ISSN 0361-7688, Programming and Computer Software, 2023, Vol. 49, No. 5, pp. 441–447. © Pleiades Publishing, Ltd., 2023.
Russian Text © The Author(s), 2023, published in Programmirovanie, 2023, Vol. 49, No. 5.

Effective Lower Bounds on the Matrix Rank and Their Applications

O. A. Zverkov[a,*] and A. V. Seliverstov[a,]**

[a] *Institute for Information Transmission Problems (Kharkevich Institute), Russian Academy of Sciences, Bol'shoi Karetnyi per. 19/1, Moscow, 127051 Russia*

**e-mail: zverkov@iitp.ru*

***e-mail: slvstv@iitp.ru*

Received June 26, 2022; revised July 27, 2022; accepted October 30, 2022

Abstract—We propose an efficiently verifiable lower bound on the rank of a sparse fully indecomposable square matrix that contains two non-zero entries in each row and each column. The rank of this matrix is equal to its order or differs from it by one. Bases of a special type are constructed in the spaces of quadratic forms in a fixed number of variables. The existence of these bases allows us to substantiate a heuristic algorithm for recognizing whether a given affine subspace passes through a vertex of a multidimensional unit cube. In the worst case, the algorithm may output a computation denial warning; however, for the general subspace of sufficiently small dimension, it correctly rejects the input. The algorithm is implemented in Python. The running time of its implementation is estimated in the process of testing.

DOI: 10.1134/S0361768823020160

1. INTRODUCTION

Python is a modern general-purpose programming language that has come out on top in recent years in various ratings and is especially popular in scientific applications. The advantages of Python are the ease of entry-level mastering, the high speed of the development process, as well as the compactness and readability of program texts. Its main disadvantage is the relatively low performance of programs, which is mainly due to the interpreted nature of the language and dynamic typing. This, however, is largely compensated for by a wide variety of available libraries that provide Python developers with access to efficient implementations of various algorithms and the possibility to connect original extension modules written in compiled languages [1]. In the context of our study, the main feature of Python is the built-in support of integers of unlimited bit length and rational numbers.

The rank of an $n \times n$ matrix over a field can be calculated using a polynomial number of processors with only $O(\log_2^2 n)$ operations over this field for each of them [2, 3]. Efficient parallelization is feasible not only for the calculation of the rank but also for the computation of an inverse matrix, LDU-decomposition of a square matrix, and Cholesky decomposition of a symmetric positive definite matrix [4, 5]. On the other hand, the complexity of calculating the rank [6] and characteristic polynomial [7–9] is close to that of matrix multiplication. Easily verifiable non-degeneracy conditions are known for circulants [10]. The problem of checking the linear independence of a system of polynomials in several variables is of theoretical interest. These polynomials can sometimes be given by short expressions or arithmetic schemes, even though their coefficient matrices are large. Therefore, easily verifiable estimates of the matrix rank are important. Generally, the calculations are carried out over the field of rational numbers; however, symbolic calculations can also be performed over fields of algebraic numbers [11, 12].

Points of a determinantal variety correspond (up to multiplication by a non-zero number) to fixed-size matrices the rank of which is bounded from above. A determinantal variety can have a very large degree [13, 14]. Hence, over an algebraically closed field, it is impossible to check the rank of a matrix by using a small non-branching program. On the other hand, these varieties contain linear subspaces, which can sometimes be used to estimate the rank [15].

Using rank estimates, we consider a heuristic algorithm for checking the incidence of a given subspace to some vertex of a unit cube. This check is equivalent to finding a (0, 1)-solution to the system of equations

$$\begin{cases} x_0 = 1 \\ x_j = \ell_j(x_0, \ldots, x_s), \quad j > s \\ x_i \in \{0,1\}, \quad i > 0, \end{cases}$$

where ℓ_j denotes linear forms [16].

Even though heuristic algorithms for solving this problem under additional constraints [17–19] are

available, this problem is considered computationally hard in the worst case. Thus, it is important to design new algorithms whose computational complexity is on average limited by a polynomial in the length of the input under different constraints.

As for the recognition problem, we assume three variants of response: the input is not only accepted or rejected, but an explicit warning of computation denial is also possible. In this case, the response must be received in finite time and without errors; if the easily verifiable condition is met, then the computation denial warning can be issued only on a small portion of inputs among all inputs of a given length [20, 21]. When estimating algebraic complexity, it is sufficient to require that the computation denial warning be generated on a set of inputs for which some polynomial other than an identically zero one vanishes [16, 22].

The paper is organized as follows. Section 2 presents theoretical results. Section 3 describes the implementation of the algorithm in Python. Section 4 contains brief conclusions.

2. THEORETICAL RESULTS

The entries of an (0, 1)-matrix are only zeros and ones. A submatrix is a matrix obtained by deleting some rows and columns. An $n\times n$ matrix is said to be fully indecomposable if it does not contain a zero submatrix of size $s\times(n-s)$ for any s.

Let us consider square matrices with two non-zero entries in each row and each column. If a (0, 1)-matrix of this type is fully indecomposable, then its permanent is two [23]. In this case, its determinant belongs to set $\{-2,0,2\}$. Obviously, any such (0, 1)-matrix over a field of characteristic two is degenerate. An example of a degenerate matrix over an arbitrary field is a 2×2 matrix of rank one each entry of which is equal to one. Another example is the (0, 1)-circulant

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

This matrix is also degenerate, but its rank is three. The characteristic polynomial of this matrix is $x^4 - 4x^3 + 6x^2 - 4x$. We provide the lower bound for the rank of these matrices.

Theorem 1. *Suppose that we have fully indecomposable $n\times n$ matrix A where each row and each column has exactly two non-zero entries. Then, the rank of matrix A has a lower bound*: $\operatorname{rank}(A)\geq n-1$.

Proof. The condition of full indecomposability is not violated when permuting rows or columns. Therefore, without loss of generality, we assume that all entries on the main diagonal of matrix A are not zero. Then, matrix A is a sum of two non-degenerate matrices $A = D + CP$, where matrices C and D are both diagonal and non-degenerate, while P is a permutation matrix with zeros on the main diagonal. With matrix A being fully indecomposable, permutation P consists of one cycle of length n.

For any permutation matrix Q, matrix QAQ^{-1} is obtained from matrix A by consistent permutation of rows and columns. In this case, the entries on its main diagonal remain on the main diagonal. Therefore, for some permutation matrix Q, matrix QAQ^{-1} has non-zero entries on the main diagonal, directly above the main diagonal, and one more entry in the first column and last row. Such matrix QAQ^{-1} has the following form (for $n > 6$):

$$\begin{pmatrix} * & * & 0 & \cdots & 0 & 0 & 0 \\ 0 & * & * & \cdots & 0 & 0 & 0 \\ 0 & 0 & & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & * & * & 0 \\ 0 & 0 & 0 & \cdots & 0 & * & * \\ * & 0 & 0 & \cdots & 0 & 0 & * \end{pmatrix},$$

where symbol * denotes non-zero entries. Since the permutation of rows or columns does not affect the rank of a matrix, the rank of matrix QAQ^{-1} coincides with the rank of matrix A. To complete the proof of the theorem, it is sufficient to find a non-degenerate $(n-1)\times(n-1)$ submatrix B in matrix QAQ^{-1}. Submatrix B is obtained by deleting the first row and first column of matrix A. This matrix is upper triangular. Each entry on the main diagonal of matrix B is not zero, which is why B is not degenerate.

In Theorem 1, the condition of complete indecomposability is important. For instance, with even n, a block-diagonal $n\times n$ matrix the diagonal of which contains 2×2 blocks with four ones in each block has a rank of $n/2$.

Hereinafter, we consider fields of characteristic zero. Let us consider the problem of incidence of a given affine subspace to some vertex of an n-dimensional cube. The method consists in constructing a low-degree algebraic hypersurface that passes through each vertex of the cube but does not intersect the affine subspace. The computational complexity of this method depends on the degree of the hypersurface. On the other hand, this method is efficient only if the dimension of the affine subspace is sufficiently low. Suppose that this subspace is defined by a system of m linear equations. The dimension of the linear space of forms of degree d in variables $x_0, \ldots, x_n$ is equal to binomial coefficient $\binom{n+d}{d}$. Therefore, on average, this method is efficient when the number of equations

satisfies inequality $m \geq n - \sqrt{d(d-1)n - o(n)}$, where d is the degree of the hypersurface. We confine ourselves to the case of $d = 2$, where the hypersurface is a quadric [16].

Suppose that the n-dimensional cube is in a projective space and the homogeneous coordinates of its vertices belong to set $\{0,1\}$; however, none of the cube vertices lies on infinitely distant hyperplane $x_0 = 0$. It is well known [24] that the quadric passing through each vertex of this cube is determined by a quadratic form of the type

$$\lambda_1 x_1(x_1 - x_0) + \cdots + \lambda_n x_n(x_n - x_0).$$

If the subspace is given by the system of equations $x_j = \ell_j(x_0,\ldots,x_{n-m})$, where $n - m < j \leq n$, then the existence of the quadric that passes through each vertex of the cube and intersects this subspace only at infinity is equivalent to the resolvability of the equation

$$\sum_{k=1}^{n-m} \lambda_k x_k(x_k - x_0) + \sum_{j=n-m+1}^{n} \lambda_j \ell_j(\ell_j - x_0) = x_0^2$$

with respect to variables $\lambda_1, \ldots, \lambda_n$. In turn, this problem is equivalent to the resolvability of a system of linear equations. According to the Kronecker–Capelli theorem, this check is reduced to comparing the ranks of two matrices. Algorithm 1 does not provide polynomial computational complexity in the worst case. However, it is sometimes significantly more efficient than the exhaustive search through the cube vertices.

In Algorithm 1, the first $n - m$ columns of matrix A always contain two non-zero entries 1 and -1. In the column with number $j > n - m$, the entries are equal to second-degree polynomials in the coefficients of linear form ℓ_j.

Algorithm 1. Checking the existence of a cube vertex incident to a given subspace.

Input: integers $0 < m < n$ and linear form $\ell_j(x_0,\ldots,x_{n-m})$, where $n - m < j \leq n$.

1. The entries of matrix A are coefficients of the form

$$\sum_{k=1}^{n-m} \lambda_k x_k(x_k - x_0) + \sum_{j=n-m+1}^{n} \lambda_j \ell_j(\ell_j - x_0),$$

where the rows of matrix A correspond to second-degree monomials in variables $x_0, \ldots, x_{n-m}$, while its columns correspond to variables $\lambda_1, \ldots, \lambda_n$.

2. Augmented matrix B is obtained from matrix A by adding a column in which one is in the row corresponding to monomial x_0^2 with the other entries of this column being zero.

3. **if** $\operatorname{rank}(A) = \operatorname{rank}(B)$, **then** the input is rejected.

4. **else** the computation denial warning is generated.

Let us consider an example corresponding to a straight line on a plane. Suppose that $n = 2$, $m = 1$, and $\ell_2 = x_0 + x_1$ is a linear form. Then, we have the equation

$$-\lambda_1 x_0 x_1 + \lambda_1 x_1^2 + \lambda_2 x_0 x_1 + \lambda_2 x_1^2 = x_0^2.$$

Here, there are three monomials in variables x_0 and x_1: x_0^2, x_0x_1, and x_1^2. The corresponding matrix A has two columns and three rows:

$$A = \begin{pmatrix} 0 & 0 \\ -1 & 1 \\ 1 & 1 \end{pmatrix}.$$

The augmented matrix is

$$B = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Here, $\operatorname{rank}(A) = 2$ and $\operatorname{rank}(B) = 3$. Therefore, the desired variables λ_1 and λ_2 do not exist. Algorithm 1 outputs the computation denial warning.

Let us consider another example corresponding to a straight line in a space. Suppose that $n = 3$, $m = 2$, and $\ell_2 = 2x_0 + x_1$ and $\ell_3 = 3x_1$ are linear forms. Then, we have equation $\lambda_1(-x_0x_1 + x_1^2) + \lambda_2(2x_0^2 + 3x_0x_1 + x_1^2) + \lambda_3(-3x_0x_1 + 9x_1^2) = x_0^2$. Here, there are again three monomials in variables x_0 and x_1: x_0^2, x_0x_1, and x_1^2. However, matrix A has three columns and three rows:

$$A = \begin{pmatrix} 0 & 2 & 0 \\ -1 & 3 & -1 \\ 1 & 1 & 9 \end{pmatrix}.$$

The augmented matrix is

$$B = \begin{pmatrix} 0 & 2 & 0 & 1 \\ -1 & 3 & -3 & 0 \\ 1 & 1 & 9 & 0 \end{pmatrix}.$$

Matrix A is non-degenerate, and the ranks of matrices A and B coincide. Algorithm 1 rejects the input.

The following theorem provides a sufficient condition of type $m \geq n - \sqrt{2n - o(n)}$ under which Algorithm 1 rejects a large portion of inputs for fixed n and m. Its proof is based on constructing a basis of a special type in the space of quadratic forms.

Theorem 2. *Suppose that we have positive integers n and $m < n$ for which inequality $2n \geq (n - m + 1)(n - m + 2)$ holds. For almost every set of m linear forms $\ell_j(x_0,\ldots,x_{n-m})$, where $n - m < j \leq n$, there are values of coefficients $\lambda_1, \ldots, \lambda_n$ for which the following equality of quadratic forms holds*:

$$\sum_{k=1}^{n-m}\lambda_k x_k(x_k - x_0) + \sum_{j=n-m+1}^{n}\lambda_j \ell_j(\ell_j - x_0) = x_0^2.$$

If, for some $\varepsilon > 0$, the coefficients of linear forms ℓ_j are independently and uniformly distributed over a set of $\lceil 2n/\varepsilon \rceil$ numbers, then the probability of absence of the desired values of $\lambda_1, \ldots, \lambda_n$ does not exceed ε.

Proof. Let us consider matrix A from Algorithm 1. By the condition of the theorem, the number of rows $r = (n-m+1)(n-m+2)/2$ in matrix A does not exceed the number of columns n. In this case, a sufficient condition for the existence of desired $\lambda_1, \ldots, \lambda_n$ is equality $\operatorname{rank}(A) = r$. Each entry of matrix A is either a constant or a second-degree polynomial in the coefficients of linear forms ℓ_j. Therefore, the rth-order minor of matrix A is equal to a polynomial in the coefficients of linear forms ℓ_j the degree of which does not exceed $2r$.

According to the Schwartz–Zippel lemma [25], if this polynomial is not identically zero, then the vanishing probability does not exceed $2r/\lceil 2n/\varepsilon \rceil \le \varepsilon$.

It remains to show that, for some set of forms ℓ_j, the r-order corner minor of matrix A is not zero. For this purpose, it is sufficient to consider a set of forms whose coefficients do not necessarily belong to the set specified in the condition. Suppose that $\ell_{n-m+k} = x_0 + x_k$ for $1 \le k \le n-m$. By $f(i,k)$, we denote the number of a pair of subscripts $1 \le i < k \le n-m$, which takes values from 1 to $(n-m)(n-m-1)/2$. Suppose that $\ell_j = x_i + x_k$ for $j = 2(n-m) + f(i,k)$. For $j = r$, we assume that $\ell_r = 2x_0$.

For subscripts $1 \le i < k \le n-m$, monomial $x_i x_k$ corresponds to a row where only the entry in column $j = 2(n-m) + f(i,k)$ is not zero. This entry is equal to two. On the other hand, the columns with numbers $1 \le j \le n-m$ have two non-zero entries: -1 in the row corresponding to monomial $x_0 x_j$ and 1 in the row corresponding to monomial x_j^2. Let us rearrange the rows in matrix A in accordance with the following order of monomials: $x_0x_1, \ldots, x_0x_{n-m}, x_1^2, x_2^2, \ldots, x_{n-m}^2$, monomials $x_i x_k$ for $1 \le i < k \le n-m$, and the last one x_0^2. The corner submatrix of order r takes the form

$$\begin{pmatrix} -I & I & * & 0 \\ I & I & * & 0 \\ 0 & 0 & 2I' & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix},$$

where I is the identity matrix of order $n-m$, I' is the identity matrix of order $(n-m)(n-m-1)/2$, and 0 is the zero matrix. This submatrix is non-degenerate because elementary transformations of its rows yield a triangular matrix with non-zero entries on the main diagonal:

$$\begin{pmatrix} -I & I & * & 0 \\ 0 & 2I & * & 0 \\ 0 & 0 & 2I' & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}.$$

Therefore, $\operatorname{rank}(A) = r$.

It should be noted that the proof of Theorem 2 uses sparse matrices. Hence, the sparseness of matrix A does not prevent the successful application of the algorithm under consideration. Moreover, for sparse matrices, the complexity of rank calculation is less, which makes it possible to process high-dimensional matrices. In the proof of Theorem 2, we have considered a special case where matrix A has a full rank. Therefore, the procedure of calculating its rank can be replaced with checking the non-degeneracy of matrix A. However, this would be an unreasonable constraint on the applicability of Algorithm 1.

3. IMPLEMENTATION

Suppose that L is a matrix composed of coefficients of linear forms ℓ_j in Algorithm 1 and Theorem 2. The first row of matrix L corresponds to form ℓ_{n-m+1}, while its last row corresponds to form ℓ_n. For instance, for two linear forms $\ell_2 = 2x_0 + x_1$ and $\ell_3 = 3x_1$, this matrix is

$$L = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}.$$

Matching matrix L and matrix B takes less time than calculating the ranks of A and B. This transformation is implemented by function `compose_b()`, which takes matrix L as the input and returns matrix B, with both the matrices being represented as NumPy arrays. The order of rows in B corresponds to the lexicographic monomial order, while its first row corresponds to monomial x_0^2. Matrix A is easy to obtain by deleting the last column of matrix B. Function `compose_b()` is implemented in Python using the NumPy library.

When using integers of fixed bit length, NumPy significantly improves the efficiency of processing matrices (both in terms of speed and memory allocation) as compared to data types built in Python, e.g., lists of numbers [26]. When working with other data types, including integers of unlimited length and rational numbers, these advantages mostly vanish. However, NumPy enables concise representation of standard operations on matrices and their parts, e.g., in our case, submatrix extraction, entry-wise addition, and multiplication of rows, etc.

By $s = n - m$, we denote the dimension of a subspace for which the incidence to the vertex of the n-

dimensional cube is checked, while `height` denotes the number of second-degree monomials in variables $x_0, ..., x_s$, which is equal to $(s+1)(s+2)/2$.

To estimate the running time of function `compose_b()` for $n = $ `height`, an $(n-s)\times(s+1)$ matrix L was generated the entries of which are random integers in interval $[-10^9, 10^9]$. The time it takes to calculate matrix B is shown in Table 1. The third column of Table 1 contains results for 64-bit integers, while its last column contains results for integers of unlimited bit length, the computations over which were implemented in Python. We used Python 3.10.4 and NumPy 1.22.4. The calculations were carried out on a personal computer with Intel® Core i5-3570 and 16 GB RAM. The calculations with numbers of unlimited bit length for large matrices were not carried out because of RAM overflow, although the calculations for 64-bit numbers were continued up to $s = 300$.

The simultaneous calculation of the ranks of matrices A and B based on input matrix B is also implemented in Python. For the calculations with rational numbers, the Fraction class from the Python standard library is used.

The rank calculation is based on reducing the matrix to the echelon form. With matrix A being a submatrix of B, for both matrices A and B, this transformation can be carried out simultaneously. This makes it possible to nearly halve the running time as compared to independent calculation of their ranks.

To estimate the time it takes to evaluate function `rank_ab()` for $n = $ `height`, an $(n-s)\times(s+1)$ matrix L was generated the entries of which are random integers in interval $[-N, N]$. The evaluation time for different boundary values $N \in \{10^3, 10^6, 10^9\}$ is shown in Table 2. The larger the matrix, the faster the increase in the running time with an increase in the average bit length of matrix entries.

It should be noted that the rank calculation method implemented using the NumPy library, which is based on the singular value decomposition, is not suitable in this case because of possible errors associated with the use of floating point numbers. For instance, when calculating the rank with NumPy 1.22.4 in such an inaccurate way, at $k \geq 16$, the rank of a diagonal 2×2 matrix with entries 1 and 10^k on the main diagonal proves equal to one. That is why, to calculate the matrix rank, we used a new implementation of the well-known Gaussian algorithm for reducing the matrix to the echelon form, which allowed us to work with numbers of unlimited bit length.

Thus, the final implementation of Algorithm 1 is function `may_have_01solution()`, which receives matrix L as the input. The value is `True` when Algorithm 1 would output the computation denial warning.

Table 1. Time (in seconds) it takes to compute matrix B at different values of s and n for two integer types: 64-bit and unlimited bit length

Parameters		Integer type	
s	n	64-bit	unlimited
20	231	0.01	0.02
30	496	0.03	0.08
40	861	0.07	0.20
50	1326	0.14	0.50
60	1891	0.25	1.00
70	2556	0.43	1.90
80	3321	0.68	3.20
90	4186	1.10	5.00
100	5151	1.50	8.20
110	6216	2.20	12
120	7381	3.00	16
130	8646	4.20	24
140	10011	5.40	36
150	11476	7.50	51
160	13041	9.50	66
170	14706	13	89
180	16471	15	
190	18336	20	
200	20301	23	
210	22366	29	
220	24531	34	
230	26796	43	
240	29161	50	
250	31626	62	
260	34191	83	
270	36856	89	
280	39621	100	
290	42486	130	
300	45451	170	

The `False` value means that Algorithm 1 rejects the input. The input can be a matrix with rational entries.

The listings of the functions discussed above with the corresponding examples are available at http://lab6.iitp.ru/-/qualg.

4. CONCLUSIONS

In this paper, we have described a heuristic polynomial-time algorithm implemented in Python for recognizing subspaces of sufficiently low dimension that are incident to any of the vertices of a multidimensional unit cube; in the worst case, this problem is considered computationally hard. The algorithm

Table 2. Time (in seconds) it takes to calculate the rank of matrix B for different values of parameters s, n, and N

Parameters		N		
s	n	10^3	10^6	10^9
5	21	0.02	0.02	0.02
6	28	0.04	0.05	0.07
7	36	0.09	0.10	0.19
8	45	0.21	0.34	0.50
9	55	0.45	0.77	1.20
10	66	0.91	1.70	2.70
11	78	1.80	3.50	5.70
12	91	3.30	6.90	12
13	105	6	13	23
14	120	11	24	42
15	136	18	42	76
16	153	30	73	130
17	171	50	120	230
18	190	79	200	380
19	210	120	330	610
20	231	190	510	960

either correctly rejects the input or reports that there is no obvious obstacle to the incidence. In particular, the derived bounds on the complexity of this problem can be useful for estimating the reliability of cryptographic protocols based on the search for (0,1)-solutions to systems of equations [19, 27].

The algorithm uses matrix rank calculation. The estimates for the rank of an indecomposable sparse matrix obtained in Theorem 1 make it possible to test the calculation of high-order matrix ranks.

ACKNOWLEDGMENTS

The authors are grateful to an anonymous referee for useful comments that made it possible to improve this paper.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

1. Gevorkyan, M.N., Korolkova, A.V., Kulyabov, D.S., and Sevast'yanov, L.A., A modular extension for a computer algebra system, *Program. Comput. Software,* 2020, vol. 46, no. 2, pp. 98–104.
2. Chistov, A.L., Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic, *Lect. Notes Comput. Sci.,* 1985, vol. 199, pp. 63–69. https://doi.org/10.1007/BFb0028792
3. Mulmuley, K., A fast parallel algorithm to compute the rank of a matrix over an arbitrary field, *Combinatorica,* 1987, vol. 7, no. 1, pp. 101–104. https://doi.org/10.1007/BF02579205
4. Malaschonok, G. and Tchaikovsky, I., About big matrix inversion, *Comput. Algebra,* Abramov, S.A. and Sevastyanov, L.A., Eds., Moscow: MAKS Press, 2021, pp. 81–84. https://doi.org/10.29003/m2019.978-5-317-06623-9
5. Malaschonok, G.I. and Sidko, A.A., Supercomputer environment for recursive matrix algorithms, *Program. Comput. Software,* 2022, vol. 48, pp. 90–101.
6. Cheung, H.Y., Kwok, T.C., and Lau, L.C., Fast matrix rank algorithms and applications, *J. ACM,* 2013, vol. 60, no. 5, pp. 1–25. https://doi.org/10.1145/2528404
7. Abdeljaoued, J. and Malaschonok, G.I., Efficient algorithms for computing the characteristic polynomial in a domain, *J. Pure Appl. Algebra,* 2001, vol. 156, pp. 127–145. https://doi.org/10.1016/S0022-4049(99)00158-9
8. Pereslavtseva, O.N., Calculation of the characteristic polynomial of a matrix, *Discrete Math. Appl.,* 2011, vol. 21, no. 1, pp. 109–128.
9. Neiger, V. and Pernet, C., Deterministic computation of the characteristic polynomial in the time of matrix multiplication, *J. Complexity,* 2021, vol. 67, no. 101572, pp. 1–35. https://doi.org/10.1016/j.jco.2021.101572
10. Chen, Z., On nonsingularity of circulant matrices, *Linear Algebra Appl.,* 2021, vol. 612, pp. 162–176. https://doi.org/10.1016/j.laa.2020.12.010
11. Alaev, P.E. and Selivanov, V.L., Fields of algebraic numbers computable in polynomial time. I, *Algebra Logic,* 2020, vol. 58, pp. 447–469.
12. Alaev, P.E. and Selivanov, V.L., Fields of algebraic numbers computable in polynomial time. II, *Algebra Logic,* 2022, vol. 60, pp. 349–359.
13. Harris, J. and Tu, L.W., On symmetric and skew-symmetric determinantal varieties, *Topology,* 1984, vol. 23, no. 1, pp. 71–84. https://doi.org/10.1016/0040-9383(84)90026-0
14. Harris, J., *Algebraic Geometry,* New York: Springer, 1992. https://doi.org/10.1007/978-1-4757-2189-8
15. Rubei, E., Affine subspaces of matrices with constant rank, *Linear Algebra Appl.,* 2022, vol. 644, no. 1, pp. 259–269. https://doi.org/10.1016/j.laa.2022.03.002
16. Seliverstov, A.V., Binary solutions to large systems of linear equations, *Prikl. Diskretnaya Mat.,* 2021, no. 52, pp. 5–15. https://doi.org/10.17223/20710410/52/1
17. Kuzyurin, N.N., Polynomial-average algorithm in integer linear programming, *Sib. Zh. Issled. Oper.,* 1994, vol. 1, no. 3, pp. 38–48.
18. Kuzyurin, N.N., An integer linear programming algorithm polynomial in the average case, *Discrete Analysis and Operations Research,* Korshunov, A.D., Ed., Dordrecht: Springer, 1996, vol. 355, pp. 143–152. https://doi.org/10.1007/978-94-009-1606-7

19. Pan, Y. and Zhang, F., Solving low-density multiple subset sum problems with SVP oracle, *J. Syst. Sci. Complexity,* 2016, vol. 29, pp. 228–242. https://doi.org/10.1007/s11424-015-3324-9

20. Rybalov, A.N., On the generic complexity of the subset sum problem for semigroups of integer matrices, *Prikl. Diskretnaya Mat.,* 2020, no. 50, pp. 118–126. https://doi.org/10.17223/20710410/50/9

21. Rybalov, A.N., On the generic complexity of the occurrence problem for semigroups of integer matrices, *Prikl. Diskretnaya Mat.,* 2022, no. 55, pp. 95–101. https://doi.org/10.17223/20710410/55/7

22. Seliverstov, A.V., Heuristic algorithms for recognition of some cubic hypersurfaces, *Program. Comput. Software,* 2021, vol. 47, no. 1, pp. 50–55.

23. Minc, H., (0, 1)-matrices with minimal permanents, *Isr. J. Math.,* 1973, vol. 15, pp. 27–30. https://doi.org/10.1007/BF02771770

24. Seliverstov, A.V. and Lyubetsky, V.A., About forms equal to zero at each vertex of a cube, *J. Commun. Technol. Electron.,* 2012, vol. 57, no. 8, pp. 892–895. https://doi.org/10.1134/S1064226912080049

25. Schwartz, J.T., Fast probabilistic algorithms for verification of polynomial identities, *J. ACM,* 1980, vol. 27, no. 4, pp. 701–717. https://doi.org/10.1145/322217.322225

26. Harris, C.R., Millman, K.J., van der Walt, S.J., et al., Array programming with NumPy, *Nature,* 2020, vol. 585, no. 7825, pp. 357–362. https://doi.org/10.1038/s41586-020-2649-2

27. Chen, Y.A. and Gao, X.S., Quantum algorithm for Boolean equation solving and quantum algebraic attack on cryptosystems, *J. Syst. Sci. Complexity,* 2022, vol. 35, pp. 373–412. https://doi.org/10.1007/s11424-020-0028-6

Translated by Yu. Kornienko