Algorithms Seminar 2001–2002,
F. Chyzak (ed.), INRIA, (2003), pp. 29–32.

Available online at the URL
http://algo.inria.fr/seminars/.

Computation of the Inverse and Determinant of a Matrix

Gilles Villard
CNRS and LIP, ENS Lyon (France)

May 27, 2002

Summary by Emmanuel Thomé

Abstract

We investigate here the complexity of different computational problems related to linear algebra, under several models. We see how these complexities are related to each other, and how in most cases they can be shown to be very closely related to the compexity of matrix multiplication.

1. Introduction

Most algorithms advertised here are of probabilistic nature, and the reader must be aware of this fact. More precisely, no distinction will be made between Monte Carlo- or Las Vegas-type probabilistic algorithms. Our major concern is that these algorithms are usable in practice (implementations have been made for most of them). When comparing algorithms, we are only interested in the major contribution to the complexity. We shall ignore constants as well as logarithmic factors in complexity estimates. The notation $O\tilde{}(x)$ will be used to reflect this consideration. Furthermore, the use of fast Fourier transform arithmetic is always assumed when applicable.

1.1. **Definitions.** We fix notation for the complexities of different computations that can be done on matrices of size n over a domain R (which will be either a Euclidean domain or a finite field), where the meaning of "complexity" depends on the computational model that will be chosen later on:

- $\mathsf{DET}(n)$: computing the determinant of a matrix of size n;
- $\mathsf{INV}(n)$: computing the inverse (when it exists) of a matrix of size n;
- $\mathsf{LINSYS}(n)$: solving a linear system of n equations with n unknowns;
- $\mathsf{MM}(n)$: multiplying two matrices of size n.

As the complexity for multiplying two matrices expressed in terms of arithmetic operations is crucial to many respects, it will appear in several places. We will use w for the exponent associated to this problem. Currently, the algorithm of Coppersmith–Winograd for matrix multiplication has the lowest aymptotical estimate, $O\big(n^w\big)$ with $w \approx 2.38$.

2. Classical Complexity Results under Different Models

2.1. **Algebraic model.** Here we assume that R is a field (therefore division is possible) and we express complexities in terms of arithmetic operations in R. The following facts are known:

Proposition 1 (Strassen, 1969)**.** $\mathsf{DET}(n) \prec \mathsf{MM}(n)$ and $\mathsf{LINSYS}(n) \prec \mathsf{MM}(n)$.

Proposition 2 (Strassen, 1973, Baur & Strassen, 1983)**.** $\mathsf{MM}(n) \prec \mathsf{DET}(n)$.

An easy consequence of this fact is that under the arithmetic model, the complexities of DET and MM are the same. Using the asymptotically fastest matrix multiplication algorithms, a complexity of n^w is therefore possible. Furthermore, the results above also imply that LINSYS(n) can be solved in at most the same complexity. It is not known, however, whether one can do better than n^w for LINSYS(n), and this has been an open problem for soon thirty years.

2.2. **Bit complexity.** All bit operations are equally counted under that model. For instance, this is the relevant model when computations are carried over the integers. The size of the input becomes important beyond the only consideration of the dimension of the matrix. The length of the coefficients is important too, therefore we introduce $\|A\|$, the norm of A, as the biggest of its coefficients (in absolute value). The determinant of an $n \times n$ matrix A is an integer of size $O\big(n \log \|A\|\big)$ and can be computed in time:

$$\mathsf{DET}(n) = \tilde{O}\big(n^{w+1} \log \|A\|\big).$$

The above complexity result is obtained by using the Chinese remainder theorem: the result is evaluated modulo a sufficiently large set of primes, and then recovered by interpolation.

As for linear systems, we have a much better result, since LINSYS(n) can be solved in complexity $\tilde{O}\big(n^w \log^2 \|A\|\big)$ using p-adic (Hensel) lifting. This result is extensively explained in Storjohann's thesis [6] as well as in [7].

2.3. **Division-free complexity.** The complexity results given in the algebraic model above assume that division is allowed and the results of Strassen quoted above do make use of this fact. What happens if we remove the possibility of computing inverses? For instance, if R is a Euclidean domain and not a field, inverses cannot be computed whereas the determinant is well-defined.

The following trick might help to carry the results concerning the arithmetic model to the division-free model, with an impact on the complexity. Suppose for instance that we want to compute the determinant of the matrix A over the domain R. The idea is to work in $R[[u]]$ (the ring of formal power series over R), setting $B(u) = I + u(A - I)$. Therefore $B(0) = I$ and $B(1) = A$. The computation of the determinant of B using the recursive factorization algorithm of Strassen is possible, and requires only the computation of inverses of elements in $1 + uR[[u]]$. Since

$$\frac{1}{1-z} = 1 + z + z^2 + \ldots = (1+z)(1+z^2)(1+z^4)\ldots,$$

it is possible to compute this quantity without inverting elements of R. Since the result obtained is necessarily a polynomial in $R[u]$ of degree n, we can afford to carry computations only modulo u^{n+1}. Final evaluation of this polynomial at $u = 1$ yields $\det A$. The consequence of this is that we obtain a complexity of $\tilde{O}\big(n^{w+1}\big)$ for computing a determinant in the division-free model.

2.4. **Black-box (sparse) complexity.** We say that we do black-box computations with a given matrix when no investigation is made on the inner structure of the matrix and the matrix is merely used in one single operation: multiplication by a vector. In theory, this operation requires $O\big(n^2\big)$ multiplications, but when the matrix has the property of being *sparse* (as few as $O(\log n)$ non-zero coefficients per row), then the cost can be for instance $\tilde{O}(n)$.

Under this model, the computation of the determinant or the characteristic polynomial is done in $\tilde{O}\big(n^w\big)$, which is no better than the arithmetic complexity. However, the computation of the minimal polynomial, and therefore the solution of a linear system, can be obtained much more efficiently. Algorithms gathered under the name of "Krylov subspace techniques" are quite successful for this purpose. The Lanczos algorithm can be regarded as the Gram–Schmidt process for finding

a self-orthogonal vector. The Wiedemann algorithm is another method that can be quickly described. Choose two random vectors u and v. Then compute the minimum generating polynomial of the scalar sequence $v^T u$, $v^T Au$, ..., $v^T A^k u$, ... With high probability (on u, v, and A), this polynomial is also the minimum polynomial of the sequence I, A, ..., A^k, ..., which is also the minimum polynomial of A. Since the minimum generating polynomial of a scalar sequence can be computed in time $O(n^2)$ using the Berlekamp–Massey algorithm, using the first $2n$ coefficients of the sequence, it follows that the minimum polynomial can be computed in time $O^{\sim}(n^2)$.

We will see later that in fact the relative easiness of the computation of the minimum polynomial can be well understood in terms of the invariant factors of the matrix, of which the minimum polynomial is the biggest, while the characteristic polynomial is their product.

2.5. **Overview of classical costs.** If we summarize these results, focusing only on the dominating term and taking the size of the inputs constant, we obtain:

	Arithmetic	Division free	Binary	Sparse
LINSYS	n^w	–	n^w	n^2
DET	n^w	n^{w+1}	n^{w+1}	n^w

3. Recent Progress (2000–2002)

Recent works in the field have contributed to the following improvements:

	Division free	Binary (general)	Binary (polynomial)	Sparse
LINSYS	–	n^w	$n^w d$	n^2
DET	n^{w+1} ↓ $n^{2.698}$ [4, 5]	n^{w+1} ↓ n^w [2, 4, 7]	$n^{w+1}d$ ↓ $n^3 d$ (INV) [3]	n^w ↓ $n^{2.25}$ [8, 10]

Two examples will be presented here.

3.1. **Computing the inverse of a polynomial matrix.** Given a polynomial $n \times n$ matrix A of degree d over a field K, we present here an algorithm that computes the inverse of A in time $O^{\sim}(n^3 d)$, which is in fact the size of the output.

It should be noted that a closely related result has been obtained recently by Storjohann [7], who shows that the determinant of A can be computed in time $O^{\sim}(n^w d)$. It is not known, however, whether the computation of the inverse reduces to the computation of the determinant. On the other hand, our algorithm can be derived to obtain another method for computing the determinant (although not as efficient as Storjohann's).

The polynomial inverse computation algorithm described here is a block divide-and-conquer procedure. If the input matrix has the form $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$, assuming that the input is generic, we proceed through the following steps for computing the left inverse of the matrix:

1. Compute cofactors $U(X)$ and $V(X)$ such that the equality $U(X)A(X) + V(X)C(X) = 0$ holds. This can be achieved by a matrix Euclidean algorithm, using [1] or [9] for instance, in time $O^{\sim}(n^3 d)$. With generic input, the degrees of $U(X)$ and $V(X)$ are less than or equal to d.
2. Similarly, obtain cofactors $S(X)$ and $T(X)$ such that we have $S(X)B(X) + T(X)D(X) = 0$.
3. Multiply the input matrix on the left by $\left(\begin{smallmatrix} S & T \\ U & V \end{smallmatrix}\right)$. The resulting matrix is block-diagonal, and the blocks have degree $2d$ at most.
4. Compute the inverse of the blocks on the diagonal recursively.

All of these steps can be performed in time $O^{\sim}(n^3 d)$. There are two recursive calls for the inversion of matrices on the diagonal. These are inverses of $\frac{n}{2} \times \frac{n}{2}$ matrices of degree $2d$. The resulting complexity can therefore easily be shown to be equal to $O^{\sim}(n^3 d)$.

We have assumed that the degrees of $U(X)$ and $V(X)$ were balanced, which is true for most inputs, but might fail for some particular matrices. In such a case, it is possible to precondition (multiply on the left by some matrix whose inverse is known) the input matrix so that the required conditions are met.

3.2. **Computing the characteristic polynomial and invariant factors of a sparse matrix.** If A is a sparse matrix, we have seen already that its minimal polynomial can be easily computed in time $O^{\sim}\left(n^2\right)$. Using this, we will now see that much more can be obtained. We define the invariant factors of A (denoted $f_1, \ldots, f_n$) to be the coefficients of the diagonal on the Smith normal form $S(X)$ of the polynomial matrix $A - XI$. Recall that the Smith normal form is the unique diagonal matrix such that $S(X) = U(X)(A - XI)V(X)$, where U and V have determinant 1, and the coefficients on the diagonal divide each other.

Obviously, the characteristic polynomial of A is the product of its invariant factors. Since these invariant factors divide each other, it is easy to see that at most $\sqrt{n}$ of them are distinct. We show that it is possible to discover the chain of distinct invariant factors and their multiplicities by a divide-and-conquer procedure, with at most $O\left(\sqrt{n}\right)$ computations of individual invariant factors. The crux of this algorithm is the ability to compute f_k just as easily as we are already able to find the minimal polynomial f_n. To achieve this, we simply compute the gcd of f_n with the minimal polynomials of $A+B$, where B is a matrix of rank $n-k$. This gives the desired result. The matrix B can even be chosen to be a product of two Toeplitz matrices, which eases the computations. The design of the recursive discovery procedure is then straightforward. This yields a total complexity of $O^{\sim}\left(n^{2.5}\right)$. The latter can be lowered to $O^{\sim}\left(n^{2.25}\right)$ using block techniques.

Bibliography

[1] Beckermann (Bernhard) and Labahn (George). – A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM Journal on Matrix Analysis and Applications*, vol. 15, n° 3, 1994, pp. 804–823.

[2] Eberly (Wayne), Giesbrecht (Mark), and Villard (Gilles). – On computing the determinant and Smith form of an integer matrix. In *41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000)*, pp. 675–685. – IEEE Computing Society, Los Alamitos, CA, 2000.

[3] Jeannerod (Claude-Pierre) and Villard (Gilles). – Computing the inverse of a polynomial matrix. – 2002. In preparation.

[4] Kaltofen (Erich) and Villard (Gilles). – On the complexity of computing determinants (extended abstract). In *Computer mathematics (Matsuyama, 2001)*, pp. 13–27. – World Scientific, River Edge, NJ, 2001.

[5] Kaltofen92 (Erich). – On computing determinants without divisions. In Wang (Paul S.) (editor), *International Symposium on Symbolic and Algebraic Computation 92*. pp. 342–349. – ACM Press, New York, 1992. Proceedings of ISSAC'92, Berkeley, California (July 27-29, 1992).

[6] Storjohann (Arne). – *Algorithms for matrix canonical forms*. – Dissertation, Swiss Federal Institute of Technology, December 2000. Diss. ETH No. 13922.

[7] Storjohann (Arne). – High-order lifting [extended abstract]. In Mora (Teo) (editor), *ISSAC 2002 (July 7-10, 2002. Université de Lille, Lille, France)*. pp. 246–254. – ACM Press, New York, 2002. Conference proceedings.

[8] Storjohann (Arne) and Villard (Gilles). – Computing the characteristic polynomial of a sparse matrix. – 2002. In preparation.

[9] Thomé (Emmanuel). – Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *Journal of Symbolic Computation*, vol. 33, n° 5, 2002, pp. 757–775. – Computer algebra (London, ON, 2001).

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