Laboratoire de Modélisation et Calcul IMAG, BP 53,
51 av. des Mathématiques.
38041 Grenoble, France.

LU based algorithms for the characteristic polynomial over a finite field

by

Clément Pernet and Zhendong Wan
Clement.Pernet@imag.fr, Wan@cis.udel.edu

11th June 2003

Department of Computer and Information Science,
University of Delaware,
Newark, DE 19716 USA.

1 Introduction

Linear algebra computation over finite fields is important because many symbolic computation problems directly or indirectly rely on linear algebra using efficient finite field representations.

The use of block algorithms for linear algebra organizes algorithms to use matrix multiplications, with the two following consequences:

Theoretical: Reduce the time complexity in terms of $M(n) = n^\omega$

Practical: Allows us to use the well-tuned BLAS level 3 Matrix-Multiply routine. Although BLAS are designed for numerical computations, it was demonstrated in [1] that a wrapping of BLAS for finite fields is also possible and worthy.

In this poster, we will focus on the characteristic polynomial computation for dense matrices over finite fields. Two algorithms will be presented. Both are based on a block LU factorization algorithm, which is the LSP factorization of [3]. This algorithm will be presented in section 2.

They are also based on a new minimal polynomial computation algorithm presented in section 3.

Section 4 and 5 are dedicated to each algorithm for the CharPoly, and section 6 gives a comparison based on timings experiments.

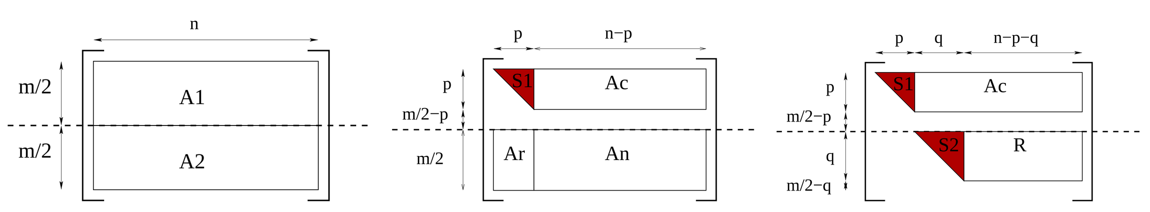
2 `LUdivine`: a block LSP factorization

The LSP factorization algorithm was introduced in [3]. LSP is a generalization of LUP for singular matrices: S is semi-upper triangular (after removing the zero rows, it is upper triangular with unit diagonal).

Indeed, the algorithm described in section 2.1 can be viewed as a block LUP algorithm for non singular matrices. But singular matrices can make zero rows appear in the upper block, making S_1 non invertible. One could remove these zero rows by doing expensive row permutations. The idea of the LSP factorization is to avoid these permutations, and consider S_1 as implicitely triangular.

In some cases, the matrix can be singular of rank $k < n$, but having its first k rows linearly independent. Then, no zero rows will occur during the factorization and LSP can be replaced by LUP. This will be the case for our `Minpoly` algorithm decribed in section 3.

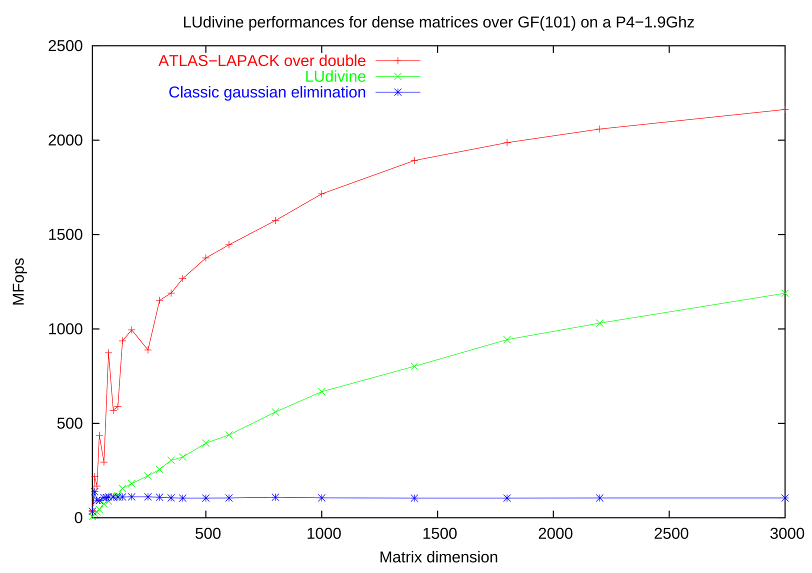
2.1 Algorithm: LSP(A)



1. The matrix is split in the row dimension
2. A recursive call computes $A_1 = L_1 \cdot S_1 \cdot P_1$
3. Update A_2: $A_2 \leftarrow A_2 \cdot P_1^t$
4. Update A_r: $A_r \leftarrow A_r \cdot S_1^{-1}$
5. Update A_n: $A_n \leftarrow A_n - A_r \cdot A_c$
6. A recursive call computes $A_n = L_2 \cdot S_2 \cdot P_2$
7. Update A_c: $A_c \leftarrow A_c \cdot P_2^t$

Only two routines are involved: Matrix Multiplication and Matrix Triangular system solve. Now triangular system solve can be reduced to Matrix Multiplications. As a consequence the time complexity is $O(m^{\omega-1}n)$.

2.2 Performances



This figure shows that `LUdivine` is faster than a classic Gaussian elimination, for matrices of order greater than 100, with an improvement factor of 10 for $n = 3000$. Numerical BLAS remains twice as fast because of the cost of back and forth conversions between finite field and `double`.

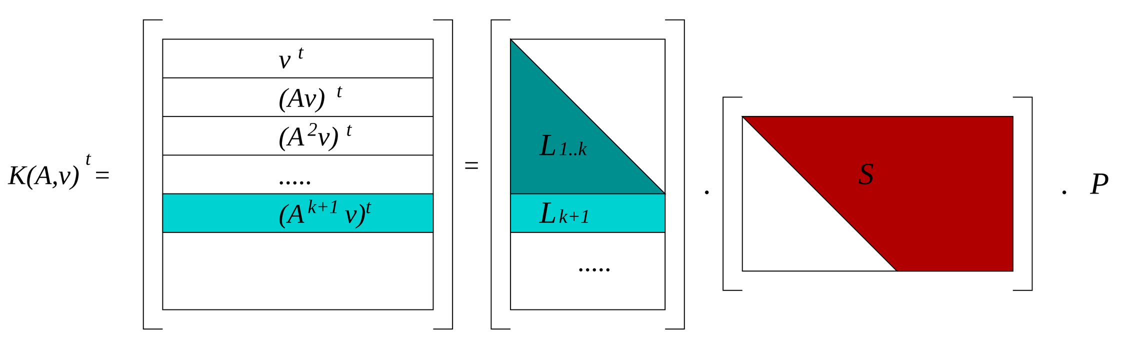
3 The Minimal Polynomial of (A,v)

3.1 Definitions

A is a $n \times n$ matrix, v a vector

- $MinPoly(A, v) = P$, Polynomial of lowest degree such that $P(A).v = 0$
- Krylov Matrix $K(A, v)$: $K(A, v) = (v, Av, A^2v, \ldots, A^nv)$

3.2 Algorithm: Minpoly(A,v)



1. Compute $X = K(A, v)^t$ — $O(n^\omega.log(n))$
2. Compute $(L, S, P) = LSP(X)$ (also gives $k := Rank(S)$) — $O(n^\omega)$
3. $m = (m_0, \ldots, m_k) := L_{k+1}.L_{1..k}^{-1}$ — $O(n^2)$
4. return $X^k - m_kX^{k-1} - \cdots - m_1X - m_0$

3.3 Sketch of proof

The construction of X implies that its first k rows (defining the submatrix $X_{1...k}$ are linearly independent, and X_{k+1} is a linear combination of them:

$$X_{k+1} = (A^kv)^t = \sum_{i=0}^{k-1} m_i(A^iv)^t = m \cdot X_{1...k}$$

where $Minpoly(A, v)(X) = X^k - m_kX^{k-1} - \cdots - m_1X - m_0$.

Thus

$$L_{k+1}SP = m \cdot L_{1...k}SP$$

And finally $m = L_{k+1}.L_{1...k}^{-1}$

3.4 Notes

- The time complexity of this algorithm is $O(n^\omega log(n))$. If the Krylov matrix K is already computed it is reduced to $O(n^\omega)$, and even $O(n^2)$ if K is already factorized in LSP form.
- Choosing randomly v for $K(A, v)$ makes the algorithm Monte-Carlo for MinPoly(A).
- LSP can be replaced by LUP since the first k rows are independent.

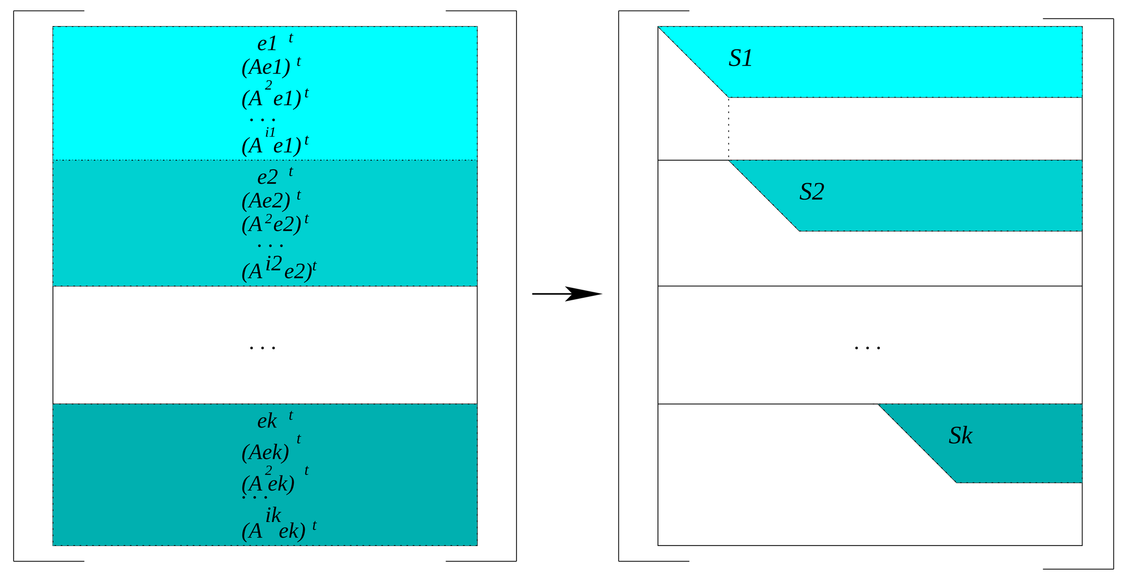
4 CharPoly: Keller-Gehrig meets Ibara

In [4], Keller-Gehrig gives an $O(n^\omega log(n))$ algorithm for CharPoly.

4.1 Principle

- Construct $U = (e_1, Ae_1, \ldots, A^{k_1}e_1, e2, Ae_2, \ldots)$ a basis of F^n where $I_n = (e_1, \ldots, e_n)$. This involves matrix multiplications and a so-called step-form elimination routine, designed for this application.
- $U^{-1}AU$ is block upper triangular, and the diagonal blocks are companion matrices. Then, get charpoly as the product the charpoly of these companion blocks.

4.2 Our improvements:



- Replace the step-form elimination of X by $LSP(X^t)$.
- Get each factor by our algorithm: $Minpoly(A, e_i)$. The LSP factorization of each $K(A, e_i)$ is already available, since it was computed to find the linear dependencies with LSP. Therefore, the expensive part of $Minpoly(A, e_i)$ is avoided.

This means that we replaced the complex step form elimination of Keller-Gehrig by the simpler LSP factorization of Ibarra & al. Then, we have a straightforward method for finding the factors: apply only the third step of the $Minpoly(A, v)$ algorithm (of section 3), which is a triangular system solve. The first two steps have already been computed.

5 CharPoly: Krylov-LU method

5.1 Principle:

Krylov methods, exposed in [2], are based on similarity of the matrix A. The first step is to reduce A to an upper block triangular matrix

$$U^{-1}AU = \begin{bmatrix} F & Y \\ 0 & A_2 \end{bmatrix}$$

where F is the companion matrix of $MinPoly(A, v)$. Calling recursively the algorithm on A_2 makes the matrix block upper triangular. Its diagonal is only composed of companion blocks. Then the characteristic polynomial is the product of the companion block polynomials.

However, the above method is a global outline. Let us specify some steps in more detail: how to get the $MinPoly(A, v)$, how to find the similarity transform to compute A_2...

Several algorithms, such as Danilevskii's, Samuelson's and others presented in [2], can be viewed as variants of Krylov's method. Their time complexity is $O(n^3)$. We propose another $O(n^3)$ algorithm, based on Krylov's method, and close to Danilevskii's algorithm. Whereas Danilevskii's algorithm uses elementary Gauss-Jordan elimination matrices, we use the LU factorization to transform the matrix into a block triangular form. Although it does not provide any theoretical improvement of the time complexity, the use of blocks makes it of practical interest because level 3 BLAS can be used.

5.2 Our algorithm

1. Pick a random vector v
2. Compute $X = K(A, v)^t$
3. Compute $(L, S, P) = LSP(X^t)$
4. Compute $M(x) = Minpoly(A, v)$ with the MinPoly algorithm of section 3
5. if $(deg(M) = n)$ then return $CharPoly(A) = M(x)$
6. else
7. Compute $A' = PA^tP^t = \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix}$ where A_{11} is $k \times k$.
8. return $Charpoly(A) = M(x)CharPoly(A'_{22} - A'_{21}S_1^{-1}S_2)$

5.3 Sketch of proof

The LSP factorization computes:

$$X = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} [S_1|S_2]P$$

Let us deduce the invertible matrix

$$\overline{X} = \underbrace{\begin{bmatrix} L_1 & 0 \\ 0 & I_{n-k} \end{bmatrix}}_{\overline{L}} \underbrace{\begin{bmatrix} S_1 & S_2 \\ 0 & I_{n-k} \end{bmatrix}}_{\overline{S}} P = \begin{bmatrix} X_{1..k} \\ \left[\, 0 \;\; I_{n-k} \,\right] \end{bmatrix} P$$

Now $X_{1..k}A^t = C^tX_{1..k}$ where C is the companion matrix associated to the minimal polynomial of A. Therefore

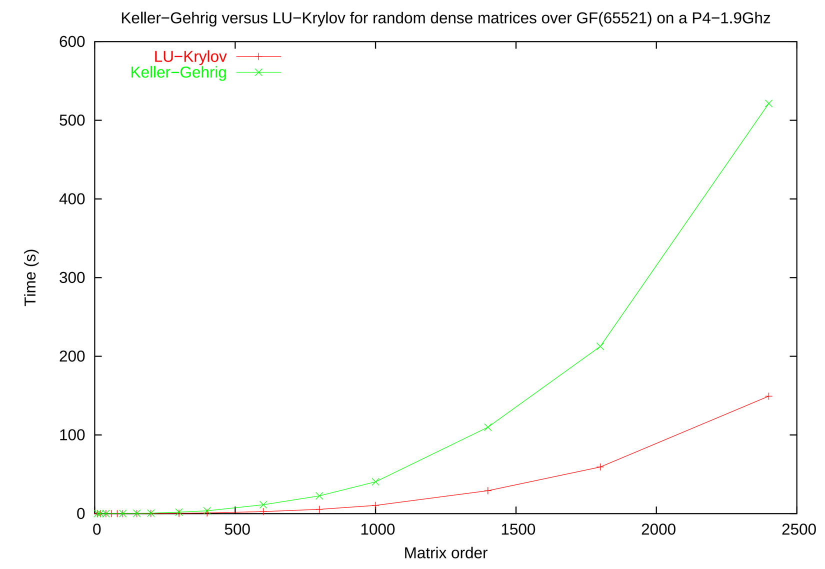
$$\overline{X}A^t\overline{X}^{-1} = \begin{bmatrix} C^t & 0 \\ \multicolumn{2}{c}{\left[\, 0 \;\; I_{n-k} \,\right] PA^tP^t\overline{S}^{-1}\overline{L}^{-1}} \end{bmatrix} = \begin{bmatrix} C^t & 0 \\ \multicolumn{2}{c}{\left[\, A'_{21} \;\; A'_{22} \,\right] \overline{S}^{-1}\overline{L}^{-1}} \end{bmatrix} = \begin{bmatrix} C^t & 0 \\ Y & X_2 \end{bmatrix}$$

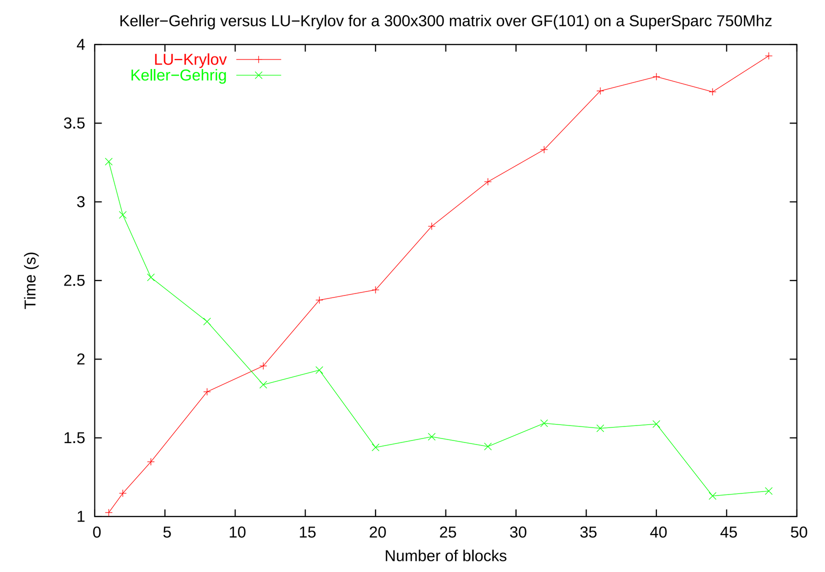
where $X_2 = A'_{22} - A'_{21}S_1^{-1}S_2$

6 Experiments: Krylov-LU vs. Keller-Gehrig and Ibara

For our experiments, we implemented these algorithms in C++ routines, using a templated field representation, and the standard BLAS interface. We used the modular finite field representation of `Givaro` and the portable BLAS `ATLAS`.

The memory complexity of our implementation is $48n^2$ bytes for `Keller-Gehrig` algorithm, and $32n^2$ bytes for `LU-Krylov` (a field element is represented on 4 bytes and the BLAS routines are working on `double` having 8 bytes). This sets the limits respectively $n < 4729$ and $n < 5792$ for `Keller-Gehrig` and `LU-Krylov`, when working with 1Gb of RAM.





The first figure compares the timing of the routines called over random dense matrices. For these matrices, the characteristic polynomial equals the minimal polynomial. The `Krylov-LU` algorithm only computes one LSP factorization for the minimal polynomial, whereas the `Keller-Gehrig` algorithm needs $log(n)$ LSP factorizations. This explains why `Krylov-LU` algorithm is faster.

In the second figure, we compare the timing for a given order, but depending on the number of invariant factors of the entry matrix. For a small number of blocks, `Krylov-LU` remains faster, but after a cross-over point, `Keller-Gehrig` is the faster one. The reason for this is that `Krylov-LU` performs as many LSP factorization as the number of blocks, whereas `Keller-Gehrig` needs at most $log(n)$ of them. More precisely, the number of LSP factorizations for the `Keller-Gehrig` algorithm is $log(k)$ where k is the order of the biggest invariant factor. This explains why the computation time is decreasing with the number of blocks, for a fixed matrix order.

From these comparisons, it appears that neither algorithm is always better. The user must choose between them, depending on the number of invariant factors of the Froebenius form. If this information is not available, these algorithms could be combined into a hybrid one: `Krylov-LU` is used for the computation of the first block. Then if the degree of the corresponding minimal polynomial is low (ie. under a given threshold), `Keller-Gehrig` is used for the rest of the computation.

7 Conclusion

We have presented two algorithms for the computation of the characteristic polynomial over a finite field. They are both based on existing ideas, respectively developed by Krylov and Danilevskii, and by Keller-Gehrig.

Our contribution was to show that the introduction of the LSP factorization of Ibarra & al. as a basic tool for these algorithms was not only possible, but also led to interesting improvements : time complexity remains the same but the use of block algorithms provides high practical efficiency and algorithms become simpler from the programmer's point of view. We have also presented a new algorithm for the minimal polynomial of dense matrices.

The two algorithms are complementary and the number of invariant factors of the entry matrix determines which one should be used: `Krylov-LU` for a few blocks, and `Keller-Gehrig` for many blocks. Therefore a hybrid algorithm can be deduced for the case of a matrix whose structure is unknown.

These routines have been integrated in a specific package designed for efficient dense linear algebra over finite fields: FFLAS-FFLAP [1]. They are also available within the Linbox package designed for blackbox linear algebra [2].

It now remains to deal with the case of large sparse matrices, for which our algorithms are too expensive in memory. This will enable anyone interested in the computation of the characteristic polynomial of dense or sparse matrices over finite fields to benefit from high performance.

References

[1] J.-G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In T. Mora, editor, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, Lille, France*. ACM Press, New York, July 2002.

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

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

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

[1] www-lmc.imag.fr/Jean-Guillaume.Dumas/FFLAS
[2] www.linalg.org