OPEN ACCESS

GRS Journal of Multidisciplinary Research and Studies

Abbriviate Tittle- GRS J Mul Res Stud

ISSN *(Online)- 3049-0561*

https://grspublisher.com/journal-details/GRSJ

Vol-2, Iss-4 (Apr- 2025)

Advances in Dense Linear Algebra: From Gauss to Strassen

Moumouni Djassibo Woba[1], Zoungrana Amidou[2], Zongo Moumouni[3],

[1]Université Lédéa Bernard OUEDRAOGO (B.F)

[2,3]Université Norbert ZONGO, (B.F)

***Corresponding Author :Moumouni Djassibo Woba**

"Université Lédéa Bernard OUEDRAOGO (B.F)"

Received: 20.02.2025 **Accepted: 28.03.2025** **Published: 01.04.2025**

Abstract: Linear algebra lies at the core of many algorithmic problems. Standard matrix multiplication and the Gaussian elimination method have arithmetic complexity that is cubic in terms of input size. In this article, we show that for most dense linear algebra problems—multiplication, inversion, determinant, system solving—more efficient algorithms exist with strictly sub-cubic complexity.

Keywords: *Algorithms, matrix, Linear algebra, Multiplication, Determinant.*

Introduction

In mathematics, it is customary to consider a problem trivial once it has been reduced to a question of linear algebra. However, from a computational perspective, the efficiency of matrix operations remains a critical concern.

The algorithmic challenges in linear algebra hide very subtle difficulties. Generally, the first questions encountered when working with matrices concern the efficient computation of matrix-matrix products, matrix-vector products, inverses, and system solving. The answers to these questions vary significantly depending on the type of matrices involved. A possible classification is as follows:

- Dense Matrices: These are arbitrary matrices with no particular structure. We will see that dense matrix algorithms can essentially be reduced to matrix multiplication, whose complexity is an extremely delicate question.
- Sparse Matrices: Many linear problems are formulated in terms of matrices with a large number of zero entries, called sparse matrices. In such cases, dense algorithms are inappropriate; it is possible to exploit sparsity using better-suited tools based on linear recurrences.[1]
- Structured Matrices: There exist specific families of matrices often as- sociated with linear mappings between polynomial spaces, such as Sylvester- type matrices for resultants, Vandermonde-type matrices for evaluation and interpolation, Toeplitz and Hankel-type matrices for Padé and Padé–Hermite approximations, etc. For these well-identified types of matrices, either ad-hoc algorithms or a unified theory based on the concept of displacement rank are developed.

Schematically, dense matrix algorithms are the slowest, with complexity ranging between $O(n^2)$ and $O(n^3)$ for size n

The complexity of computations involving sparse matrices is approximately $O(n^2)$, while that of structured matrices is $O(n)$, up to logarithmic factors.

The focus of this article is on dense matrix algorithms. We will work over an effective field denoted by $\mathbb{K}$ and with the algebra of square matrices $Mn(\mathbb{K})$ with coefficients in $\mathbb{K}$. Note, however, that most results in this article extend to the case where the field $\mathbb{K}$ is replaced by an effective ring A, and square matrices are replaced by rectangular matrices with coefficients in A.

The questions addressed or mentioned in this article include the complexity of computing matrix multiplication in $Mn(\mathbb{K})$, inverses, determinants, characteristic or minimal polynomials, solving linear systems, and putting matrices into various canonical forms (row-echelon form, LUP decomposition, block companion form, etc.).

For these questions, we have a first family of naive algorithms based on direct application of definitions or mathematical properties. While the naive algorithm for multiplication has reasonable cubic complexity, in other cases the use of naive algorithms is strongly discouraged. For example, if $A = (a_{i,j})_{i,j=1}^{n}$ is a matrix in $Mn(\mathbb{K})$, we would not use the definition

$$et(A) = \sum_{\sigma \in \mathrm{Sn}} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

to compute its determinant; indeed, this would lead to an algorithm with complexity $O(n.n!)$, which is hyper-exponential in the size of A. Similarly, defining the rank of a matrix as the size of the largest non-zero minor does not lend itself directly to efficient algorithmization via exhaustive search. In a slightly different vein, Cramer's formulas for solving a system. Classical linear algebra algorithms are not of great practical utility (however, they serve to bound solution sizes and are often useful in complexity estimates). A second class of algorithms, based on Gaussian elimination, allows most of the above-mentioned problems to be solved reasonably efficiently. Applied to a matrix A in $Mn(\mathbb{K})$, this method provides algorithms with $O(n^3)$ complexity for computing the rank $rg(A)$, the determinant $det(A)$, the inverse A^{-1} (if A is invertible), row-echelon form, LUP decomposition of A, and solving a linear system with matrix A.

The main result of this article is that we can do better, and there exists a constant $2 \leq \omega < 3$ (dependent a priori on the base field

$\mathbb{K}$) that governs the complexity of matrix multiplication and all linear algebra operations.

To formalize this point, we associate an exponent with each problem. In the case of multiplication, it is defined as follows:

Definition

A real number θ is said to be a feasible exponent for matrix multiplication over the field $\mathbb{K}$ if there exists an algorithm with arithmetic complexity $O(n^{\theta})$ to multiply two arbitrary matrices in $Mn(\mathbb{K})$. We define $\omega_{\mathrm{mul}} = inf\{\theta \mid \theta$ is a feasible exponent$\}$.

A priori, we should include the field $\mathbb{K}$ as an index for the exponents θ and ω_{mul}. However, the results presented hereafter are independent of $\mathbb{K}$. The naive algorithm for multiplication has complexity $O(n^3)$, so $\theta = 3$ is a feasible exponent and $\omega_{\mathrm{mul}} \leq 3$. On the other hand, ω_{mul} must be at least 2: the number of elements in a matrix is n^2, and at least that many operations are needed to write it down.

We can similarly define exponents for all other problems, such as ω_{inv} for inversion, ω_{polcar} for the characteristic polynomial, ω_{det} for the determinant, ω_{LUP} for LUP decomposition, etc. We then have the following result.[2]

Theorem

The exponents ω_{mul}, ω_{inv}, ω_{polcar}, ω_{det} , and $\omega_{\mathrm{LUP}}$are all equal and strictly less than 2.38; we denote by ω their common value. The exponent corresponding to solving linear systems is less than or equal to ω.

This theorem contains two types of results:

- Proofs of equivalence between problems.
- Upper bounds on their complexity, i.e., algorithms.[3]

We will show that $\omega_{\mathrm{det}} \leq \omega_{\mathrm{polcar}} \leq \omega_{\mathrm{mul}} = \omega_{\mathrm{inv}}$ and $\omega_{\mathrm{inv}} \leq \omega_{\mathrm{det}}$, which allows us to conclude that this chain of inequalities is actually a chain of equalities.

Applications

The linear algebra problems discussed above are encountered frequently in practice; the results of the Theorem will be invoked repeatedly in this article. For example, the Beckermann–Labahn algorithm for computing Padé–Hermite approximants ultimately relies on multiplying polynomial matrices. The same applies to Storjohann's algorithm for solving polynomial linear systems and the algorithm for finding series solutions of algebraic or linear differential equations.

Many polynomial system-solving algorithms also rely on linear algebra: Gröbner basis computation methods can be reduced to large sparse linear systems (eventually); the complexity results we provide for Gröbner basis computations rely on fast row-echelon form computations; the geometric resolution algorithm uses a Newton iteration involving the inversion of formal power series matrices.[4].

In a different context, polynomial factorization algorithms, algorithms for differentially finite series, and summation/integration algorithms all involve solving linear systems as a subproblem. Linear algebra lies at the heart of many other algorithmic problems not covered in this work, such as integer factorization or discrete logarithm in cryptanalysis.

Finally, note that a large portion of the world's computers spend their cycles performing linear algebra computations, whether for combinatorial optimization (linear programming via the simplex algorithm), numerical simulation (finite element methods), or web page ranking (Google's PageRank system, which relies on finding an eigenvector of a massive sparse matrix).

Matrix multiplication

Like integer and polynomial multiplication, matrix multiplication is a funda- mental operation in computer algebra, whose complexity analysis turns out to be highly non-trivial.

The naive algorithm for multiplying an $m \times n$ matrix by an n-dimensional vector uses $O(mn)$ arithmetic operations: mn multiplications and $m(n-1)$ additions. A result by Winograd states that in general, we cannot do better: for real matrices with algebraically independent coefficients over $\mathbb{Q}$, the naive algorithm is optimal. Since matrix multiplication can be interpreted as a sequence of matrix-vector products, a natural question is whether the naive matrix multiplication is also nearly optimal.

The answer to this question, along with the most important results of this section (listed chronologically), are compiled in the following theorem.

Theorem

The naive matrix multiplication is not optimal. Let $\mathbb{K}$ be a field. Two matrices in $Mn(\mathbb{K})$ can be multiplied using:

1. $O(n^3)$ operations in $\mathbb{K}$, via the naive algorithm.
2. $\left\lceil n^2 \frac{n^2}{2} \right\rceil + 2n\left\lceil \frac{n^2}{2} \right\rceil \approx \frac{1}{2}n^3 + n^2$ multiplications in K.
3. $\left\lceil n^2 \frac{n^2}{2} \right\rceil + (2n - 1)\left\lceil \frac{n^2}{2} \right\rceil \approx \frac{1}{2}n^3 + n^2$ multiplications in $\mathbb{K}$ if the characteristic of $\mathbb{K}$ is not 2 and division by 2 is free.
4. $O(n^{log_2 7}) \approx O(n^{2.81})$ operations in $\mathbb{K}$.[5].

These results hold over a commutative ring A; for (1) and (4), the commutativity assumption can even be dropped. Part (2) is due to Winograd and was historically the first improvement over the naive algorithm. It halves the number of multiplications but increases the number of additions. This constitutes progress for a wide class of rings A where multiplication is more costly than addition.

For example, the binary splitting technique requires multiplying matricescontaining large integers.

The improvement (3) was obtained by Waksman, showing that two 2×2 matrices can be multiplied in 7 multiplications. The common drawback of Wino- grads and Waksman's algorithms is their use of $\mathbb{K}$'s commutativity, which be- comes an obstacle to recursive application, preventing further improvement on the upper bound of ω_{mul}. Many believed that results like (2) and (3) were optimal, in the sense that $\frac{1}{2}n^3$ multiplications in $\mathbb{K}$ would be required for multiplication in $Mn(\mathbb{K})$. This was disproven by Strassen's discovery of (4), which follows from a surprisingly simple yet far-reaching insight: two 2×2 matrices can be multiplied in 7 multiplications even if $\mathbb{K}$ is non-commutative.

The non-optimality of the naive matrix multiplication justifies introducing the following definition.

Definition

An application $MM : \mathbb{N} \to \mathbb{N}$ is called a matrix multiplication function (for a field $\mathbb{K}$) if:

- Two arbitrary matrices in $Mn(\mathbb{K})$can be multiplied using at most $MM\ (n)$operations in $\mathbb{K}$.
- MM is non-decreasing and satisfies the inequality $MM\ (n/2) \leq \frac{MM(n)}{4}$ for all $n \in \mathbb{N}$.

Thus, Theorem 3.1 shows that functions like $n \mapsto n^3$ or $n \mapsto n^{log_2 7}$ (up to constant factors) are matrix multiplication functions. Similar to the case of polynomial multiplication functions M.

This concept is useful as it allows stating complexity results independent of the choice of matrix multiplication algorithm

The second condition ensures consistency: if $MM\ (n) = c.n^{\theta}$ for Some constant $c > 0$, then θ must be at least 2. [6]

Matrix multiplication

This condition will be used to show that in divide-and-conquer matrix algorithms, the total cost is essentially dominated by the final recursive call.

Naive multiplication

The cubic multiplication algorithm takes as input two matrices $A = (a_{i,j})_{1\leq i,j\leq n}$ and $X = (x_{i,j})_{1\leq i,j\leq n}$ in $Mn(\mathbb{K})$, and computes their product $R = AX$ using the formula:

$$r_{i,k} = \sum_{j=1}^{n} a_{i,j}x_{j,k} \qquad (1)$$

The cost to compute a single entry of $\mathbb{R}$ is n multiplications and $n -$ 1 additions, leading to an overall complexity of $O(n^3)$. [7].

Winograd's algorithm

It is possible to approximately halve the number of scalar multiplications re- quired to compute the product AX. This demonstrates that the naive matrix multiplication is not optimal. Assume for simplicity that n is even, $n = 2k$. Winograd's algorithm relies on the following identity, inherited from Karatsuba's method: if ℓ/c is a row vector $(a_1, \dots, a_n)$ and c is a column $vector\ {}^t(x_1, \dots, x_n)$, then the dot product $(\ell\ |c) = \sum_i a_i x_i$ can be written as:

$$(\ell|c) = (a_1 + x_2)(a_2 + x_1) + \dots + (a_{2k-1} + x_{2k})(a_{2k} + x_{2k-1}) - \sigma(\ell) - \sigma(c), \qquad (2)$$

where $\sigma(\ell) = a_1a_2 + \ ...\ + a_{2k-1}a_{2k}$ and

$\sigma(c) = x_1x_2 + \dots + x_{2k-1}x_{2k}$. Note that $\sigma(\ell)$ can be computed in $k = n/2$ multiplications and $k - 1 = n/2 - 1$ additions.[8]

If $\ell_1, \dots, \ell_n$ are the rows of A and $C_1, \dots, C_n$ are the columns of X, then the (i, j) entry of the product AX is $(\ell_i|c_j)$. The idea is to precompute the quantities $\sigma(\ell_i)$ and $\sigma(c_i)$ for $1 \leq i \leq n$; this precomputation requires:

$n\left(\frac{n}{2}+\frac{n}{2}\right) = n^2$ multiplications and $n\left(\frac{n}{2} - 1 + \frac{n}{2} - 1\right) = n^2$additions.

Once the $\sigma(\ell_i)$and $\sigma(c_i)$ are computed, all elements $r_{i,j}$ of

$R = AX$ Can be calculated as: $\frac{1}{2}n^2.n^2 = \frac{1}{2}\ n^3$ multiplications and

$n^2.\left(\ n + \left(\frac{n}{2} - 1\right) + 2\right) = \frac{3}{2}n^3 + n^2$ addition.

In total, the cost of Winograd's matrix multiplication algorithm is $\frac{1}{2}n^3 + n^2$multiplications and $\frac{3}{2}n^3 + 2n^2 - 2n$ additions.

Thus, we have effectively converted $\frac{3}{2}n^3$ multiplications into additions, which is a significant improvement.

Waksman's Algorithm

To compute the product of 2×2 matrices:

$$R = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x & y \\ z & t \end{pmatrix}$$

The above Winograd algorithm uses 8 multiplications, writing:

$$R = \begin{pmatrix} (a + z)(b + x) - ab - zx & (a + t)(b + y) - ab - ty \\ (c + z)(d + x) - cd - zx & (c + t)(d + y) - cd - ty \end{pmatrix}$$

Thus, it provides no improvement over the naive algorithm for this size. The following algorithm, due to Waksman, can be seen as an improvement over Winograd's algorithm. It performs the 2×2 matrix multiplication in 7 operations, provided that in the base field $\mathbb{K}$, the element 2 is invertible and division by 2 is free. The idea is to write the matrix $\mathbb{R}$ as:

$$\mathcal{R} = \frac{1}{2}\begin{pmatrix} \alpha_1 & - & \alpha_2 & \beta_1 & - & \beta_2 \\ \gamma_1 & - & \gamma_2 & \delta_1 & - & \delta_2 \end{pmatrix} \qquad (4)$$

Where

$$\begin{aligned} \alpha_1 &= (a + z)(b + x), \\ \alpha_2 &= (a - z)(b - x), \\ \beta_1 &= (a + t)(b + y), \\ \beta_2 &= (a - t)(b - y), \\ \gamma_1 &= (c + z)(d + x), \\ \gamma_2 &= (c - z)(d - x), \\ \delta_1 &= (c + t)(d + y), \\ \delta_2 &= (c - t)(d - y), \end{aligned}$$

Furthermore, thanks to the identity

$$\begin{aligned} (\gamma_1 + \gamma_2) + (\beta_1 + \beta_2) = (\alpha_1 + \alpha_2) + (\delta_1 + \delta_2) \\ = 2(ab + cd + xz + ty), \end{aligned}$$

The element δ_2 is a linear combination of the other seven products $\alpha_1, \alpha_2, \dots, \delta_1$, and thus does not need to be obtained via an additional multiplication.[9]

The algorithm derived from this, like Winograd's algorithm, cannot be used recursively: the proof of the underlying formulas relies on the commutativity of elements in the base field $\mathbb{K}\,(e.g., bz = zb, bt = tb, etc.)$. During recursive calls, the objects to be multiplied would themselves be matrices, for which the commutativity property no longer holds. Thus, Winograd's and Waksman's algorithms are termed commutative algorithms.

Strassen's algorithm

Strassen's algorithm was the first to break below $O(n^3)$. Like Karatsuba's algorithm for polynomials, it leverages a reduction in the number of multiplications for 2×2 matrices, which translates to an exponent improvement during recursive application. However, matrix multiplication's non-commutativity imposes an additional constraint: for recursive use, the new algorithm (unlike Waks-man's) must satisfy a non-commutativity condition.

Let A and X be 2×2 matrices to be multiplied using a non-commutative algorithm. Intuitively, this means that during the multiplications performed by such an algorithm, the elements a, b, c, d of A must remain on the left, and the elements x, y, z, t of X on the right (this is not the case for Winograd's and Waksman's algorithms, which mix elements of A and X).

The naive algorithm is indeed non-commutative but requires 8 multiplications. The Strassen algorithm we present reduces this to:

- Computing linear combinations of the elements of A and X,

Input: Two matrices $A, X \in Mn(\mathbb{K})$, where n = 2^k. Output: The product AX

1. If $n = 1$, return AX.
2. Decompose

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ and } X = \begin{pmatrix} x & y \\ z & t \end{pmatrix} \qquad (5\,)$$

Where a, b, c, d and x, y, z, t are in $\frac{Mn}{2(\mathbb{K})}$.

3. Recursively compute the products

q1 = a(x + z),

q2 = d(y + t),

q3 = (d − a)(z − y),

q4 = (b − d)(z + t),

q5 = (b − a)z,

q6 = (c − a)(x + y),

q7 = (c − d)y.

4. Compute the sums

r1,1 = q1 + q5,

r1,2 = q2 + q3 + q4 − q5,

r2,1 = q1 + q3 + q6 − q7,

r2,2 = q2 + q7.

5. Return

$$\begin{pmatrix} r_{1,1} & r_{1,2} \\ r_{2,1} & r_{2,2} \end{pmatrix}$$

Strassen's Algorithm for Multiplying Two Matrices

- Perform 7 products of these linear combinations
- Recombine the results to obtain the four elements of the product AX

Explicitly, the formulas to apply are given in steps (3) and (4). Let's proceed recursively. By padding with zero rows and columns, if necessary, we can assume the matrices A and X are of size $n = 2^k$ for some $k \in \mathbb{N}$. We perform a block decomposition of A and$X$:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \text{ and } X = \begin{pmatrix} x & y \\ z & t \end{pmatrix}$$

In this notation, a, b, c, d and x, y, z, t are square matrices of size $\frac{n}{2}$. The key insight is this: thanks to non-commutativity, the formulas given for the 2×2 case still apply if a, b, c, d and x, y, z, t are matrices. Thus, they reduce the product of size n to 7 products of size $\frac{n}{2}$, plus a certain number C (here $C = 18$) of additions of size $\frac{n}{2}$, used to form linear combinations of the blocks of A and X, and of the products q_i.

The complexity $S(n)$ of Strassen's algorithm satisfies the recurrence

$$S(n) \leq 7S\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right)^2 \qquad (6)$$

By invoking the divide-and-conquer lemma with parameters m = 7, p = s = 2, κ = 1, and q = 4, we derive the inequality:

$$S(n) \leq \left(1 + \frac{C}{3}\right) n^{log_2 7}$$

The core idea of Strassen's algorithm is entirely general: improving multi-plication for small sizes improves the exponent.

Interpretation of Strassen's formulas

It is not immediately intuitive to provide insight into Strassen's formulas (unlike Karatsuba's, which implicitly relies on polynomial evaluation at $0, 1, +\infty$). We present an argument, due to Fiduccia and postdating the algorithm's discovery, to motivate its design.

The starting point is the observation that finding a non-commutative algo-rithm to compute the matrix product $R = AX$ for $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $X = \begin{pmatrix} x & y \\ z & t \end{pmatrix}$

Reduces to finding an algorithm for matrix-vector multiplication Mv, where

$$M = \begin{pmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{pmatrix} \text{ and } v = \begin{pmatrix} x \\ z \\ y \\ t \end{pmatrix} \qquad (7)$$

The continuation of the argument relies on the following observation: any matrix (of arbitrary size) of the form

$$\begin{pmatrix} \alpha & \alpha \\ \in \alpha & \in \alpha \end{pmatrix}, \begin{pmatrix} \alpha & -\alpha \\ \in \alpha & -\in \alpha \end{pmatrix} \; or \; \begin{pmatrix} \alpha & \alpha \\ -\alpha & -\in \alpha \end{pmatrix}$$

Where $\in \{0, 1\}$ and $\alpha \in \mathbb{K}$, with all elements except the four explicitly marked being zero, can be multiplied by a vector in a single multiplication in $\mathbb{K}$. Such a matrix is called elementary.

The idea is then to decompose the matrix M into a sum of 7 elementary matrices. We start by subtracting from M the two elementary matrices.

$$E_1 = \begin{pmatrix} a & a \\ a & a \end{pmatrix} \; and \; E_2 = \begin{pmatrix} d & d \\ d & d \end{pmatrix}$$

Corresponding to the diagonal elements of A. The resulting matrix is

$$M_1 = \begin{pmatrix} b - a & c - a \\ d - a & a - d \\ b - d & c - d \end{pmatrix}$$

Strassen himself does not remember the exact origin of his formulas. according to Landsberg," Strassen was attempting to prove, by process of elimination, that such an algorithm did not exist when he arrived at it." In any case, Strassen claims," First I had realized that an estimate of tensor rank 8 for two-by-two matrix multiplication would give an asymptotically faster algorithm. Then I worked over $\mathbb{Z}/2\mathbb{Z}$ — as far as I remember — to simplify matters." In other words, there must have been manual search involved.[10]

By construction, this matrix has the property that the (1, 1) element of the second block is the negative of the (2, 2) element of the first. By subtracting the elementary matrix

$$E_2 = \begin{pmatrix} d - a & a - d \\ d - a & a - d \end{pmatrix}$$

from M_1, we obtain the matrix

$M_2 = \begin{pmatrix} b-a & c-a \\ d-a & a-d \\ b-d & c-d \end{pmatrix}$

Which decomposes as $M_3 + M_4$, with

$M_3 = \begin{pmatrix} b-a & \\ a-d & b-d \end{pmatrix}$ And $M_4 = \begin{pmatrix} c-a & d-a \\ c-d & \end{pmatrix}$ (8)

Since $a - d = (b-d) - (b-a)$ and $d - a = (c-a) - (c-d)$, each of the matrices M_3 and M_4 can be written as the sum of 2 elementary matrices:

$M_3 = E_4 + E_S$, where $E_4 = \begin{pmatrix} b-a \\ a-b \end{pmatrix}$, $E_s = (b-d \quad b-d)$ and

$M_4 = E_6 + E_7$, where $E_6 = (c-a \quad c-a)$, $E_7 = \begin{pmatrix} d-c \\ c-d \end{pmatrix}$

Returning to the equality

$$\begin{pmatrix} r_{1,1} & r_{1,2} \\ r_{2,1} & r_{2,2} \end{pmatrix} = \sum_{i=1}^{7} E_i . v$$

Expressing$R = AX$, we finally obtain the formulas given in steps (3) and (4)

Can we do better than 2.81?

Since Strassen proved that the exponent ω_{mul} for matrix multiplication is sub- cubic, the natural question became determining its true value. To this end, since the 1970s, numerous tools have been developed-- increasingly sophisticated—and ingenuity has been abundant. New concepts such as bilinear complexity, tensor rank, and approximate algorithms have been extensively used, notably by Bini, Pan, Schonhage, Strassen, Winograd, and others. These concepts will not be covered in this work.[11]

The best current algorithm, from 2014 and due to François Le Gall, has complexity $O(n^{2.3728639})$. However, it is conjectured that $\omega_{\text{mul}} = 2$, but proving (or refuting) this conjecture remains out of reach for now.

In practice

Fast algorithms for linear algebra have long had a poor reputation; concretely, Strassen's algorithm—and, to a lesser extent, Pan's 2.77-exponent algorithm (adopted by Kaporin)—are the only fast algorithms (with exponents below 3) that have been successfully implemented.

In a good implementation, say over a finite field defined modulo a reasonable-sized integer, Winograd's and Waksman's algorithms are immediately efficient. Strassen's algorithm then becomes better for sizes on the order of a few dozen (64 is a reasonable threshold).

Most other known algorithms rely on highly complex techniques, which result in enormous constants (and logarithmic factors) in their complexity estimates. Consequently, in their current forms, they cannot be efficient for sizes below millions or billions.

Other linear algebra problems

It is important to multiply matrices, but it is even more useful to invert them, compute their characteristic polynomials, etc.

- **Gaussian elimination**

Recall that the classical Gaussian elimination algorithm solves most classical linear algebra problems in a much more efficient manner than naive algorithms.

- **Definition**

--Row-Echelon Form. A matrix is in row-echelon form if:

i. All zero rows are below non-zero rows,
ii. The first non-zero coefficient (called the pivot) of each non-zero row is strictly to the right of the first non-zero coefficient of the preceding row.[12]

A row-echelon form of a matrix *A* is a row-echelon matrix *B* left-equivalent to A (i.e., there exists an invertible square matrix *P* such that *A* = *P B*).

The Gaussian algorithm (without column pivoting) computes, using only elementary row operations (corresponding to left multiplications by invertible matrices), a row-echelon form of *A*, on which the rank and determinant of *A* can be directly read. Indeed, the non-zero rows of a row-echelon matrix *B* form a basis for the vector space spanned by the rows of the original matrix A. The number of non-zero rows of *B* equals the rank of *A*. If *A* is square, its determinant equals the product of the diagonal entries of *B*.

A variant of the Gaussian algorithm, called Gauss-Jordan elimination, produces a reduced row-echelon form of A: this is a row-echelon form where the pivots are 1, and all other entries in the pivot columns are zero. Applied to the augmented matrix A˜ = (A|I_n) formed by concatenating A with the identity matrix In, this variant is used to compute the inverse A^{-1} . Indeed, the Gauss Jordan algorithm transforms $A^{\sim}$ into an equivalent matrix whose left block is the identity, i.e., it replaces A˜ with ($I_n|A^{-1}$).

The same algorithm can solve the system Ax = b, where b ∈ K_n, by augmenting matrix A with the vector b. Another variant of Gaussian elimination computes an LUP decomposition, which in turn enables solving linear systems, inverting matrices, computing determinants, etc.

- **Definition**

— Matrices L, U, P, and LUP Decomposition. A matrix is called upper triangular, respectively lower triangular, if all entries below (resp. above) the main diagonal are zero. A permutation matrix is a square matrix P with entries 0 or 1, where each row and column contains exactly one non-zero entry. An LU decomposition, resp. LUP decomposition, of a matrix A is a factorization of the form A = LU, resp. A = LUP, where L is lower triangular, U is upper triangular, and P is a permutation matrix.

In summary, we assume the following statement holds.

- **Theorem**

— Gaussian Elimination. For any matrix A ∈ Mn(K), the following can be computed in O(n 3) operations in K:

i. The rank rg(A), the determinant det(A), and the inverse A−1 if A is invertible;
ii. A basis (affine) of solutions for Ax = b for any b ∈ Kn;
iii. An LUP decomposition and a reduced row-echelon form of *A*

Main result

- **Theorem**

— Gaussian Elimination is not Optimal. Let $\mathbb{K}$ be a field and let θ be a feasible exponent for matrix multiplication over $\mathbb{K}$. For any matrix $A \in Mn(\mathbb{K})$, the following can be computed:

i. The determinant $det(A)$;

ii. The inverse A^{-1} (if A is invertible),
iii. The solution $x \in \mathbb{K}^n$ of the system $Ax = b$ for any $b \in \mathbb{K}^n$, if A is invertible;

- **Theorem**

— For any matrix A ∈ Mn(K), the following can be computed in O(ñ^θ) operations in K:

i. The determinant $det(A)$;
ii. The inverse A^{-1} (if AA is invertible);
iii. The solution $x \in \mathbb{K}^n$ of the system $Ax = b$ for any $b \in \mathbb{K}^n$, if A is invertible;
iv. The characteristic polynomial of A;
v. $An\ LUP$ decomposition of A.
vi. The rank $rg(A)$ and a reduced row-echelon form of A;
vii. A basis of the kernel of A.

Multiplication is no harder than inversion

Let A and B be $n \times n$ matrices. We wish to compute $C = AB$. For this, we define

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix} \qquad (9)$$

We then have the identity

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}$$

which allows us to reduce the product of size n to the inversion of size $3n$. This proves our assertion and the inequality$\omega_{mul} \leq \omega_{inv.}$

Inversion, determinant calculation, and system solv-Ing are no harder than matrix multiplication

We will show that it is possible to invert $n \times n$ matrices using a divide-and- conquer algorithm with $O(\tilde{n}^{\theta})$ arithmetic operations, for any feasible exponent θ.

- **Matrix inversion**

The matrix inversion algorithm presented here, due to Strassen, is a recursive algorithm that can be interpreted as block Gaussian elimination. This algorithm requires inverting certain submatrices of A to ensure correctness.

Strassen's algorithm for inverting a matrix

We assume for simplicity that all these matrices are invertible. The general case (arbitrary matrix A) is more delicate and requires an efficient divide-and- conquer-based computation of an LUP decomposition of A.

The starting point is the following non-commutative identity, where we sup-pose $n = 2$ and a,$Z \in \mathbb{K} \setminus \{0\}$:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \times \begin{pmatrix} a & 0 \\ 0 & Z \end{pmatrix} \times \begin{pmatrix} 1 & a^{-1}b \\ 0 & 1 \end{pmatrix}$$

Where $Z = d - ca^{-1}b$ is the Schur complement of a in A.

This identity is easily derived using Gaussian elimination on A and allows obtaining the following factorization:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \begin{pmatrix} 1 & a^{-1}b \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} a^{-1} & 0 \\ 0 & Z^{-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \qquad (10)$$

The key point is that the identity is non-commutative: it remains valid in any non-commutative ring, provided the elements a and $Z = d - ca^{-1}b$ are invertible. This allows recursive application. In each recursive call, the algorithm performs two inversions and several multiplications. Specifically, using the notation of Algorith, the six products ce, bt, $ceb = (ce)b, ebt = e(bt), (ebt)(ce),$ and $t(ce)$ suffice. This analysis leads to the following result.

- **Theorem**

— If all encountered submatrices are invertible, the cost of inverting A is 3 MM (n) + O(n^2).

- **Proof**

The inversion algorithm satisfies the recurrence:

$$I(n) \leq 2I\left(\frac{n}{2}\right) + 6MM\left(\frac{n}{2}\right) + Cn^2$$

Where C is a constant. Applying the divide-and-conquer lemma with parameters $m = p = s = 2, \kappa = 1,$ and $q = 4$ suffices.

An immediate consequence of the Theorem is that solving the linear system $Ax = b$, for $b \in \mathbb{K}^n$ and invertible A satisfying the theorem's hypotheses, can also be done in $O(MM\,(n)) = O(n^{\theta})$ operations in $\mathbb{K}$.

Determinant Calculation

The factorization shows that a slight adaptation of the inversion algorithm al- lows computing the determinant of A simultaneously with its inverse, at the same complexity. To do this, it suffices to replace steps (1), (3), (5), and (7) of Algorithm 8.2 with steps (1'), (3'), (5'), and (7') described below:

1' If $n = 1$, return A^{-1} and A.
3' Compute $e := a^{-1}$ and $d_a := det(a)$ recursively.
5' Compute $t := Z^{-1}$ and $d_z := det(Z)$ recursively
7' Return $\begin{pmatrix} x & y \\ z & t \end{pmatrix}$ and $d_a d_z$

Weakening the Hypotheses

At this stage, we have proven parts (a)– (c) of the Theorem under an additional hypothesis. When K is a subfield of the real numbers ℝ, this hypothesis can be removed by noting that, for any invertible matrix A in $Mn(\mathbb{R})$, the matrix $B = tA.A$ is positive definite and satisfies the theorem's hypotheses. Thus, the inverse of A can be computed via $A^{-1} = B \cdot tA$ in $O(MM\,(n)) = O(n^{\theta})$ operations in $\mathbb{K}$.

It is not difficult to show that if matrix A is of type L (lower triangular) or U (upper triangular), Strassen's algorithm computes its inverse without requiring additional hypotheses.

This observation forms the basis of a general inversion algorithm due to Bunch and Hopcroft.

This algorithm first computes an LUP decomposition of an arbitrary invertible matrix over any field K in $O(MM\,(n))$ operations and uses it to compute the inverse (and solve systems) at the same cost.

- **Solving overdetermined systems**

When a matrix is not square, computing its kernel reduces easily to the square case.

i. **Remark**

— Invertibility detection is also achievable in $O(MM\,(n))$ operations using an algorithm that computes the rank of A with this complexity; this algorithm is beyond our scope.

ii. Theorem

Let K be a field, and suppose there is a matrix multiplication algorithm in Mn(K) with complexity MM (n) = O(n^(θ)),θ ≥ 2. The computation of a basis for the kernel of an m × n matrix A (with m ≥ n) over K costs O(m˜n^(θ-1)) operations in 𝕂.

iii. Proof

The Theorem implies this lemma if $m = n$. Thus, we may assume m is a multiple of n by adding zero rows to matrix A. Let A_1 denote the matrix of the first m rows of A, A_2 the next m rows, etc.

We can compute a basis for the kernel of A_1 with a cost of $O(\tilde{n}^{\theta})$: let N_1 be the matrix whose columns form this basis, so that $A_1N_1 = 0$. We then compute the matrix N_2 whose columns form a basis for the kernel of A_1N_1, such that $A_2N_1N_2 = 0$. Continuing similarly with A_3, we compute N_3 such that $A_3N_1N_2N_3 = 0$, etc. Finally, the columns of $N_1 \cdots N_{m/n}$ form a basis for the kernel of A. This yields the total cost $O(\tilde{m}n^{\theta-1})$as claimed.

iv. Characteristic polynomial calculation

Let A be an $n \times n$ matrix over 𝕂. In this section, we present an efficient algorithm due to Keller-Gehrig for computing the characteristic polynomial $\chi_A(X) = \det(XI_n - A)$ of A. We will detail a specific simple case (and common scenario). Specifically, we assume throughout that $\chi_A(X) = X_n + p_{n-1}X_{n-1} + \cdots + p_0$ is irreducible in $\mathbb{K}[X]$; in particular, it coincides with the minimal polynomial of A. The algorithm crucially relies on the following lemma.

v. Lemma

— Let A be an $n \times n$ matrix over 𝕂, whose characteristic polynomial is irreducible in $\mathbb{K}[X]$.

Let v be a non-zero vector in Kn, and let $P \in M_n(K)$ be the matrix whose j-th column is $A^{j-1}v$ for $1 \leq j \leq n$. Then P is invertible, and the matrix P −1AP is of companion type. Companion Matrix : A matrix $C = (c_{i,j})_{1\leq i,j\leq n}$ is called a companion matrix if its first n − 1 columns contain only zeros except for the entries $c_{2,1}, c_{3,2}, \ldots, c_{n,n-1}$cn,n−1, which all equal 1. The key advantage of Lemma 8.7 is that the characteristic polynomial of a companion matrix C is $X^n - \sum_{i=1}^{n} C_{i,n}X^{i-1}$, requiring no arithmetic operations to compute.

vi. Proof

To prove this lemma, we first observe that, due to the hypothesis on A, the family $B = \{v_1, v_2, \ldots, v_n\}$, where $v_i = A^{i-1}v$, forms a basis for the 𝕂-vector space $\mathbb{K}^n$, hence P is invertible. Indeed, assuming the contrary would imply the existence of a non-zero polynomial $Q \in \mathbb{K}[X]$ of degree less than n such that $Q(A)v = 0$. Without loss of generality, assume Q has minimal degree with this property. Since $\chi_A(A)v = 0, Q$ divides χ_A; but since χ_A is irreducible, this implies Q is constant, leading to $v = 0$, a contradiction.

Since $Av_{i-1} = v_i$, the matrix C representing the linear map $w \to Aw$ in basis B is a companion matrix. The change-of-basis theorem then gives $= P^{-1}AP$, concluding the proof.

- **Characteristic polynomial calculation**

Input: A matrix $A \in Mn(\mathbb{K})$ with $n = 2k$.

Output: Its characteristic polynomial $\det(XI_n - A)$.

```
i.   Choose a non-zero vector $v \in \mathbb{K}^n$.
ii.  M := A;  P := v.
iii. For $i$ from 1 to $k$, set $P$ to the horizontal concatenation of
     $P$ and $MP$ ,then set $M := M^2$.
iv.  $C := P^{-1}AP$.
v.   Return $Xn + pn - 1Xn - 1 + \cdots + p_0$ ,$where\ t =$
     $(-p_0, \ldots, -p_{n-1})$ is thelast column of C.
```

Algorithm — keller-gehrig's characteristic polynomial algorithm

Since matrices A and $C = P^{-1}AP$ are similar, they share the same characteristic polynomial. The key idea of Keller-Gehrig's algorithm is to construct the matrix P from Lemma, then determine C via multiplication and inversion, and finally read the polynomial coefficients from the last column of C.

In terms of complexity, the only costly step is constructing matrix . The naive approach of computing the Krylov sequence $v, Av, \ldots, A_{n-1}v$ via successive vector-matrix multiplications costs cubic time. A crucial insight is to group these vector-matrix products into matrix-matrix multiplications:

Compute in $O(n^2)$ operations the vectors v and Av, then compute the binary exponentiation sequence $A, A2, A4, A8$,... in $O(log\ n)$ steps. Finally, compute the columns of P via products like $A2 \times (v|Av) = (A2v|A3v), then\ A4 \times (v|\cdots|A3v) = (A4v|\cdots|A7v)$, etc. Each such product is performed using an $n \times n$ matrix multiplication, with artificial zero columns added to the right-hand factors.

This method yields the algorithm above. We have just proven the following weaker version of point (d):

- **Theorem**

— Let 𝕂 be a field, and suppose there exists a matrix multiplication algorithm in $Mn(\mathbb{K})$ with complexity $MM\ (n) = O(n^{\theta}), \theta \geq 2$. If the characteristic polynomial of a matrix $A \in Mn(\mathbb{K})$ is irreducible in $\mathbb{K}[X]$, it can be computed in $O(MM\ (n) log\ n)$ operations in 𝕂.

Division-free algorithms

It is sometimes useful to compute the characteristic polynomial without per- forming divisions in the coefficient ring.

- **Theorem — Berkowitz**

The characteristic polynomial of an $n \times n$ matrix with coefficients in a commutative ring A can be computed in $O(n^{\theta+1})$ operations in A, where θ is a feasible exponent for multiplication.

However, it is straightforward to achieve this same complexity bound by assuming divisions by $2, \ldots, n$ are possible in the coefficient ring (which holds, for example, in an $\mathbb{K} - algebra$ over a field 𝕂). Indeed, the $i - th$ Newton sum of the characteristic polynomial of matrix A can be obtained as $Tr(A_i)$. It suffices to use the algorithm to compute the characteristic polynomial coefficients of A from these traces for $i = 0, \ldots, n - 1$. This is the Le Verrier method.

A different perspective was developed by Landsberg [?], who showed that open questions about matrix multiplication complexity can be formulated in geometric and representation theory terms. Finally, another active research path to prove (or disprove) that

$\omega_{mul} = 2$ was pioneered by Cohn and Umans [?]. They developed a unified approach reducing matrix multiplication to studying the discrete Fourier transform in group algebras associated with certain finite groups, which have good properties regarding their irreducible representations. Through this lens, they recovered $\omega_{mul} < 2.41$ and reduced $\omega_{mul} = 2$ to combinatorial and group-theoretic conjectures.

- **Determinant and permanent**

The permanent of a matrix $A \in Mn(\mathbb{K})$ is defined as

$$perm(A) = \sum_{\sigma \in \mathrm{Sn}} \prod_{i=1}^{n} a_i, \sigma_{(i)}. \qquad (11)$$

Despite the similarity between this definition and that of $det(A)$, no polynomial- time algorithm is known for computing perm(A) over a field $\mathbb{K}$ of characteristic different from 2. Valiant [?] proved that computing the permanent of $n \times n$ matrices with $\{0, 1\}$ entries is an NP-hard problem.

Conclusion

Dense linear algebra plays a central role in numerous algorithmic domains, from matrix multiplication to solving linear systems. Recent advancements, notably thanks to algorithms such as those by Strassen, Winograd, and Waksman, have shown that it is possible to significantly improve the complexity of matrix operations. In particular, we have established that the exponent of matrix multiplication is below 3, while conjecturing it might reach 2.

These results open the way to practical applications across diverse fields, ranging from cryptography to numerical simulation, through combinatorial optimization. However, despite these advances, many challenges remain in devel- oping efficient and robust algorithms, especially for very large matrix sizes or in non-commutative contexts.

It is crucial to continue research efforts on these algorithms, not only to enhance the performance of algebraic computations but also to explore new applications that could benefit from these advanced techniques. Future work should focus on understanding the theoretical limits of these algorithms and seeking innovative methods to tackle complex algorithmic problems. Dense linear algebra, with its multiple facets and challenges, remains a dynamic and promising field of study, essential for the development of modern technologies and computational applications.

References

1. Rota, G. C. (Ed.). (1976). *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.
2. Ben-Or, M., & Tiwari, P. (1988, January). A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (pp. 301-309).
3. Cabay, S., & Labahn, G. (1989, July). A fast, reliable algorithm for calculating Pade-Hermite forms. In *Proceedings of the ACM-SIGSAM 1989 international symposium on Symbolic and algebraic computation* (pp. 95-100).
4. Giorgi, P., Jeannerod, C. P., & Villard, G. (2003, August). On the complexity of polynomial matrix computations. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation* (pp. 135-142).
5. Hermite, C. (1873). Extrait d'une lettre de Monsieur Ch. Hermite à Monsieur Paul Gordan.
6. Hermite, C. (1874). *Sur la fonction exponentielle*. Gauthier-Villars.
7. Mahler, K. (1968). Perfect systems. *Compositio mathematica*, *19*(2), 95-166.
8. Massey, J. (1969). Shift-register synthesis and BCH decoding. *IEEE transactions on Information Theory*, *15*(1), 122-127.
9. Mills, W. H. (1975). Continued fractions and linear recurrences. *Mathematics of Computation*, *29*(129), 173-180.
10. Padé, H. (1892). Sur la représentation approchée d'une fonction par des fractions rationnelles. In *Annales scientifiques de l'Ecole normale supérieure* (Vol. 9, pp. 3-93).
11. Shafer, R. E. (1974). On quadratic approximation. *SIAM Journal on Numerical Analysis*, *11*(2), 447-460.
12. Zierler, N. (1968). Linear recurring sequences and error-correcting codes. *Error Correcting Codes (Proc. Sympos. Math. Res. Center, Madison, Wis., 1968)*, 47-59.