BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 70(2), 2022, Article number: e140535
DOI: 10.24425/bpasts.2022.140535

Matrix black box algorithms – a survey

Jerzy RESPONDEK*

The Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, ul. Akademicka 16, 44-100 Gliwice, Poland

Abstract. The implementations of matrix multiplication on contemporary, vector-oriented, and multicore-oriented computer hardware are very carefully designed and optimized with respect to their efficiency, due to the essential significance of that operation in other science and engineering domains. Consequently, the available implementations are very fast and it is a natural desire to take advantage of the efficiency of those implementations in other problems, both matrix and nonmatrix. Such an approach is often called a black box matrix computation paradigm in the literature on the subject. In this article, we gathered a broad series of algorithms taking advantage of the efficiency of fast matrix multiplication algorithms in other mathematical and computer science operations.

Key words: matrix algebra; matrix multiplication algorithm; numerical recipes; recursion.

1. INTRODUCTION

The approach of shifting the computational burden of different problems towards matrix multiplication is called black box algorithms in the literature. A common goal of this paradigm is to construct an algorithm of the same time complexity as the involved matrix multiplication algorithm. That is often feasible but in certain problems we obtain a complexity being a function of the matrix multiplication complexity, usually with a logarithmic or linear factor, which is not a bad achievement, either. In the whole article, $O(M(n))$ will stand for the time of the involved matrix multiplication algorithm as the reference point for the efficiency of the algorithms surveyed in this article.

We gathered algorithms that yield from the efficiency of fast matrix multiplication algorithms in other mathematical and computer science operations. Particularly, these are other matrix and linear algebra operations, like matrix inversion, LU decomposition, calculating the determinant, and solving a set of linear equations. Efficient matrix multiplication can also be utilized in the problems of other domains, like graphs properties analysis, grammar parsing, operating on the polynomials as well as other problems.

2. THE MATRIX MULTIPLICATION ALGORITHMS

The matrix multiplication plays a key role in this survey since other matrix operations are built on this basis. Thus the efficiency of the latter depends on the efficiency of the former.

The recursive paradigm offers great possibilities for improving the matrix multiplication efficiency. Its nonrecursive implementation achieves the time complexity $O(n^3)$, arising directly from the mathematical definition. The first attempt to improve the efficiency of the matrix multiplication was made in 1969 by V. Strassen in [1], where an algorithm with the $O(n^{2.807})$ time complexity was presented.

This article initialized a long series of works with a common general aim, i.e., to decrease the k exponent in the $O(n^k) = O(M(n))$ algorithm time complexity. Since 1969 that exponent has been decreased a number of times, to 2.78041 in 1978 [2], 2.7799 in 1979 [3], 2.548 in 1981 [4], 2.5166 in 1982 [5], 2.495548 in 1982 [6], and to 2.48 in 1986 [7].

Until 2013, the fastest matrix multiplication algorithm had been algorithm [8], with the coefficient $k = 2.376$, in the $O(n\hat{}k)$ complexity. In 2013 [9] improved the bound to 2.37369. In 2014 [10] presented an algorithm that decreased the k coefficient to 2.372873 and the final up-to-date complexity is 2.3728639, proposed also in 2014 in [11]. The progress of decreasing the k exponent in the $O(n^k)$ complexity we summarized in Fig 1.

Nonetheless, the big "O" notation shows the asymptotic growth rate but hides the constant. It is well known, not only

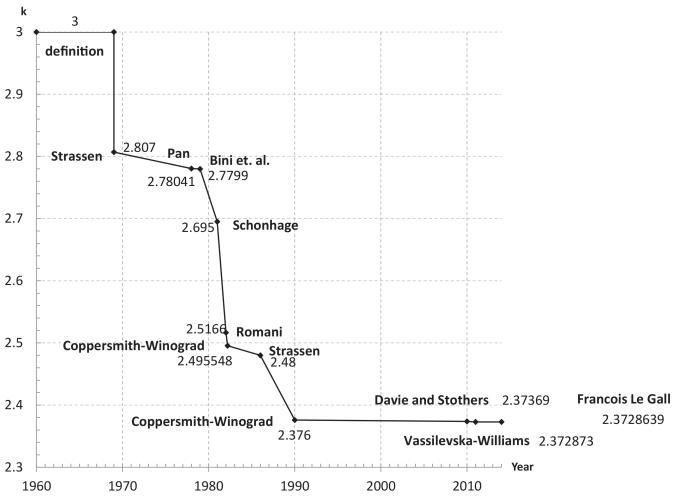

Fig. 1. Algorithms improving the efficiency of the matrix multiplication

*e-mail: jerzy.respondek@polsl.pl

Manuscript submitted 2019-07-30, revised 2019-07-30, initially accepted for publication 2020-02-24, published in April 2022.

in the matrix field, that the more sophisticated a given algorithm is, with a lower growth rate, the cost for that is just the raising constant. The Strassen's version, as well as its modified Winograd's version [12], are useful for relatively small matrices but more sophisticated algorithms, with a smaller k exponent, can compete with the standard $O(n\hat{}3)$ method only for large matrices.

Let us show by the example of the classical Strassen algorithm [1] how to improve the efficiency of the matrix multiplication. Let C denote the matrix product of matrices $[A]_{n\times n}$, $[B]_{n\times n}$ with real entries in the real (number) domain. We divide each of the matrix factors A,B into four submatrices, each with the dimensions of $(n/2)\times(n/2)$, as in the following formula (1):

$$\begin{bmatrix} C_{11} & \vdots & C_{12} \\ \cdots & \cdots & \cdots \\ C_{21} & \vdots & C_{22} \end{bmatrix}_C = \begin{bmatrix} A_{11} & \vdots & A_{12} \\ \cdots & \cdots & \cdots \\ A_{21} & \vdots & A_{22} \end{bmatrix}_A \cdot \begin{bmatrix} B_{11} & \vdots & B_{12} \\ \cdots & \cdots & \cdots \\ B_{21} & \vdots & B_{22} \end{bmatrix}_B . \tag{1}$$

The innovative idea of the Strassen algorithm consists of the following elements:

- Calculate a certain number of auxiliary expressions from submatrices, particularly with the use of matrix multiplication (but on half-sized matrices).
- Construct a linear combination of the obtained subexpression in a way that ensures that the product $C_{11},\ldots,C_{22}$ entries fulfill the mathematical definition formula.
- Apply this paradigm recursively.

Basically, the computational work in all those algorithms constructed in the general recursive way is shifted from the time-expensive matrix multiplications towards much less costly additions/subtractions and – if necessary – to transpositions.

As an auxiliary subexpression, the Strassen algorithm proposes to calculate the following 7 matrix terms $M_1,\ldots,M_7$, each one requiring exactly one matrix multiplication:

$$\begin{aligned} &M_1 = (A_{11}+A_{22})(B_{11}+B_{22}), \quad M_2 = (A_{21}+A_{22})B_{11}\,, \\ &M_3 = A_{11}(B_{12}-B_{22}), \quad M_4 = A_{22}(B_{21}-B_{11}), \\ &M_5 = (A_{11}+A_{12})B_{22}\,, \quad M_6 = (A_{21}-A_{11})(B_{11}+B_{12}). \\ &M_7 = (A_{12}-A_{22})(B_{21}+B_{22}). \end{aligned} \tag{2}$$

The formula (3) presents the final solution to the problem, i.e., the proper linear combinations:

$$\begin{bmatrix} C_{11} & \vdots & C_{12} \\ \cdots & \cdots & \cdots \\ C_{21} & \vdots & C_{22} \end{bmatrix}_C = \begin{bmatrix} M_1+M_4-M_5+M_7 & \vdots & M_3+M_5 \\ \cdots & \cdots & \cdots \\ M_2+M_4 & \vdots & M_1+M_3-M_2+M_6 \end{bmatrix}. \tag{3}$$

We leave it as an exercise for an interested reader, who may check how, with the use of the matrix terms M_* (2) it is possible to construct the matrix product, like in the formula (3).

We can see that now we need 7 half-size matrix multiplications and 18 (4.5 times more!) matrix additions/subtractions and the complexity recursive equation receives the form (4):

$$T(n) = 7T(n/2) + 18(n/2)^2. \tag{4}$$

The general recursion theorem gives the algorithm with time complexity equal to $O(n^{\log_2 7}) \approx O(n^{2.807})$ ([13] Section 4.3).

There are also some interesting theoretical analyses of the Strassen algorithm:

- [12] showed how to multiply 2×2 matrix also in 7 multiplications but decreased the number of additions/subtractions from 18 to 15.
- [14] proposed an optimal implementation of the algorithm [12] with minimal possible constant coefficient hidden in the big "O" notation.
- [15] proved that at least 7 multiplications are necessary to multiply 2×2 matrices.
- [16] showed that at least 15 additions/subtractions are necessary for a 7 multiplications algorithm.

3. RECURSIVE INVERSION OF TRIANGULAR MATRICES

Let us assume that we want to invert the following $n\times n$ size block matrix (5), divided into four sub-matrices, each with the dimensions of $(n/2)\times(n/2)$. Let $\Delta = A_{22} - A_{21}A_{11}^{-1}A_{12}$. We use the following identity ([17] p. 72, 73, [18] p. 231):

$$\begin{aligned} A &= \begin{bmatrix} A_{11} & \vdots & A_{12} \\ \cdots & \cdots & \cdots \\ A_{21} & \vdots & A_{22} \end{bmatrix}, \\ A^{-1} &= \begin{bmatrix} A_{11}^{-1}+A_{11}^{-1}A_{12}\Delta^{-1}A_{21}A_{11}^{-1} & \vdots & -A_{11}^{-1}A_{12}\Delta^{-1} \\ \cdots & \cdots & \cdots \\ -\Delta^{-1}A_{21}A_{11}^{-1} & \vdots & \Delta^{-1} \end{bmatrix}. \end{aligned} \tag{5}$$

The formula (5) is a recursive tool to invert triangular matrices by the matrix multiplication paradigm. Let us assume that we have a nonsingular upper triangular matrix, so $A_{21}=0$ and each of the $A_{\cdot}$ diagonal submatrices is nonsingular. In the block-triangular case $\Delta = A_{22}$, so Δ is invertible. The formula (5) in the upper triangular case receives quite a compact form:

$$A = \begin{bmatrix} A_{11} & \vdots & A_{12} \\ \cdots & \cdots & \cdots \\ \mathbf{0} & \vdots & A_{22} \end{bmatrix}, \quad A^{-1} = \begin{bmatrix} A_{11}^{-1} & \vdots & -A_{11}^{-1}A_{12}A_{22}^{-1} \\ \cdots & \cdots & \cdots \\ \mathbf{0} & \vdots & A_{22}^{-1} \end{bmatrix}. \tag{6}$$

On each recursion level, we must perform the following operations: 2 inversions of the $(n/2)\times(n/2)$ matrices, $2(n/2)\times(n/2)$ matrix multiplications, and $n\hat{}2/4$ negations of the sign in the right upper quarter. Thus the recursive complexity equation has the form:

$$T(n) = 2T(n/2) + 2M(n/2) + n^2/4, \tag{7}$$

where $T(n)$ is the unknown time complexity of the triangular matrix inversion and $M(n)$ is the complexity of the involved matrix multiplication algorithm (Section 2). Considering that $M(n) = O(n^{2+\varepsilon})$ for a given $\varepsilon > 0$, we can estimate $n^2/4 \leq M(n)$ and from the equation (7) it follows that:

$$T(n) \leq 2T(n/2) + 3M(n/2). \tag{8}$$

Since $M(n) = O(n^{2+\varepsilon})$, in the general recursion theorem ([13] Section 4.3) we have the case when the $f(n) = 3M(n/2)$ decides on the overall algorithm complexity and $T(n) = O(M(n))$.

4. RECURSIVE INVERSION OF MATRICES

The Strassen original inversion algorithm [1] does work on every positive-definite matrix. Let us assume that the matrix to be inverted is symmetric and positively defined, in the block form (9):

$$A = \begin{bmatrix} B & C^T \\ C & D \end{bmatrix}. \tag{9}$$

From positive definiteness of A follows the positive definiteness of B in (9), so also its invertibility. We can decompose the matrix A into the product of three-block triangular and block-diagonal matrices:

$$A = \begin{bmatrix} B & C^T \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CB^{-1} & I \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & B^{-1}C^T \\ 0 & I \end{bmatrix}, \tag{10}$$

where $S = D - CB^{-1}C^T$ is the so-called Schur residual of the matrix A with respect to B. From (10) and the positive definiteness of A follows that $\begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix} > 0$, thus S is also positive definite so is invertible, because $\det \begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix} > 0 \Leftrightarrow \det(B) \cdot \det(S) > 0 \Leftrightarrow \det(S) > 0$. Now it is easy to verify that the inversion of A has the form (11):

$$A^{-1} = \begin{bmatrix} I & -B^{-1}C^T \\ 0 & I \end{bmatrix} \begin{bmatrix} B^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CB^{-1} & I \end{bmatrix}. \tag{11}$$

We can observe that the equation (11) is the equation (5) applied to the case where A is symmetric.

The question is how to invert each nonsingular matrix by the formula (11). The universal solution is given in [19]. We invert an auxiliary matrix A^TA, which is symmetric and positive for any nonsingular matrix. Next, we can obtain the desired inverse from the following formula:

$$A^{-1} = \left(A^TA\right)^{-1}A^T. \tag{12}$$

Multiplying both sides of (12) we obtain the identity. The last issue is the time complexity of the proposed algorithm. From the formula (11) this recursive equation follows:

$$I(n) \leq 2I(n/2) + 4M(n/2) + O(n^2), \tag{13}$$

where $2I(n/2)$ is responsible for 2 inversions, $4M(n)$ is for 4 multiplications and the last term $O(n^2)$ encapsulates all the necessary additions/subtractions and transpositions. Moreover, $O(M(n)) > O(n^2)$ so we can rewrite the equation (13) in the form:

$$I(n) \leq 2I(n/2) + O(M(n)). \tag{14}$$

From the general recursion theorem ([13] Section 4.3) we have $T(n) = I(M(n))$.

Example

Let us calculate the inversion of the 3×3 matrix A_0, extended to 4×4 size in A:

$$A_0 = \begin{bmatrix} 2 & 1 & 0 \\ 6 & 3 & 1 \\ -3 & 0 & 2 \end{bmatrix}_{3\times 3}, \quad A = \left[\begin{array}{c:c} A_0 & 0 \\ \hdashline 0 & I_1 \end{array}\right]. \tag{15}$$

Observe:

$$A^TA = \left[\begin{array}{cc:cc} 49 & 20 & 0 & 0 \\ 20 & 10 & 3 & 0 \\ \hdashline 0 & 3 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]_{4\times 4} = \left[\begin{array}{c:c} B_{(1)} & C_{(1)}^T \\ \hdashline C_{(1)} & D_{(1)} \end{array}\right], \tag{16}$$

$$B_{(1)}^{-1} = \begin{bmatrix} 1/9 & -2/9 \\ -2/9 & 49/90 \end{bmatrix},$$

$$S_{(1)} = D_{(1)} - C_{(1)}B_{(1)}^{-1}C_{(1)}^T = \begin{bmatrix} 1/10 & 0 \\ 0 & 1 \end{bmatrix} = \left[\begin{array}{c:c} B_{(2)} & C_{(2)}^T \\ \hdashline C_{(2)} & D_{(2)} \end{array}\right].$$

On the second recursion level, we need to invert two 2×2 matrices $B_{(1)}$ and $S_{(1)}$. The recursive algorithm on this recursion level goes as follows, by the example of the latter. We calculate the inverse by the general formula (11). The antidiagonal elements of $S_{(1)}^{-1}$ are zero, because $C_{(2)} = 0$ and it is reduced to the form:

$$S_{(1)}^{-1} = \begin{bmatrix} B_{(2)}^{-1} & 0 \\ 0 & B_{(2)}^{-1} \end{bmatrix} = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}. \tag{17}$$

And the final inversion of A_0 in compliance with the formulas (11) and (12) is equal to:

$$A^{-1} = \left[\begin{array}{cc:c} -2 & 2/3 & -1/3 \\ 5 & -4/3 & 2/3 \\ -3 & 1 & 0 \\ \hdashline \multicolumn{2}{c:}{\mathbf{0}_{1\times 3}} & 1 \end{array} \begin{array}{c} \mathbf{0}_{3\times 1} \\ \\ \\ \hdashline \end{array}\right]_{4\times 4} = \left[\begin{array}{c:c} A_0^{-1} & \mathbf{0} \\ \hdashline \mathbf{0} & I_1 \end{array}\right].$$

5. RECURSIVE LU DECOMPOSITION OF MATRICES

In this section, we show how to decompose any square $n \times n$ nonsingular matrix with the efficiency of the used multiplication algorithm into the LU product form.

In general, mathematically the LU decomposition for a rectangular $m \times n$ matrix with a full row rank is the product of the form $A = LUP$, where L is a $m \times m$ lower-triangular matrix with 1's on the main diagonal, U is an $m \times n$ upper-triangular matrix with leading m columns with the rank m, and P is a $n \times n$ permutation matrix, where $m \leq n$. The algorithm recalls itself twice by the recursion. Fig. 2 shows the decomposition after the first recursive call (Steps 5÷9 of the algorithm).

Fig. 3 shows the final decomposition, after the second recursive call (Steps 10÷16 of the algorithm).

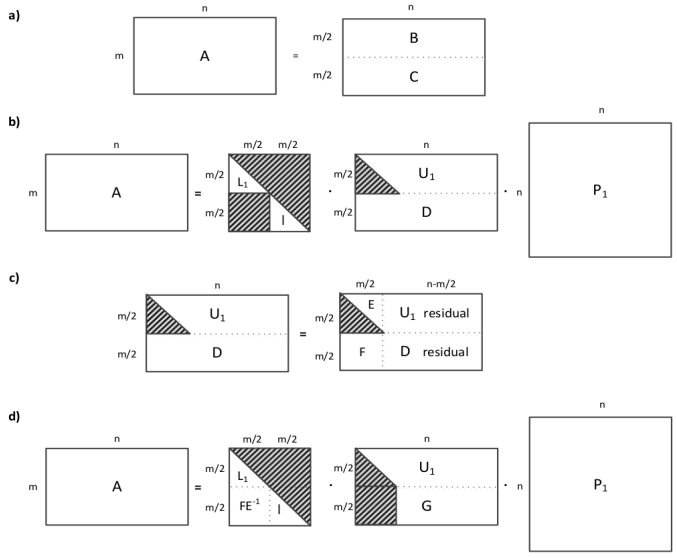

Fig. 2. The LUP decomposition after the first of two recursive calls

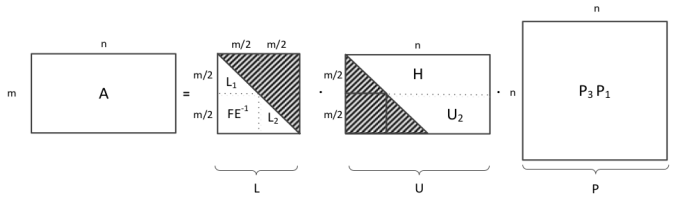

Fig. 3. The final LUP decomposition after the second recursive call

To decompose a nonsingular[1] $n \times n$ matrix, we call the auxiliary procedure FACTOR with the actual parameters (A,n,n). The procedure FACTOR has the form [18]:

procedure *FACTOR(A, m, n)*
Input: *- A: m×n matrix with full row rank*
Output:
- L: m×m lower-triangular matrix with 1's on the main diagonal
- U: m×n upper-triangular matrix with leading m columns with the rank m
- P: n×n permutation matrix
Dimensions assumptions: *- m and n are a natural power of 2 and m<=n*
if *m=1* **then**
0. L:=I_1;
1. c:=< Number of the first non-zero[2] single-row A matrix column >
2. P:=< n×n permutation matrix, swapping the 1 and c A's columns >
3. U:=AP;
4. **return** *(L, U, P);*
else
5. Divide A into (m/2)×n submatrices B and C (Fig. 2a)
6. **call FACTOR***(B, m/2, n), output are the matrices L_1, U_1, P_1*
7. D:=CP_1^{-1}[3]
; At this point we can represent A as a product of 3 matrices (Fig. 2b)
8. Let E and F be the matrices built of the leading m/2 columns of the U_1 and D matrices, respectively (Fig. 2c)
9. G:=D-$FE^{-1}U_1$
; At this point we can represent A as a matrix product presented in Fig. 2d
10. Let G' be the matrix built from the right (n-m/2) G's matrix columns.
11. **call FACTOR** *(G', m/2, n-m/2), output are the matrices L_2, U_2, P_2*
12. P_3:=diag[$I_{m/2}$, P_2]
13. H:=$U_1P_3^{-1}$
14. P:=P_3P_1
15. Let L, U have the form defined by the decomposition Fig. 3.
16. **return** *(L, U, P);*
endif

We verify the crucial part of the algorithm, i.e., zeroing the left part of the D matrix. Observe:

$$\begin{bmatrix} F & D_{\text{res}} \end{bmatrix} = \begin{bmatrix} F & FE^{-1}U_{1\text{res}} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{m/2\times m/2} & D_{\text{res}} - FE^{-1}U_{1\text{res}} \end{bmatrix}$$
$$= FE^{-1}\begin{bmatrix} E & U_{1\text{res}} \end{bmatrix} + I_{m/2}\begin{bmatrix} \mathbf{0}_{m/2\times m/2} & G' \end{bmatrix}.$$

In Step 7, we are permuting the "lower" C matrix by the inverse P_1. That step aims to rearrange the C matrix in a way that allows us to combine it in the common block matrix product after the triangular decomposition of B. Indeed, $DP_1 = CP_1^{-1}P_1 = C$. The same idea is applied in Step 13, where the order of the G' columns is also changed after its triangular decomposition in Step 11. To invert a permutation matrix, it is enough to transpose it.

To prove the correctness of the *FACTOR* procedure, we have to prove that all the assumptions enumerated in the procedure code are fulfilled both for its output parameters and for the two input matrices (B,G') which the *FACTOR* procedure invokes recursively. The proof goes by the induction.

- For $m = 1$ by the above stressed necessary assumption of the input parameter, the input matrix must have a full row rank equal to 1. Thus Step 1 is always correctly executed, because the A matrix must contain the nonzero element. The matrix $L = I_1$ (Step 0) is a lower-triangular matrix of

[1]The nonsingularity means that the matrix is square.

[2]We search for the first nonzero column just for the simplicity of the decomposition formulas in a statistical case; if $c = 1$ the permutation matrix is the identity matrix.

[3]For the permutation matrix $P^{-1} = P^T$.

the size 1×1 with 1's on the main diagonal. In Step 3, the U matrix, after the permutation of A, has its first column of the rank $m = 1$, so all the assumptions to the output matrices L, U, P are fulfilled.

- Let us assume that output variables of the *FACTOR* procedure fulfill the conditions enumerated in the header of the procedure code, for an input matrix size of a certain $m = 2^k, k \geq 0$.
- It is to be proven that those conditions still hold true for the input matrix size for $2m = 2^{k+1}$.

We call the *FACTOR* procedure with the input matrix A of the row rank equal to $2m$. It means that each subset of the $2m$ A's matrix rows is linearly independent, so each of the matrices B and C has the rank m. It ensures that the first call of the *FACTOR* procedure in line 6 has a correct input parameter.

By the base induction step, the first m columns of the matrix U_1 have the rank m as well, so the E matrix is invertible.

From the inequality $\operatorname{rank}[AB] \leq \min(\operatorname{rank} A, \operatorname{rank} B)$, we can conclude that the rank of each matrix in the product in Fig. 2d is at least $2m$. Thus, the row rank of G' is m (see previous but one paragraph) and the second call of the *FACTOR* procedure in line 11 is ensured to work correctly.

By the inductive assumption, both the L_1, L_2 matrices are lower-triangular matrices with 1's on the main diagonal. Thus, the L matrix, defined on the diagram in Fig. 3, preserves these properties.

Now it remains to prove that the U matrix, defined on the same diagram in Fig. 3, is an upper-triangular matrix with leading $2m$ columns with the rank $2m$. Those leading $2m$ columns of the U matrix form a triangular matrix, built block-diagonally by the leading m columns of two matrices: H and U_2. The latter is the result of the *FACTOR* call for G', which we proved above, that works correctly, thus produces the U_2 matrix with the leading columns rank equal to m.

The case of the H matrix is a bit more complicated. According to Step 13, it consists of the permuted columns of the U_1 matrix, returned by calling the *FACTOR* procedure for the B parameter. That permutation is defined by the matrix P_3. The U_1 matrix by the inductive assumption has a full row rank for its leading columns. The construction of P_3, defined on the diagram in Fig. 3, reveals that those leading columns of the matrix U_1 are not affected by this permutation, since the first $m \times m$ diagonal P_3 block is just an identity matrix.

The leading m columns of the matrices H and U_2 are triangular, so all their diagonal entries are nonzero. Thus the determinant of the leading $2m$ columns of the U matrix is also nonzero, so the U matrix has a full row rank. Q.E.D.

Let $T(m,n)$ be the execution time of the call $FACTOR(*,m,n)$. Again we assume that $M(n)$ is the complexity of the involved matrix multiplication algorithm, i.e., $M(n) = O(n^{2+\varepsilon})$ for certain $0 < \varepsilon \leq 1$.

- The base step for $m = 1$ is executed in time $T(1,n) = bn$ for a certain constant b, since we have to search for the nonzero column in the $1 \times n$ matrix A in Step 1. In the worst case, it takes exactly n scalar iterations.
- Recursive calls of the procedure in Steps 6 and 11 take $T(m/2,n)$ and $T(m/2,n-m/2)$ steps, respectively.
- In lines 7 and 13, we are permuting the $m \times n$ matrices C and U_1 thus the complexity of those steps is $O(mn)$.
- Calculation of E^{-1} can be performed by the inverting algorithm specialized in triangular matrices, presented in Section 3, in time $M(m/2)$Calculation of the term FE^{-1} takes also $M(m/2)$ time.
- In the calculation of the term $(FE^{-1}) \cdot U_1$, we should take into account that both m and n are the natural powers of 2 and $m \leq n$[4]. Thus, to obtain the product $(FE^{-1}) \cdot U_1$, we can use the block matrix algebra. Observe:

$$\underbrace{\left[FE^{-1}\right]}_{m/2} \cdots \underbrace{\left[U_1^{(1)} \quad U_1^{(2)} \quad \ldots \quad U_1^{(n/(m/2))}\right]}_{(n/(m/2)) \text{ times}}$$
$$= \underbrace{\left[FE^{-1}U_1^{(1)} \quad FE^{-1}U_1^{(2)} \quad \ldots \quad FE^{-1}U_1^{(n/(m/2))}\right]}_{n}.$$

According to the above scheme, we can calculate the term $(FE^{-1}) \cdot U_1$ in $n/(m/2)$ multiplications of $(m/2) \times (m/2)$ size submatrices and the overall time of that operation is $O((n/m)M(m/2))$.

We can construct the recursive inequality of the complexity for the LU decomposition algorithm with respect to variables m, n:

$$T(m,n) \leq \begin{cases} T\left(\frac{m}{2},n\right) + T\left(\frac{m}{2},n-\frac{m}{2}\right) \\ \quad + c\frac{n}{m}M\left(\frac{m}{2}\right) + dmn, & m > 1, \\ bn, & m = 1. \end{cases} \tag{18}$$

The second recursive call of the procedure in Step 11 is invoked for the submatrix of the $(m/2) \times (n-m/2)$ size, which is smaller than the recursive call in Step 6, being the submatrix of the $(m/2) \times n$ dimensions. Importantly, the *FACTOR* algorithm uses no conditional instructions. Thus the execution time of the algorithm is shorter in the case of smaller dimensionality of the problem to be solved, so $T(m/2,n-m/2) < T(m/2,n)$.

The fourth factor in (18) can be rewritten as $4dn/m \cdot (m/2)\hat{}2$. For the matrix multiplication we have $M(m/2) > (m/2)\hat{}2$, thus we can merge the third and the fourth factor in (18) obtaining for a certain e constant the recursive inequality:

$$T(m,n) \leq \begin{cases} 2T\left(\frac{m}{2},n\right) + e\frac{n}{m}M\left(\frac{m}{2}\right), & m > 1, \\ bn, & m = 1. \end{cases} \tag{19}$$

From the general recursion theorem ([13] Section 4.3) the complexity of the *FACTOR* procedure is equal to (20):

$$T(m,n) = O\left(\frac{n}{m}M(m)\right). \tag{20}$$

[4]That is assured by the first call of the FACTOR procedure, with nonsingular, thus square, matrix as an argument.

Example

Let us calculate the LU decomposition of the 4×7 matrix A_0, extended to the 4×8 size in A:

$$A_0 = \begin{bmatrix} 0 & 0 & 3 & 6 & 2 & 2 & 3 \\ 1 & 7 & 2 & -9 & 3 & 4 & -5 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & -2 & 0 & 4 & -3 & 2 & 2 \\ 2 & 3 & -1 & -6 & 6 & 7 & 2 \end{bmatrix}_{4\times 7},$$

$$A = [A_0 \mid \mathbf{0}_{4\times 1}]_{4\times 8} = \begin{bmatrix} [B_{(1\div 4)}]_{2\times 8} \\ \cdots \\ [C_{(1\div 4)}]_{2\times 8} \end{bmatrix}. \tag{21}$$

Step 6 of the algorithm 6 calls itself recursively to decompose the 2×8 matrix $B_{(1\div 4)}$. Since for $B_{(1\div 2)}$ we have $m = 1$, here Steps $0\div 5$ of the algorithm 5 apply. The first nonzero column of $B_{(1\div 2)}$, which is a single-row matrix, is at the position $c = 3$, so its LU decomposition has the following form:

$$B_{(1\div 2)} = \begin{bmatrix} \underset{\sim}{0} & 0 & \underset{\sim}{3} & 6 & 2 & 2 & 3 & 0 \end{bmatrix}_{1\times 8} = L_{(1\div 2)} U_{(1\div 2)} P_{(1\div 2)}$$
$$= [1] \begin{bmatrix} \underset{\sim}{3} & 0 & \underset{\sim}{0} & 6 & 2 & 2 & 3 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & \underset{\sim}{1} & \\ 0 & 1 & 0 & \mathbf{0}_{3\times 1} \\ \underset{\sim}{1} & 0 & 0 & \\ & \mathbf{0}_{1\times 3} & & I_5 \end{bmatrix}. \tag{22}$$

Now we decompose that "down" $D_{(1\div 2)}$ matrix. Observe how the algorithm Steps $7\div 13$ work in a single row case:

$$E_{(1\div 2)} = 3, \quad F_{(1\div 2)} = 2, \quad F_{(1\div 2)} E^{-1}_{(1\div 2)} = 2/3,$$
$$G_{(1\div 2)} = D_{(1\div 2)} - F_{(1\div 2)} E^{-1}_{(1\div 2)} U_{1(1\div 2)}$$
$$= [2\,7\,1\ -9\,3\,4\ -5\,0] - 2/3 \cdot [3\,0\,0\,6\,2\,2\,3\,0]$$
$$= [0 \vdots 7\ 1\ -13\ 5/3\ 8/3\ -7\ 0]$$
$$= \left[0 \vdots G'_{(1\div 2)}\right]_{1\times 8}.$$

In Step 11, we have the second recursive call of the *FACTOR* procedure for the rows $1\div 2$, but now the matrix $G'_{(1\div 2)}$ is the input parameter. It is a row vector with nonzero first left element thus the LU decomposition is trivial: $G'_{(1\div 2)} = [1] \cdot G'_{(1\div 2)} \cdot I_7$. Now we can construct the decomposition of the $B_{(1\div 4)}$ matrix (21):

$$B_{(1\div 4)} = \begin{bmatrix} 1 & 0 \\ \underset{\sim}{\frac{2}{3}} & 1 \end{bmatrix} \left[\begin{array}{cccccccc} \underset{\sim}{3} & 0 & \underset{\sim}{0} & 6 & 2 & 2 & 3 & \\ \underset{\sim}{0} & 7 & \underset{\sim}{1} & -13 & 5/3 & 8/3 & -7 & \end{array} \vdots\ \mathbf{0}_{2\times 1}\right]$$
$$\cdot \left[\begin{array}{ccc|c} 0 & 0 & \underset{\sim}{1} & \\ 0 & 1 & 0 & \mathbf{0}_{3\times 1} \\ \underset{\sim}{1} & 0 & 0 & \\ \hline & \mathbf{0}_{1\times 3} & & \mathbf{I}_5 \end{array}\right].$$

Let us return to the decomposition of the matrix A.

$$D_{(1\div 4)} = C_{(1\div 4)} P^{-1}_{(1\div 4)} = \begin{bmatrix} \underset{\sim}{0} & -2 & \underset{\sim}{0} & 4 & -3 & 2 & 2 & 0 \\ \underset{\sim}{1} & 3 & \underset{\sim}{2} & -6 & 6 & 7 & 2 & 0 \end{bmatrix},$$

$$E_{(1\div 4)} = \begin{bmatrix} 3 & 0 \\ 0 & 7 \end{bmatrix}, \qquad F_{(1\div 4)} = \begin{bmatrix} 0 & -2 \\ -1 & 3 \end{bmatrix},$$

$$F_{(1\div 4)} E^{-1}_{(1\div 4)} = \begin{bmatrix} 3 & 0 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} 0 & -2 \\ -1 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 0 & -2/7 \\ -1/3 & 3/7 \end{bmatrix}.$$

The algorithm in Step 9, zeroing the $D_{(1\div 4)}$ matrix left part, goes as follows:

$$G_{(1\div 4)} = D_{(1\div 4)} - F_{(1\div 4)} E^{-1}_{(1\div 4)} U_{1(1\div 4)}$$
$$= \left[\begin{array}{cc:cccc:cc} 0 & 0 & \frac{2}{7} & \frac{2}{7} & -\frac{53}{21} & \frac{58}{21} & 0 & 0 \\ & & & & & & & \\ 0 & 0 & \frac{11}{7} & \frac{11}{7} & \frac{125}{21} & \frac{137}{21} & 6 & 0 \end{array}\right] = \left[\mathbf{0}_{2\times 2} \vdots G'_{(1\div 4)}\right].$$

From the LU decomposition of the $G'_{(1\div 4)}$ matrix, we can obtain the following:

$$G'_{(1\div 4)} = \begin{bmatrix} 1 & 0 \\ \frac{11}{2} & 1 \end{bmatrix} \left[\begin{array}{cccc:c} \frac{2}{7} & \overset{\rightleftarrows}{-\frac{53}{21}} & \overset{\rightleftarrows}{\frac{2}{7}} & \frac{58}{21} & 0 & \\ 0 & \frac{119}{6} & 0 & -\frac{26}{3} & 6 & \end{array} \vdots\ \mathbf{0}_{2\times 1}\right]$$
$$\cdot \operatorname{diag}\left[I_1, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, I_3\right].$$

In compliance with Fig. 3, we can obtain the LU decomposition of the matrix $A = L_{1(1\div 8)} U_{1(1\div 8)} P_{1(1\div 8)}$:

$$\left[\begin{array}{cc:cc} 1 & & & \\ \frac{2}{3} & 1 & & \\ \cdots & \cdots & \cdots & \cdots \\ 0 & -\frac{2}{7} & 1 & \\ -\frac{1}{3} & \frac{3}{7} & \frac{11}{2} & 1 \end{array} \vdots\ \mathbf{0}\right] \cdot \left[\begin{array}{cccccccc} \overset{\rightarrow}{3} & 0 & \overset{\leftarrow}{0} & \overset{\downarrow}{2} & \overset{\uparrow}{6} & 2 & 3 & \\ & 7 & 1 & \frac{5}{3} & -13 & \frac{8}{3} & -7 & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & \frac{2}{7} & -\frac{53}{21} & \frac{2}{7} & \frac{58}{21} & 0 & \\ \mathbf{0} & & & \frac{119}{6} & 0 & -\frac{26}{3} & 6 & \end{array} \vdots\ \mathbf{0}_{4\times 1}\right]$$
$$\cdot \operatorname{diag}\left[\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, I_3\right]. \tag{23}$$

6. OTHER MATRIX BLACK BOX OPERATIONS

6.1. Solving the system of linear equations

We can solve a system of linear equations $Ax = b$ by two methods:

- Matrix inversion recursive algorithm

We invert the system matrix A by the algorithm presented in Section 4. The solution is straightforward: $x = A^{-1}b$.

• Matrix LU decomposition

Firstly we decompose the system matrix by the algorithm presented in Section 5, obtaining $A = LUP$. Next, we solve the system $LUPx = b$ in two well-known steps.

6.2. The determinant of a matrix

We can find the determinant of a square matrix by two methods:

• Matrix LU decomposition

Again, we decompose a given system matrix by the algorithm from Section 5. Next, we use the well-known equality expressing the determinant of the product:

$$\det\left[A\right] = \det\left[L\right] \cdot \det\left[U\right] \cdot \det\left[P\right]. \tag{24}$$

Since the matrices L, U are triangular, their determinants are equal to the product of the main diagonal elements. The matrix L contains on its main diagonal 1's, thus $\det[L] = 1$.

To obtain the determinant of the permutation matrix P, it is enough to decompose it into cycles, and next to count those cycles, which are of even length. A linear-time algorithm to find a parity of a permutation by that method is presented in Lipski [20] pp. 20–21.

• Recursive matrix inversion

We can apply the recursive matrix inversion algorithm, described in Section 4, to the recursive calculation of the determinant of the inverted matrix, but with the accuracy to the sign. Indeed, from the equality (10) it follows:

$$\det\left[A\right] = \det\left[B\right] \cdot \det\left[D - CB^{-1}C^T\right].$$

Since $\det[A^T] = \det[A]$ and the known property of the block triangular matrices is that their determinant is equal to the product of the determinants of the diagonal matrix blocks (which here are the identity matrices), from (10) we can obtain the square of the matrix determinant:

$$\det\left[A\right] \cdot \det\left[A^T\right] = \det^2\left[A\right] = \det\left[B\right] \cdot \det\left[D - CB^{-1}C^T\right]. \tag{25}$$

6.3. Other applications

- **Matrix inversion**: we can invert any nonsingular matrix in another way than it was showed in Section 4. Namely, we can use the LU decomposition by the following matrix calculations:

$$A^{-1} = (LUP)^{-1} = P^{-1}U^{-1}L^{-1} = P^T U^{-1} L^{-1}.$$

  The inversions U^{-1} and L^{-1} we can obtain by the inversion algorithm 3, specialized just to triangular matrices.
- **Polynomial calculations**: [21] in Section 2.6 proposes how to evaluate a polynomial at a large number of points at once by matrix multiplication in $O(M(n))$ time.
- **Rank of the matrix and related problems**: [22] proposed a generalized LUP decomposition, here called the SQP decomposition, applicable also to matrices without the full row rank. The results of the article can be also used to calculate the rank of a matrix together with the respective nonsingular minor in $O(M(n))$ time.
- **The characteristic polynomial of a matrix**: [23] proposed three algorithms to calculate the matrix characteristic polynomial coefficients, with different generality and efficiency. Algorithm 1 works in $O(\log n \cdot M(n))$ time but is applicable only in special cases of the input matrix. Algorithm 3 is the fastest, achieving $O(M(n))$ time, i.e., the same as the used matrix multiplication algorithm, but is not fully general either. Algorithm 3 works in $O(\log n \cdot M(n))$ time but works for any square input matrix.
- **Graph paths problems**: [24] computes the shortest distances between all pairs of vertices of an undirected, unweighted graph in $O(\log n \cdot M(n))$ time. [25] generalizes these results to the case when the weights are between 0 and B with $O(B^2 \cdot \log n \cdot M(n))$ time.
- **Formal grammar problems**: [26] achieved $O(M(n))$ time in the problem of context-free recognition.
- **Compiler construction**: [27] shows how to find a transitive closure of a graph in $O(M(n))$ time algorithm. [28] solves an inverse problem, i.e., how to multiply matrices with the use of the available transitive closure algorithm.

7. SUMMARY

In this article, we surveyed a series of algorithms on efficient matrix multiplication algorithms and their applications. The dependencies of all the procedures are summarized in Fig. 4.

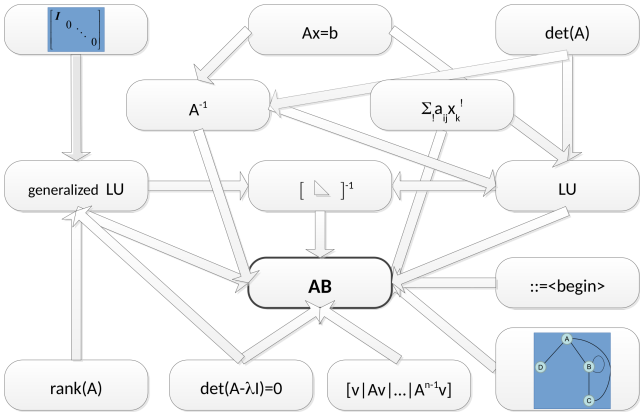

Fig. 4. Dependency graph of the black box algorithms

As an example of applications of matrices one can mention [29] exploiting the structure of the matrix and [30] based on the matrix block decomposition.

The quite recent work [31] from 2016 on the Strassen algorithm proved that those algorithms can be efficiently implemented even for small matrices, not quite square and for multicore architecture.

The pursue for the fastest matrix multiplication is still going on. These questions remain open:

- [32] in 2003 proved that the theoretical lower multiplication bound for 3×3 matrices algorithm is 19, but the so-far best result is 23 [33].
- [34] from 2017 multiplies 5x5 matrices by 99 multiplications, with $k = 2.8551$, being also worse than the oryginal 2×2 Strassen algorithm.

- In the scientific community working with the matrix multiplication algorithm, there is a popular hypothesis that it is possible to multiply the matrices in $O(n^{2+\varepsilon})$ time, for any (!) positive ε.
- The analysis on the numerical stability issue of inversion methods, i.e., direct (11) and its Schönhage modification (12), is presented in Higham [19], Section 26.3.2.
- Work [22] presents, how to obtain Moore-Penrose pseudoinverse in $O(M(n))$ time. An open question is how to obtain a Drazin pseudoinverse, used in control problems [35].

ACKNOWLEDGEMENTS

This work was supported by Statutory Research funds of Department of Applied Informatics, Silesian University of Technology, Gliwice, Poland (02/100/BK_20/0003).

REFERENCES

[1] V. Strassen, "Gaussian elimination is not optimal", *Numer. Math.*, vol. 13, pp. 354–356, 1969.
[2] V. Pan, "Strassen's algorithm is not optimal", in *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, 1978, pp. 166–176, doi: 10.1109/SFCS. 1978.34.
[3] D. Bini, M. Capovani, G. Lotti., and F. Romani, "O(n^2.7799) complexity for approximate matrix multiplication", *Inf. Proc. Lett.*, vol. 8, no. 5, pp. 234–235, 1979.
[4] A. Schonhage, "Partial and Total Matrix Multiplication", *SIAM J. Comput.*, vol. 10, no. 3, pp. 434–455, 1981.
[5] F. Romani, "Some properties of disjoint sums of tensors related to matrix multiplication", *SIAM J. Comput.*, vol. 11, pp. 263–267, 1982.
[6] D. Coppersmith and S. Winograd, "On the Asymptotic Complexity of Matrix Multiplication", *SIAM J. Comput.*, vol. 11, no. 3, pp. 472–492, 1982.
[7] V. Strassen, "The Asymptotic Spectrum of Tensors and the Exponent of Matrix Multiplication", in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, pp. 49–54, doi: 10.1109/SFCS.1986.52.
[8] D. Coppersmit and S. Winograd, "Matrix multiplication via arithmetic progressions", *J. Symb. Comput.*, vol. 9, no. 3, pp. 251–280, 1990.
[9] A. Davie and A. Stothers, "Improved bound for complexity of matrix multiplication", *Proc. Royal Society of Edinburgh: Section A Mathematics*, vol. 143, no. 2, pp. 351–369, 2013.
[10] V. Vassilevska-Williams, "Multiplying matrices in O(n^2.373) time", Tech. Rep., Stanford University, 2014.
[11] F. Le Gall, "Powers of tensors and fast matrix multiplication", in *ISSAC '14: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, 2014.
[12] S. Winograd, "Some remarks on fast multiplication of polynomials", *Complexity of Sequential and Parallel Numerical Algorithms*, J.F. Traub, Ed., Academic Press, New York, 1973, pp. 189–208.
[13] T.H. Cormen, Ch.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms*, The MIT Press, Cambridge, 2009.
[14] P.C. Fischer, "Further Schemes for Combining Matrix Algorithms", *Automata, Languages and Programming. ICALP 1974. Lecture Notes in Computer Science*, Loeck J., Ed., vol 14, pp. 428–436, 1974, doi: 10.1007/978-3-662-21545-6_32.
[15] J.R. Hopcroft and L.R. Kerr, "On Minimizing the Number of Multiplications Necessary for Matrix Multiplication", *SIAM J. Appl. Math.*, vol. 20, no. 1, pp. 30–36, 1971.
[16] R. Probert, "On the Complexity of Matrix Multiplication", University of Waterloo, Comp. Sci. Tech. Report CS-73-27.
[17] T. Kaczorek, *Wektory i macierze w automatyce i elektrotechnice*, 2nd ed. WNT, Warszawa, 1998, [in Polish].
[18] J. Bunch and J. Hopcroft, "Triangular factorization and inversion by fast matrix multiplication", *Math. Comput.*, vol. 28, no. 125, pp. 231–236, 1974.
[19] A. Schonhage, "Fast Schmidt Orthogonalization and Unitary Transformations of Large Matrices", *Complexity of Sequential and Parallel Numerical Algorithms*, J.F. Traub, Ed., Academic Press, New York, 1973, pp. 283–291.
[20] W. Lipski, *Kombinatoryka dla programistów*, WNT, Warszawa, 1989, [in Polish].
[21] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, Am. Elsevier Publ. Comp., New York, 1975.
[22] O.H. Ibarra, S. Moran, and R. Hui, "A generalization of the fast LUP matrix decomposition algorithm and applications", *J. Alg.*, vol. 3, pp. 45–56, 1982.
[23] W. Keller-Gehrig, "Fast algorithms for the characteristic polynomial", *Theor. Comput. Sci.*, vol. 36, pp. 309–317, 1985.
[24] R. Seidel, "On the All-Pairs-Shortest-Path Problem", in *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing*, ACM Press, New York, 1992, pp. 745–749.
[25] N. Alon, Z. Galil, and O. Margalit, "On the exponent of the all pairs shortest path problem", in *Proc. 32nd Ann. IEEE Symp. on Found. Comp. Sci.*, 1991, pp. 569–575.
[26] L.G. Valiant, "General context-free recognition in less than cubic time", *J. Comput. Syst. Sci.*, vol. 10, pp. 308–315, 1975.
[27] I. Munro, "Efficient determination of the transitive closure of a directed graph", *Inform. Process. Lett.*, vol. 2, pp. 56–58, 1971.
[28] M.J. Fisher and A.R. Meyer, "Boolean matrix multiplication and transitive closure", in *12th IEEE Ann. Symp. on Switching and Automata Theory*, 1971, pp. 129–131.
[29] T. Kaczorek, "Structure decomposition of normal 2D transfer matrices", *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 52, no. 4, pp. 353-357, 2004.
[30] X. Yang, D. Liu, D. Zhou, and R. Yang, "Moving cast shadow detection using block nonnegative matrix factorization", *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 66, no. 2, pp. 229–234, 2018, doi: 10.24425/122103.
[31] J. Huang, T.M. Smith, G.M. Henry, and R.A. Geijn, "Strassen's Algorithm Reloaded", in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 690–701, doi: 10.1109/SC.2016.58.
[32] M. Blaser, "On the complexity of the multiplication of matrices of small formats", *J. Complexity*, vol. 19, no. 1, pp. 43-60, 2003, doi: 10.1016/S0885-064X(02)00007-9.
[33] J.D. Laderman, "A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications", *Bull. Am. Math. Soc.*, vol. 82, no. 1, pp. 126–128, 1976.
[34] A. Sedoglavic, "A noncommutative algorithm for multiplying 5×5 matrices using 99 multiplications", hal-01562131v5, 2017.
[35] T. Kaczorek, "Drazin inverse matrix method for fractional descriptor discrete-time linear systems", *Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 64, no. 4, pp. 395–399, 2016. doi: 10.24425/bpas.2016.98076.