan Steel, for his cooperation and his helpful explanations about Magma.

References

[1] J. Abdeljaoued and G. I. Malaschonok. Efficient algorithms for computing the characteristic polynomial in a domain. *J. of Pure and Applied Algebra*, 156:127–145, 2001.

[2] W. Baur and V. Strassen. The complexity of partial derivatives. *Theor. Computer Science*, 22(3):317–330, 1983.

[3] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, 18(3):147–150, 1984.

[4] J.-G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In T. Mora, editor, *ISSAC'2002*. ACM Press, New York, July 2002.

[5] J.-G. Dumas, P. Giorgi, and C. Pernet. FFPACK: Finite field linear algebra package. In Gutierrez [10].

[6] J.-G. Dumas, B. D. Saunders, and G. Villard. On efficient sparse integer matrix Smith normal form computations. *J. Symb. Comp.*, 32(1/2):71–99, July–Aug. 2001.

[7] W. Eberly. Reliable krylov-based algorithms for matrix null space and rank. In Gutierrez [10].

[8] J. v. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press. 1999.

[9] M. Giesbrecht and A. Storjohann. Rational forms of integer matrices. *J. Symb. Comp.*, 34(3):157–172, 2002.

[10] J. Gutierrez, editor. *ISSAC'2004. Santander, Spain*. ACM Press, New York, July 2004.

[11] A. Householder. *The Theory of Matrices in Numerical Analysis*. Blaisdell, Waltham, Mass., 1964.

[12] O. H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. of Algorithms*, 3(1):45–56, Mar. 1982.

[13] E. Kaltofen. On computing determinants of matrices without divisions. In P. S. Wang, editor, *ISSAC'92*. ACM Press, New York, July 1992.

[14] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Comp. Complexity*, 13:91–130, 2004.

[15] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoretical computer science*, 36:309–317, 1985.

[16] H. Lombardi and J. Abdeljaoued. *Méthodes matricielles - Introduction à la complexité algébrique*. Springer, 2004.

[17] T. J. O'Gorman, J. M. Ross, A. H. Taber, J. F. Ziegler, H. P. Muhlfeld, C. J. Montrose, H. W. Curtis, and J. L. Walsh. Field testing for cosmic ray soft errors in semiconductor memories. *IBM J. of R&D*, 40(1):41–50, Jan. 1996.

[18] C. Pernet. Calcul du polynôme caractéristique sur des corps finis. Master's thesis, U. J. Fourier, June 2003. `www-lmc.imag.fr/lmc-mosaic/Clement.Pernet`.

[19] C. Pernet and Z. Wan. L U based algorithms for characteristic polynomial over a finite field. *SIGSAM Bull.*, 37(3):83–84, 2003. Poster available at `www-lmc.imag.fr/lmc-mosaic/Clement.Pernet`.

[20] A. Steel. Personnal communication. 2005

[21] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, Institut für Wissenschaftliches Rechnen, ETH-Zentrum, Zürich, Switzerland, Nov. 2000.

[22] A. Storjohann. Computing the frobenius form of a sparse integer matrix. Manuscript, Apr. 2000.

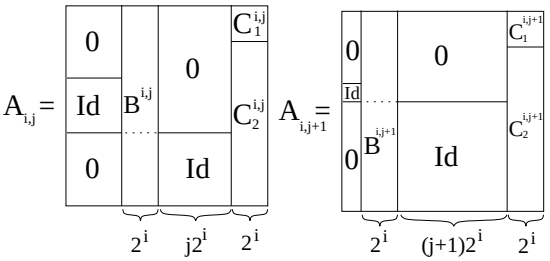
[23] G. Villard. Computing the Frobenius normal form of a sparse matrix. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *CASC'00*, Oct. 2000.

APPENDIX

A. ON KELLER-GEHRIG'S THIRD ALGORITHM

A.1 Principle of the algorithm

We first recall the principle of this algorithm, so as to determine the exact constant in its algebraic time complexity. This advocates for its practicability. First, let us define a m-Frobenius form as a $n \times n$ matrix of the shape: $\left[\begin{smallmatrix} 0 & M_1 \\ Id_{n-m} & M_2 \end{smallmatrix}\right]$. Note that a 1-Frobenius form is a companion matrix, whose characteristic polynomial is given by the opposites of the coefficients of its last column. The aim of the algorithm is to compute the 1-Frobenius form A_0 of A by computing the sequence of matrices $A_r = A, \ldots, A_0$, where A_i has the 2^i-Frobenius form and $r = \lceil \log n \rceil$. The idea is to compute A_i from A_{i+1} by slicing the block M of A_{i+1} into two $n \times 2^i$ columns blocks B and C. Then, similarity transformations with the matrix $U = \left[\begin{smallmatrix} 0 & C_1 \\ Id_{n-2^i} & C_2 \end{smallmatrix}\right]$ will "shift" the block B to the left and generate an identity block of size 2^i between B and C. More precisely, one computes the sequence of matrices $A_{i,0} = A_{i+1}, A_{i,1}, \ldots, A_{i,s_i} = A_i$, where $s_i = \lceil n/2^i \rceil - 1$, by the relation $A_{i,j+1} = U_{i,j}^{-1} A_{i,j} U_{i,j}$, with the following notations :



As long as C_1 is invertible, the process will carry on, and make at last the block B disappear from the matrix. This last condition is restricting. This is why this algorithm is only valid for generic matrices.

A.2 Proof of lemma 3.1

We will denote by $X_{a\ldots b}$ the submatrix composed by the rows from a to b of the block X. For a given i, `KG3` performs $n/2^i$ similarity transformations. Each one of them can be described by the following operations:

1: $B'_{n-2^i+1\ldots n} = C^{-1}_{1\ldots 2^i} B_{1\ldots 2^i}$
2: $B'_{1\ldots n-2^i} = -C_{2^i+1\ldots n} B'_{n-2^i+1\ldots n} + B_{2^i+1\ldots n}$
3: $C' = B' C_{\lambda+1\ldots\lambda+2^i}$
4: $C'_{2^i+1\ldots 2^i+\lambda} += C_{1\ldots\lambda}$
5: $C'_{2^i+\lambda+1\ldots n} += C_{2^i+\lambda+1\ldots n}$

The first operation is a system resolution, and consists in a LUP factorization and two triangular system solve with matrix right hand side. The two following ones are matrix multiplications, and we do not consider the two last ones, since their cost is dominated by the previous ones. The cost of a similarity transformation is then: $T_{i,j} = T_{\text{LUP}}(2^i, 2^i) + 2T_{\text{TRSM}}(2^i, 2^i) + T_{\text{MM}}(n-2^i, 2^i, 2^i) + T_{\text{MM}}(n, 2^i, 2^i)$. Thus, we have $T_{\text{LUP}}(m,n) = \frac{C_\omega}{2^{\omega-1}-2} m^{\omega-1} \left(n - m\frac{2^{\omega-2}-1}{2^{\omega-1}-1}\right)$ (from [5, Lemma 4.1] and [18]) and $T_{\text{TRSM}}(2^i, 2^i) = \frac{C_\omega m n^{\omega-1}}{2\left(2^{\omega-1}-1\right)}$. Therefore $T_{i,j} = \frac{C_\omega 2^{\omega-2}}{2\left(2^{\omega-2}-1\right)\left(2^{\omega-1}-1\right)}(2^i)^\omega + \frac{C_\omega}{\left(2^{\omega-2}-1\right)}(2^i)^\omega + C_\omega(n-2^i)(2^i)^{\omega-1} + C_\omega n (2^i)^{\omega-1}$. Rewrite this as $T_{i,j} = C_\omega (2^i)^\omega \underbrace{\left(\frac{2^{\omega-3}+2^{\omega-1}-1}{\left(2^{\omega-2}-1\right)\left(2^{\omega-1}-1\right)} - 1\right)}_{D_\omega} + 2nC_\omega(2^i)^{\omega-1}$ and, as the total cost is $T = \sum_{i=1}^{log(n/2)} \sum_{j=1}^{n/2^i-1} T_{i,j}$, it becomes $T = \sum_{i=1}^{log(n/2)} \left(\frac{n}{2^i} - 1\right) C_\omega D_\omega (2^i)^\omega + 2nC_\omega(2^i)^{\omega-1}$. Simplifying, we obtain $T = C_\omega \sum_{i=1}^{log(n/2)} (D_\omega - 3) n (2^i)^{\omega-1} + 2n^2(2^i)^{\omega-2} - D_\omega (2^i)^\omega$ and since $\sum_{i=1}^{log(n/2)} (2^i)^x = \frac{n^x-1}{2^x-1} = \frac{n^x}{2^x-1} + o(n^x)$ this ends the proof.